



SERIA INFORMATICA

R. M. Hristev



**GHIDUL
UTILIZATORULUI SI
PROGRAMATORULUI
SPECTRUM**

EDITURA APH

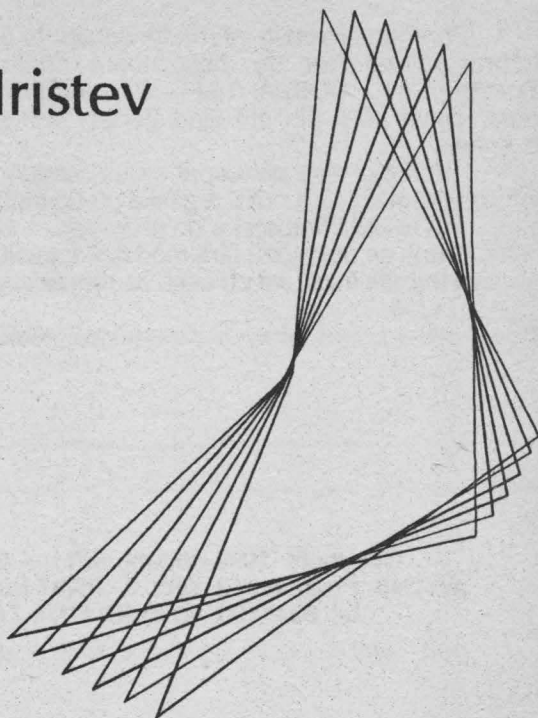


BUCURESTI 1992



SERIA INFORMATICĂ

R. M. Hristev



**GHIDUL
UTILIZATORULUI SI
PROGRAMATORULUI
SPECTRUM**

EDITURA APH



București 1992

Prin numeroasele informații conținute cartea se adresează tuturor utilizatorilor de calculatoare compatibile Sinclair ZX Spectrum (HC, COBRA, TIM-S, CIP, JET) conținând numeroase date, peste 90% din ele fiind pentru prima oară publicate în România

Volumul este conceput ca un manual de utilizare în care informațiile necesare pot fi regăsite cu ușurință.

În sintaxa limbajelor de programare simbolurile scrise cu litere **aldine** se utilizează fără modificări; cuvintele scrise cu litere *cursive* trebuie înlocuite cu ceea ce reprezintă ele

AUTORUL

Copyright 1992 Editura APH — SRL
str. Cap. Preda nr. 12, sect. 5, 76437 București 69,
tel. 80.20.30, 80.93.97, 8074.77

Redactor, tehoredactor, coperta: R. M. Hristev

*Bun de tipar 20.8.1992. Apărut 1992. Format 70x100/16
Coli de tipar 11.5*

Tiparul executat la Tipografia I. I. R. U. C.

1.1 RUTINELE RESTART

0000 Start

Rutină de inițializare, DE încărcat cu FFFF, adresa maximă posibilă pentru memoria RAM

0008 Eroare

Rutină de tratare a erorilor, RST 08 este urmat de DEFB n unde n=codul-erorii—1. Codul erorii: 0 pentru OK, ..., 1B pentru Tape loading error.

0010 Tipărire/Afișare caracter

Rutina de tipărire/afișare a unui caracter la canalul adresat de CURCHL.

La intrare: în A codul caracterului, celelalte informații în variabilele de sistem.

0018 Culege caracter

Conținutul locației adresate de CH-ADD este adus în HL, return dacă caracterul este afișabil, dacă nu, CH-ADD este incrementat și se repetă testul.

0020 Culege următorul caracter

Rutină utilizată la parcurgerea liniilor de program BASIC de către interpretor.

0028 Calcul în virgulă flotantă

Rutina de calcul în virgulă flotantă, fiecare apel RST 28 este urmat de o serie de DEFB care se termină cu DEFB 38, fiecare byte definind un cod de operație.

0030 BC spații

Rutina creează locații libere în zona de lucru, utilizează WORKSP care conține adresa de start a spațiului de lucru.

La intrare: BC conține numărul locațiilor

0038 Întrerupere mascabilă

Rutină de tratare a întreruperilor mascabile, se incrementează ceasul de timp real FRAMES la fiecare 20ms și se citește tastatura..

0053 Continuare RST 08

Adresa de întoarcere a interpretorului este o locație DEFB care conține codul erorii întâlnite. DEFB este transferat în ERR-NR.. Stiva este ștersă. Salt pentru resetarea stivei-calculator.

0066 Întrerupere nemascabilă

Rutină de tratare a întreruperilor nemascabile; neutilizată în Spectrum original, aceasta conține o eroare: instrucțiunea JR NZ trebuie înlocuită cu JR Z (utilizează NMIADD)

0074 CH-ADD+1

Rutina incrementează CH-ADD; încarcă A cu locația adresată de CH-ADD+1

007D Sare peste

Rutina testează dacă codul din A este afișabil; setează HL și CH-ADD în concordanță cu A.

0095 Tabela cuvintelor cheie

Tabela cuvintelor-cheie BASIC după care se face afișarea lor; cuprinde: ?, RND, ... RETURN, COPY. Bitul 7 al ultimei litere al fiecărui cuvânt este setat.

0205 Tabela tastelor (1)

codurile simbolurilor obținute prin: cursor L și CS; cuprinde: B, H, ...Q, A.

022C Tabela tastelor (2)

codurile simbolurilor obținute prin: cursor E, taste literare; cuprinde: READ, BIN, ... STR\$, LN.

0246 Tabela tastelor (3)

codurile simbolurilor obținute prin: cursor E, taste literare și SS; cuprinde: ~, BRIGHT, ..., [, BEEP.

0260 Tabela tastelor (4)

codurile de control obținute prin: taste numerice și CS; cuprinde: DELETE, EDIT, ..., cursor-dreapta, GRAPHICS.

026A Tabela tastelor (5)

codurile simbolurilor obținute prin: taste literare și SS; cuprinde: STOP, *, ..., AND, :

0284 Tabela tastelor (6)

codurile simbolurilor obținute prin: cursor E, taste numerice și SS; cuprinde: FORMAT, DEF FN, ..., POINT, CAT.

1.2 RUTINELE DE TASTATURĂ

028E Scanarea tastaturii

Rutina principală de citire a tastaturii de 40 de taste.

La ieșire: în E valoare de la 00 la 27, codul tastei sau FF, nici o tastă apasată
 în D codul tastei shift apăsate sau FF (în DE este FFFF dacă nici o
 tastă nu a fost apăsată)

Portul tastaturii (în C): FE

02BF Tastatura

Rutina de citire a tastaturii apelată la fiecare 20ms, decodifică tasta apasată, pune codul în LAST-K și bitul 5 al FLAGS este setat. Se utilizează 2 sisteme de variabile KSTATE0 ... KSTATE3 și KSTATE4 ... KSTATE7. Un set devine disponibil pentru o nouă tastă numai după 1/10s adică 5 apelări ale rutinei.

0308 Repetarea tastei

Este o subrutină a Tastatura (02BF); o tastă se consideră repetată prima oară după o perioadă REPDEL (normal 0.7s) și ulterior după o perioadă REPPER (normal 0.1s).

031E K-test

Rutina testează dacă nu a fost apăsată nici o tastă sau numai o tastă shift: dacă da return, dacă nu calculează codul tastei.

0333 Decodarea tastei

La intrare: în E codul tastei
 în D valoarea FLAGS
 în C modul cursor

La ieșire: în A codul simbolului, pentru tasta sau ansamblul de taste apăsat

1.3 RUTINE DIFUZOR

03B5 Beeper

Rutina care activează și dezactivează difuzorul, repetat, astfel încât se obține sunetul.

La intrare: în DE produsul dintre frecvența sunetului și durata lui (în s)
 în HL contor pentru bucla de timp = $3.5 \cdot 106 / 8 / f \cdot 30.125$

03F8 BEEP

La intrare: pe stiva-calculator, în vîrf, se găsește "înălțimea" notei și sub ea "durata" notei

Rutina efectuează verificări asupra "înălțimii" și "duratei" sunetului și pregătește parametrii pentru BEEPER pe care o apelează.

046E Tabela frecvențelor

261.63	do	369.99	fa#
277.18	do#	392	sol
293.66	re	415.30	sol#
311.13	re#	440	la
329.63	mi	466.16	la#
349.23	fa	493.88	si

Observație: La fiecare salt la octava superioară respectiv inferioară frecvențele se dublează, respectiv înjumătățesc.

04AA Neutilizat

1.4 RUTINELE CASOTOFON

Pentru toate rutinele:

DE conține lungimea blocului de bytes

IX conține adresa de început a blocului de bytes

A conține 00 pentru header, FF pentru program/date

04C2 SA-bytes

Rutină de salvare a blocurilor de bytes.

053F SA/LD-Ret

Rutină de retur după o operație de salvare sau încărcare.

0556 LD-bytes

Rutină de citire/verificare a blocurilor de bytes.

05E3 LD-edge

La intrare: în B constanta de timp

în C tipul de margine

La iesire: în F: C=0 la eroare, Z=0 la Break

0605 SAVE, LOAD, VERIFY, MERGE

Rutina este comună comenzilor SAVE, LOAD, VERIFY și MERGE.

TADDR este E0 pentru SAVE E1 pentru LOAD

E2 pentru VERIFY E3 pentru MERGE

Header-ul este format în zona de lucru, astfel:

locația: IX+00 conține tipul

IX+01 până la IX+0A conține numele (FF în IX+01 dacă este nul)

IX+0B și IX+0C conțin lungimea în bytes

IX+0D până la IX+10 conțin informații dependente de tipul blocului

07CB VERIFY

Rutina VERIFY, datele sînt numai comparate nu încărcate.

0802 LOAD date

Rutină comună pentru LOAD și VERIFY.

0808 LOAD

Rutină de încărcare a unui program BASIC și a datelor sale.

08B6 MERGE

Rutina efectuează: a) încărcarea unui bloc de bytes

b) suprapune liniile BASIC

c) suprapune variabilele programelor BASIC

092C MERGE linie sau variabilă

La intrare: în F: C=0 suprapunerea unei linii BASIC

C=1 suprapunerea unei variabile

Z=0 adăugare

în HL adresa începutului noii intrări

în DE adresa unde se face înlocuirea

0970 SAVE

Rutină de salvare a programelor.

09A1 Mesaje-bandă magnetică

Bitul 7 al ultimului caracter este setat.

09A1	DEFB	80
09A2	DEFM	"Start tape then press any key"
09C1	DEFM	carriage return, "Program:"
09CB	DEFM	carriage return, "Number array:"
09DA	DEFM	carriage return, "Character array:"
09EC	DEFM	carriage return, "Bytes:"

1.5 RUTINE ECRAN

09F4 Print-out

Rutina print principală.

La intrare: în A: codul caracterului (afișabil sau de control)
 în variabilele de sistem: parametrul de poziție (rutina 0B03)

0A11 Tabela caracterelor de control

4E	PRINT comma	4F	neutilizat
57	Edit	5F	INK control
10	cursor stînga	5E	PAPER control
29	cursor dreapta	5D	FLASH control
54	cursor jos	5C	BRIGHT control
53	cursor sus	5B	INVERSE control
52	DELETE	5A	OVER control
37	Enter	54	AT control
50	neutilizat	53	TAB control

0A23 Cursor stînga

La intrare: în B linia curentă
 în C coloana curentă

Observație: instrucțiunea LD A, 18 trebuie înlocuită cu LD A, 19.

0A3D Cursor dreapta

Analog cursor stînga (rutina 0A23).

Observație: returul ar trebui efectuat la adresa 0ADC.

0A4F Carriage return

Se efectuează test scroll? înainte de decrementarea numărului de linie.

0A5F Print comma

A este încărcat cu 00h pentru TAB 0 sau 10h pentru TAB 16.

0A69 Afișează ?

Dacă se încearcă afișarea unui caracter neafișabil atunci se afișează ?.

0A6D Caracter de control al operanzilor

INK ... OVER au un unic operand; AT și TAB au 2 operanzi

Rutina încarcă caracterul de control în TVDATA-inf, primul operand în TVDATA-sup sau în A dacă este operand unic și cel de al 2-lea operand în A.

0AD9 Afișează caracter

Un caracter este afișat prin apelarea rutinei 0B24 urmată de rutina 0ADC.

0ADC Memorează poziția

Noua poziție linie, coloană și adresa pixel sînt încărcate în variabilele de sistem

0B03 Aduce poziția

Poziția curentă este adusă din sistemul de variabile pentru tipărire/afișare

0B24 Afișează un caracter

Rutină folosită pentru afișarea caracterelor: matrice 8*8 pixeli

La intrare: în DE adresa de bază a formei caracterelor (normal 3D00)
 în HL adresa destinație
 în BC linia și coloana curentă

0BDB Actualizează atributul

Rutina actualizează atributul de culoare al caracterului, utilizind valoarea anterioară și ATTR-T, MASK-T, P-FLAG.

0C0A Afișează mesaj

Rutina afișează cuvinte-cheie BASIC sau mesaje.

0C3B Afișare repetată

Rutină utilizată la afișarea repetată a unui caracter

0C55 Test scroll

Rutina testează necesitatea unui scroll în cazurile:

- a) la întâlnirea unui carriage return
- b) AT în INPUT
- c) linia curentă plină

La intrare: în B: numărul liniei pentru care se dorește testul

Observație: La 0CF8 mesajul scroll? ,cu marker inițial și bitul 7 al ultimului caracter setat.

0D4D Copierea temporară a atributelor de culoare

Rutină utilizată cînd atributele permanente trebuie copiate în variabilele de sistem.

0E0D Comanda CLS

Rutina șterge tot ecranul: resetează pixelii și setează atributele conform ATTR-P apoi prelucrează partea de jos a ecranului (normal liniile 22 și 23).

0DAF Șterge ecranul

Această rutină este utilizată de:

- a) comanda CLS
- b) rutina principală de execuție
- c) rutina de listare

0DD9 Clear set

La intrare: în BC linia și coloana ecran
 sau în C coloana din tamponul de imprimantă

Se calculează adresa primului bit al caracterului.

0DFE Scroll

Rutina efectuează un scroll pentru numărul de linii din registrul B.

0E44 Șterge linii

Rutina șterge numărul de linii din registrul B, din partea de jos a ecranului

0E88 Atribute

Rutina efectuează:

- a) pentru o adresă din ecran adresa atributului este returnată în HL (adresa de intrare: cea de-a 9-a linie a caracterului)
- b) pentru o linie dată în B este returnată adresa de start a liniei în BC

0E9B Adrese

Rutina calculează, pentru o linie dată în B, adresa din ecran.

0EAC Copy

Rutina copiază rînd cu rînd ecranul la imprimantă (176 rînduri)

0EDC Copy buffer

Rutina este apelată la fiecare golire a tamponului de imprimantă .

0EDF Șterge tamponul de imprimantă

0EF4 Copiază linie la imprimantă

La intrare: în HL adresa de bază a primului rînd al liniei
în B numărul rîndului

0F2C Editor

Rutina este utilizată de:

- a)rutina principală la intrarea unei noi linii BASIC
- b)comanda INPUT

0F81 Aadaugă un caracter

Rutina adaugă codul unui caracter în linie EDIT sau INPUT.

0FA0 Tabela tastelor editor

09	Edit	70	Delete
66	cursor stînga	7E	Enter
6A	cursor dreapta	CF	SS
50	cursor jos	D4	Graphics
B5	cursor sus		

0FA9 Tasta Edit

Rutina este utilizată în:

- a)modul Edit: prin apăsarea tastei Edit este pregătită o linie pentru editare
- b)INPUT: șterge răspunsul curent și alocă spațiu pentru unul nou

1007 Editare-cursor stînga

Rutina se termină în Editare-cursor dreapta (rutina 100C)

100C Editare-cursor dreapta

1015 Editare-delete

Cursorul se mișcă în stînga

101E Editare-ignoră 2 coduri

1024 Editare-Enter

Variabilele de sistem, de tratare a erorilor, sînt actualizate și salt la rutinele de tratare ale erorilor dacă este cazul.

1031 Editare margine

Adresa cursorului din HL este decrementată dacă cursorul nu se află la marginea liniilor dar astfel încît cursorul nu este pus între caracterele de control și parametrii lor.

1059 Editare-cursor sus

1076 Editare-SS și Graphics

Rutina tratează utilizarea codurilor SS și Graphics.

107F Editare-erori

Rutină de tratare a erorilor la editare.

1097 Șterge zona de lucru

Rutina șterge zona de editare sau spațiul de lucru.

10A8 Input de la tastatură

Rutina returnează codul ultimei taste apăsate.

111D Copiere ecran-jos

Rutina este apelată pentru afișarea unei linii din zona de editare sau în cazul unui INPUT în partea de jos a ecranului.

1190 Set HL și DE

La ieșire: în HL adresa primei locații
 în DE adresa ultimei locații
 din zona de editare sau spațiul de lucru.

11A7 Retrage-virgulă flotantă

Rutina plasează numerele în virgulă flotantă, din linia BASIC în curs de interpretare pe stiva-calculator.

1.6 RUTINE DE EXECUȚIE**11B7 Comanda NEW****11CB Inițializare**

Rutina de start sau NEW, inițializează variabilele de sistem și afișează mesajul "@1982 Sinclair Research Ltd."

12A2 Rutina principală de execuție

Rutina controlează editarea, execuția comenzilor directe și producerea mesajelor de eroare.

1391 Tabela mesajelor de eroare

1391 DEFB 80 marker inițial
 1392 Eroare-0 OK
 1394 Eroare-1 NEXT without FOR

...

1516 Eroare-Q Parameter error
 1539 ", " (virgulă și blank)
 1539 "@ 1982 Sinclair Research Ltd."

1555 Eroare-G**155D Adaugă o linie BASIC**

Rutina introduce o linie nouă de program în zona programului BASIC. Noua linie o înlocuiește pe cea veche dacă este cazul. O linie program care conține doar numărul de linie nu este introdusă.

15AF Tabela de informații pentru canale

Există 4 canale inițiale de comunicare:

K pentru tastatură	R pentru spațiul de lucru
S pentru ecran	P pentru imprimantă

Pentru fiecare canal: adresa rutinei out, adresa rutinei in.

15AF	DEFB F4, 09	rutina 09F4
	DEFB AB, 10	rutina 10AB
	DEFB 4B	canal K
15B4	DEFB F4, 09	rutina 09F4
	DEFB C4, 15	rutina 15C4

	DEFB 5	canal S
15B9	DEFB 81, 0F	rutina 0F81
	DEFB C4,15	rutina 15C4
	DEFB 52	canal R
15BE	DEFB F4, 09	rutina 09F4
	DEFB C4,15	rutina 15C4
	DEFB 50	canal P
15C3	DEFB 80	marcator de sfârșit

15C4 Eroare-J**15C6 Tabela de informații stream**

15C6	DEFB 01, 00	stream FD	canal K
15C8	DEFB 06, 00	stream FE	canal S
15CA	DEFB 0B, 00	stream FF	canal R
15CC	DEFB 01, 00	stream 00	canal K
15CE	DEFB 01, 00	stream 01	canal K
15D0	DEFB 06, 00	stream 02	canal S
15D2	DEFB 10, 00	stream 03	canal P

15D4 Așteaptă o tastă

Rutină de control pentru rutina curentă de input.

15E6 Adresa input

Rutina salvează regiștrii și introduce în HL adresa de input.

15EF Rutina principală de afișare

La intrare: în A codul caracterului

1601 Deschide canal

La intrare: în A număr de stream valid (normal între FD și 03); în funcție de stream va fi deschis canalul respectiv

1615 Flags canale

Rutina setează variabilele de sistem conform canalelor utilizate.

162D Tabela adrese canale

162D	DEFB 4B,06	canal K	offset 06	adresa 1634
162F	DEFB 53,12	canal S	offset 12	adresa 1642
1631	DEFB 50,1B	canal P	offset 1B	adresa 164D
1633	DEFB 00	marcator de sfârșit		

1634 Flags canal K**1642 Flags canal S****164D Flags canal P****1652 Deschide fereastră**

Rutina deschide o fereastră în memorie.

La intrare: în HL adresa locației de după fereastră
în BC lungimea ferestrei

Atenție: dacă se dorește eliberarea unei singure locații, atunci adresa de început a rutinei este 1652, dacă nu atunci 1655

La ieșire: în HL adresa locației dinaintea ferestrei
în DE adresa ultimei locații din fereastră

Observație: noile locații păstrează vechile valori, fereastra este de la HL+1 pînă la DE, dar în unele rutine se consideră de la HL+2 la DE+1.

1664 Pointeri

Rutina actualizează variabilele de sistem, dacă este cazul, cînd o fereastră de memorie este deschisă sau modificată..

La intrare: în BC numărul locațiilor
în HL adresa locației dinaintea poziției

168F Aduce număr de linie

La intrare: în HL adresă locație

La ieșire: în DE numărul liniei care începe la HL

Dacă HL nu reprezintă adresa primului byte a numărului de linie atunci la ieșire DE=0 (168F DEFB 00, 00 număr de linie 0)

169E - Rezervă

Rutină utilizată normal de RST 30, fereastra este creată între zona de lucru și stiva-calculator.

16B0 Set minim

Rutina resetează zona de editare, spațiul de lucru și stiva-calculator la mărimile minime.

16D4 Apelează editorul de linie

16DB Indexator

Rutină folosită pentru parcurgerea tabelor.

16E5 Comanda CLOSE#

Rutină pentru închiderea stream-urilor (00 la 03 nu pot fi închise)

1716 Tabela de adrese close stream

1716	DEFB 4B,05	canal K	offset 05	adresa 171C
1718	DEFB 53,03	canal S	offset 03	adresa 171C
171A	DEFB 50,01	canal P	offset 01	adresa 171C

Observație: nu există marcator de sfârșit

171C Închide stream

171E Date stream

La intrare: pe stiva-calculator se afișează stream-ul

La ieșire: în BC date stream

1736 Comanda OPEN#

Rutină pentru deschiderea stream-urilor. Canale valide K, k,S, s, P, p.

175D Subrutina OPEN#

Subrutină a rutinei 1736.

177A Tabela de adrese de deschidere stream

177A	DEFB 4B,06	canal K	offset 06	adresa 1781
177C	DEFB 53,08	canal S	offset 08	adresa 1785
1780	DEFB 50,0A	canal P	offset 0A	adresa 1789
1780	DEFB 00	marcator de sfârșit		

1781 Deschide K

1785 Deschide S

1789 Deschide P

1793 Comenzi CAT, ERASE, FORMAT, MOVE

Rutinele acestor comenzi se află în ZX Interface 1 astfel încât utilizarea acestora fără interfața respectivă generează eroare O.

1795 Comenzile LIST și LLIST

17F5 Punct de intrare LLIST

17F9 Punct de intrare LIST

1855 Afișează o linie BASIC

La intrare: în HL adresa primei locații a liniei

18B6 Number

Dacă $A=0E$ atunci HL este incrementat cu 6 (sare peste un număr în virgulă flotantă).

18C1 Afișează un caracter tip FLASH

Rutină apelată pentru afișarea cursorului și cursorului de eroare.

18E1 Afișează cursorul

Rutina afișează cursorul K, L, C, E sau G.

190F Culege linie BASIC

La intrare: în HL adresa S-TOP sau E-PPC și le modifică cu numărul liniei următoare

1925 Afișează linie BASIC

Rutina afișează caracterele sau cuvintele-cheie dintr-o linie BASIC.

196E Adresa liniei BASIC

La intrare: în HL numărul liniei

La ieșire: în HL adresa liniei sau a primei linii de după numărul dat
în DE adresa liniei anterioare

1980 Compară numere de linie BASIC

La intrare: în BC număr de linie BASIC
în HL adresa unei linii BASIC

La ieșire: în F: C=1 pentru egalitate

1988 Neutilizat**198B Caută toate instrucțiunile**

La intrare: în D numărul instrucțiunii dintr-o linie BASIC
sau în E codul unui cuvânt-cheie
în HL adresa locației de unde începe căutarea

La ieșire: în HL adresa locației dinaintea instrucțiunii

19B8 Următoarea linie/variabilă

Rutina caută următoarea linie BASIC sau următoarea variabilă BASIC (există 6 tipuri)

La intrare: în HL adresa curentă

La ieșire: în HL adresa următoare
în DE adresa curentă

19DD Diferența

Rutina calculează diferența cu transport între HL și DE, returnează rezultatul în BC apoi reface HL și DE și le inversează.

19E5 Recuperează

Rutina mută blocuri de bytes.

Punct de intrare 19E5: La intrare: în DE adresa sursă
în HL adresa destinație

Punct de intrare 19E8: La intrare: în DE adresa sursă
în HL adresa destinație
în BC numărul de bytes

19FB Editare-număr linie

La ieșire: în BC numărul liniei din zona editare, dacă nu există atunci BC=0
(comandă directă)

1A1B Afișare număr linie

Punct de intrare 1A1B: afișează numărul de linie din BC

Punct de intrare 1A28: afișează numărul de linie de la adresa HL

1.7 INTERPRETORUL BASIC

1A48 Tabela offset pentru comenzile BASIC

	DEFB	adresă	comandă	DEFB	adresă	comandă	
1A48	B1	1AF9	DEF FN	1A61	94	1AF5	BORDER
1A49	CB	1B14	CAT	1A62	56	1AB8	CONTINUE
1A4A	BC	1B06	FORMAT	1A63	3F	1AA2	DIM
1A4B	BF	1B0A	MOVE	1A64	41	1AA5	REM
1A4C	C4	1B10	ERASE	1A65	2B	1A90	FOR
1A4D	AF	1AFC	OPEN#	1A66	17	1A7D	GO TO
1A4E	B4	1B02	CLOSE#	1A67	1F	1A86	GO SUB
1A4F	93	1AE2	MERGE	1A68	37	1A9F	INPUT
1A50	91	1AE1	VERIFY	1A69	77	1AE0	LOAD
1A51	92	1AE3	BEEP	1A6A	44	1AAE	LIST
1A52	95	1AE7	CIRCLE	1A6B	0F	1A7A	LET
1A53	98	1AEB	INK	1A6C	59	1AC5	PAUSE
1A54	98	1AEC	PAPER	1A6D	2B	1A98	NEXT
1A55	98	1AED	FLASH	1A6E	43	1AB1	POKE
1A56	98	1AEE	BRIGHT	1A6F	2D	1A9C	PRINT
1A57	98	1AEF	INVERSE	1A70	51	1AC1	PLOT
1A58	98	1AF0	OVER	1A71	3A	1AAB	RUN
1A59	98	1AF1	OUT	1A72	6D	1ADF	SAVE
1A5A	7F	1AD9	LPRINT	1A73	42	1AB5	RANDOMIZE
1A5B	81	1ADC	LLIST	1A74	0D	1A81	IF
1A5C	2E	1A8A	STOP	1A75	49	1ABE	CLS
1A5D	6C	1AC9	READ	1A76	5C	1AD2	DRAW
1A5E	6E	1ACC	DATA	1A77	44	1ABB	CLEAR
1A5F	70	1ACF	RESTORE	1A78	15	1A8D	RETURN
1A60	48	1AA8	NEW	1A79	5D	1AD6	COPY

Adresele generate sînt adresele din tabela de parametri pentru comenzile respective.

1A7A Tabela de parametri

Reprezintă tabela parametrilor comenzilor BASIC de la LET (1A7A) pîna la CAT (1B14), adresele fiind date de tabela 1A48. Pentru fiecare comandă există 1 pînă la 8 bytes, offset de sintaxă, care se citesc în modul următor:

Dacă byte-ul este între 00 și 0B atunci (byte de sintaxă):

- 00 nu mai urmează nici un operand
- 01 utilizat în LET, urmează o variabilă
- 02 utilizat în LET, urmează o expresie numerică
- 03 urmează o expresie numerică, în lipsă se consideră 0
- 04 urmează o variabilă al cărei nume este un singur caracter
- 05 urmează o serie de articole
- 06 urmează o expresie numerică
- 07 urmează coduri de culoare
- 08 urmează 2 expresii numerice separate de virgulă
- 09 analog 08, expresiile pot fi precedate de coduri de culoare
- 0A urmează o expresie șir
- 0B pentru rutinele de bandă magnetică

byte-ul 00, 03 sau 05 este urmat de o adresă a comenzii de executat. Ceilalți bytes reprezintă coduri caracter sau cuvînt-cheie.

1B17 Interpretorul

Rutina verifică sintaxa liniei BASIC.

1B28 Bucla de comenzi

Rutina execută comenzile dintr-o linie BASIC.

1B6F Separator

Dacă separatorii necesari nu sînt găsiți se generează eroare C.

1B76 Întoarcere după o comandă

După execuția unei comenzi se efectuează o întoarcere în acest punct.

1B8A Run

Punct de intrare pentru rularea unei linii din zona de editare sau pentru verificarea sintaxei dintr-o linie din zona de editare care are mai multe comenzi.

1B9E Linie BASIC nouă (1)

La un salt în program, rutina calculează adresa noli linii.

1BB2 Comanda REM

1BB3 Sfirșit de linie BASIC

La verificarea sintaxei: return

La rulare: actualizează NXTLIN

1BBF Linie BASIC nouă (2)

La ieșire: în PPC numărul noli linii BASIC

în HL adresa

1BD1 Linie BASIC nouă (3)

La intrare: în HL adresa locației după sfirșitul liniei noi

în DE adresa locației dinaintea primului caracter al liniei

Se actualizează NXTLIN.

1BEE Caută sfirșitul liniei BASIC

1BF4 Comanda următoare

Dacă Enter atunci comanda următoare este pe linia următoare.

Dacă : atunci comanda următoare este pe aceeași linie, în continuare.

Altfel eroare.

1C01 Tabela offset pentru bytes de sintaxă

DEFB adresă			offset	DEFB adresă			offset
1C01	0F	1C10	00	1C07	7B	1C82	06
1C02	1D	1C1F	01	1C08	8E	1C96	07
1C03	4B	1C4E	02	1C09	71	1C7A	08
1C04	09	1C0D	03	1C0A	B4	1CBE	09
1C05	67	1C6C	04	1C0B	81	1C8C	0A
1C06	0B	1C11	05	1C0C	CF	1CD8	0B

1C0D Sintaxă offset 03

1C10 Sintaxă offset 00

1C11 Sintaxă offset 05

1C16 Salt la rutina de execuție a comenzii

1C1F Sintaxă offset 01

1C22 Asignarea variabilelor DEST și STRLEN

1C4E Sintaxă offset 02

1C56 Asignează o valoare

1C6C Sintaxă offset 04**1C79 Evaluarea expresiei numerice/șir**

Rutină de evaluare a unei expresii numerice sau șir

1C7A Sintaxă offset 08**1C82 Sintaxă offset 06****1CBC Sintaxă offset 0A****1C96 Sintaxă offset 07****1CBE Sintaxă offset 09**

Rutină utilizată de PLOT, DRAW și CIRCLE.

1CDB Sintaxă offset 0B**1CDE Culege un număr**

Rutina evaluează o expresie numerică; dacă nu există returnează 0.

1CEE Comanda STOP**1CF0 Comanda IF****1D03 Comanda FOR****1D86 Caută în program**

Rutina caută instrucțiuni DATA, DEF FN sau NEXT în program (cele care vor fi utilizate).

La intrare: în E codul cuvântului-cheie
în HL adresa de început a căutării

1E27 Comanda DATA**1E42 Comanda RESTORE****1E4F Comanda RANDOMIZE**

La intrare: în BC operandul, dacă BC=0 se utilizează FRAMES

La ieșire: în SEED, BC sau FRAMES

1E5F Comanda CONTINUE**1E67 Comanda GO TO****1E7A Comanda OUT****1E80 Comanda POKE****1E86 2 parametri**

Rutină utilizată de POKE.

Numărul din vârful stivei-calculator este comprimat în A, în complement față de 2.

Numărul al 2-lea din stiva-calculator este comprimat în BC.

1E94 Găsește întregul (1)

Numărul din vârful stivei-calculator este comprimat în A.

1E99 Găsește întregul (2)

Numărul din vârful stivei-calculator este comprimat în BC.

1EA1 Comanda RUN**1EAC Comanda CLEAR****1EED Comanda GO SUB****1F05 Test fereastră memorie**

Testează dacă există spațiu suficient în memorie pentru următoarea operație.

1F1A Memorie liberă

Rutina calculează cât spațiu disponibil există în memorie (echivalent cu PRINT 65536-USR 7962).

1F23 Comanda RETURN

1F3A Comanda PAUSE**1F54 Tasta BREAK**

Rutină pentru citirea tastei Break

La ieşire: în F: C=0 numai dacă SS și Break au fost apăstate

1F60 Comanda DEF FN**1FC3 Unstack-z**

Rutină utilizată pentru întoarcere rapidă dintr-o rutină la verificarea sintaxei.

1FC9 Comenzile LPRINT și PRINT**1FF5 Print carriage return****1FFC Caractere de control ale PRINT, PRINT și INPUT**

Subrutină a comenzilor PRINT, LPRINT și INPUT care manipulează AT și TAB.

2045 Sfirşitul afişării**204E Poziția de afişare**

Rutina manipulează caracterele de control ale afişării (: , ').

2070 Utilizare stream

Rutină utilizată când se folosesc diverse stream-uri.

2089 Comanda INPUT**21B9 Aassignare INPUT**

Rutină utilizată la verificarea sintaxei și la atribuire de valori variabilelor.

21D6 Utilizarea canalului K

La ieşire: în F: Z=1 la utilizarea tastaturii

21E1 Culorile de afişare (1)

Rutină de manipulare INK, PAPER și OVER.

2211 Culorile de afişare (2)

Rutină de actualizare a sistemului de variabile.

226C Culorile de afişare (3)

Rutină de manipulare FLASH și BRIGHT.

2294 Comanda BORDER**22AA Adresa pixel**

Rutină utilizată de funcția POINT și comanda PLOT.

La intrare: în BC coordonatele pixel-ului

La ieşire: în HL adresa byte-ului în care se găseşte pixel-ul
în A poziția pixel-ului din byte

22CB Funcția POINT

La intrare: adresa pixel-ului pe stiva-calculator

La ieşire: în A: 0 dacă pixel-ul are culoare PAPER
1 dacă pixel-ul are culoare INK

22DC Comanda PLOT

La intrare: adresa pixel-ului pe stiva-calculator; acesta este afişat conform
INVERSE și OVER din P-FLAG

2307 Stiva-calculator în BC

Rutina încarcă în BC 2 numere în virgulă flotantă de pe stiva-calculator, acestea trebuie să fie în intervalul 00, FF.

2314 Stiva-calculator în A

Rutina încarcă în A un număr în virgulă flotantă de pe stiva-calculator, acesta trebuie să fie în intervalul 00, FF.

2320 Comanda CIRCLE

La intrare: în virful stivei-calculator: R raza cercului
pe stivă, sub R, este coordonata Y
pe stivă, sub Y, este coordonata X

2382 Comanda DRAW

Dacă sînt dați 2 parametri X și Y se trasează un segment de dreaptă din poziția X0, Y0 din COORDS pînă în poziția X0+X, Y0+Y.

Dacă sînt dați 3 parametri X, Y, G se trasează un arc de cerc, între aceleași puncte, de lungime G radiani, în sens trigonometric (G>0) sau orar (G<0).

Ordinea pe stiva-calculator, de la virf la bază: X, Y sau X, Y, G.

247D Parametrii inițiali

Rutină utilizată de comenzile CIRCLE și DRAW pentru pregătirea parametrilor inițiali de trasare.

247B DRAW

Rutină utilizată de comanda DRAW (rutina 2382).

La intrare: X și Y pe stiva-calculator

Rutina trasează efectiv linia între X0, Y0 și X0+X, Y0+Y.

1.8 EVALUATORUL DE EXPRESII

24FB Scanare

Rutina evaluează expresia următoare.

La ieșire: un rezultat pe 5 bytes pe stiva-calculator

Dacă rezultatul este număr în virgulă flotantă atunci în FLAGS bit6 = 1

Dacă rezultatul este un șir atunci:

primul byte:	nespecificat
bytes 2,3:	adresa șirului
bytes 4,5:	lungimea șirului

2530 Sintaxa-Z

în FLAGS: bit7=0 înseamnă execuție

bit7=1 înseamnă verificarea sintaxei

2535 Scanare SCREEN\$

Rutină utilizată de funcția SCREEN\$.

La intrare: pe stiva-calculator coordonatele X, Y

La ieșire: pe stiva-calculator codul caracterului

Rutina utilizează tabela caracterelor, adresată de CHARS.

2596 Tabela funcțiilor de scanare

nume	adresă	cod	offset	adresa rutinei
"	2596	22	1C	25B3
()	2598	28	4F	25E8
zecimal	259A	2E	F2	268D
număr, 1 caracter	259C	2B	12	25AF
FN	259E	A8	56	25F5
RND	25A0	A5	57	25F8
PI	25A2	A7	84	2627
INKEY\$	25A4	A6	8F	2634

BIN	25A6	C4	E6	268D
SCREEN	25A8	AA	BF	2668
ATTR	25AA	AB	C7	2672
POINT	25AC	A9	CE	267D
-	25AE	00		marcator de sfirsit

25AF Scanare număr, 1 caracter

La întâlnirea caracterului +, scanarea continuă cu caracterul următor.

25B3 Scanare "**25EB Scanare ()****25F5 Scanare FN****25F8 Scanare RND****2627 Scanare PI****2634 Scanare INKEY\$****2668 Scanare SCREEN\$****2672 Scanare ATTR****267B Scanare POINT****2684 Scanare caracter alfanumeric****268D Scanare zecimal/BIN****26C9 Scanare variabile**

Rutină pentru recunoașterea variabilelor și asignare de valori.

2795 Tabela operatorilor

	cod	operator	cod	operator	cod	operator	cod	operator
2795	2B	+	CF		27A3	3C	<	CD
2797	2D	-	C3		27A5	C7	<=	C9
2799	2A	*	C4		27A7	C8	>=	CA
279B	2F	/	C5		27A9	C9	<>	CB
279D	5E	^	C6		27AB	C5	OR	C7
279F	3D	=	CE		27AD	C6	AND	C8
27A1	3E	>	CC		27AF	00	marcator de sfirsit	

27B0 Tabela de priorități a operatorilor

	operator	prioritate	operator	prioritate	
27B0	-	06	27B7	>=	05
27B1	*	08	27B8	<>	05
27B2	/	08	27B9	>	05
27B3	^	0A	27BA	<	05
27B4	OR	02	27BB	=	05
27B5	AND	03	27BC	+	06
27B6	<=	05			

27BD Scanare funcții FN**28AB Trece peste funcție**

Rutina parcurge DEF FN fără a modifica CH-ADD.

28B2 Caută variabile

Rutină utilizată pentru căutarea variabilelor sau argumentelor dintr-o instrucțiune DEF FN.

La intrare: în CH-ADD adresa primei litere a variabilei căutate din program sau zona de lucru

DEFADD <> 0 Implică căutare de funcție

La ieșire: dacă variabila a fost găsită, în F: C=0;

în F: C=0, Z=1 pentru șiruri și tablouri,
 în HL adresa ultimei/singurei litere a variabilei
 dacă variabila nu a fost găsită
 în F: C=1, Z=1 pentru tablouri
 în HL adresa primei litere a variabilei din linia BASIC
 în ambele cazuri în C bit 5, 6 indică tipul variabilei

2951 Caută argumente DEF FN

Subrutină a 28B2 dacă DEFADD < > 0

2996 Parametrii șiruri/tablouri

Rutină utilizată pentru găsirea parametrilor unei variabile șir sau adresa unui număr dintr-un tablou numeric, în HL; este o subrutină a comenzii DIM pentru verificarea sintaxei.

2A52 Decupare șir

La intrare: pe stiva-calculator și în A, B, C, D și E parametrii șirului

2AB1 Salvare pe stiva-calculator

Rutina transferă A, B, C, D și E pe stiva-calculator.

2ACC Expresie întreagă

Rutina returnează în BC valoarea următoarei expresii întregi și o compară cu valoarea-limită din HL.

în A: 00 fără erori sau FF eroare

2AEE Conținutul DE +2 în DE

Rutina plasează în DE conținutul locației a cărei adresă se află în DE.

2AF2 Obține HL = HL * DE

Rutina verifică sintaxa și calculează $HL = HL * DE$.

2AFF Comanda LET

Rutină de asignare de valori pentru LET, READ și INPUT.

2BA6 Transferă valori

Rutina transferă valori numerice de pe stiva-calculator sau șiruri din zona de lucru în zona variabilelor BASIC.

2BAF Subrutina LET (1)

Pentru șiruri simple.

2BC6 Subrutina LET (2)

Transferul parametrilor șirurilor în zona variabilelor BASIC.

2BEA Subrutina LET (3)

Modificarea numelui unui șir.

2BF1 Transferă număr în virgulă flotantă

Transferă un număr în virgulă flotantă de pe stiva-calculator în A, B, C, D și E.

2C02 Comanda DIM

Tablourile de numere se inițializează cu 0.

Tablourile alfanumerice se inițializează cu blank-uri.

2C88 Caracter alfanumeric

La ieșire: în F: C=1 dacă la intrare: în A cod caracter alfanumeric

2C8D Literal

La ieșire: în F: C=1 dacă la intrare: în A cod literă

2C9B Zecimal în virgulă flotantă

Rutina citește un număr zecimal cifră cu cifră și returnează un număr zecimal în virgulă flotantă pe stiva-calculator.

2D1B Număr

La ieşire: în F: $C=0$ dacă la intrare: în A cod cifră

2D22 Cifră pe stiva-calculator

Cifra din A este transformată în virgulă flotantă și transferată pe stiva-calculator.

2D28 A pe stiva-calculator

Numărul binar din A este transformat în număr în virgulă flotantă și transferat pe stiva-calculator.

2D2B BC pe stiva-calculator

Numărul binar din BC este transformat în număr în virgulă flotantă și transferat pe stiva-calculator.

2D3B Întreg în virgulă flotantă

Transformă un întreg dintr-o linie BASIC în număr în virgulă flotantă pe stiva-calculator.

1.9 RUTINE ARITMETICE**2D4F Format exponențial în virgulă flotantă**

Rutina transformă un număr din formatul xEy ($x \cdot 10^y$) în format în virgulă flotantă.

La intrare: x pe stiva-calculator
y în A

La ieşire: număr în virgulă flotantă pe stiva-calculator

2D7F Aduce întreg

Rutina încarcă în DE un număr întreg (între -65535 și 65535) din locația adresată de HL (5 bytes pe stiva-calculator).

în C: semnul: 00 pentru +, FF pentru -

2D8C Memorează întreg

Rutina încarcă în memorie un număr întreg (între -65535 și 65535) în locația adresată de HL (analog 2D7F).

în C: semnul: 00 pentru +, FF pentru -

2DA2 Virgulă flotantă în BC

Rutina comprimă un număr în virgulă flotantă de pe stiva-calculator în BC.

2DC1 LG (2^A)

Rutină utilizată de rutina 2DE3, calculează numărul de cifre al unui număr în virgulă flotantă

La intrare: în A exponentul numărului

La ieşire: pe stiva-calculator partea întregă a $\lg(2^A)$

2DD5 Virgulă flotantă în A

Rutina comprimă un număr în virgulă flotantă de pe stiva-calculator în A.

2DE3 Afişează un număr în virgulă flotantă

Rutina afişează numărul în virgulă flotantă de pe stiva-calculator.

2F8B $AC = 10 \cdot A + C$

Subrutină a rutinei 2DE3; calculează în HL: $AC = 10 * A + C$.

2F9B Pregătire de calcul

Rutina pregătește un număr în virgulă flotantă pentru operații aritmetice (adunare, scădere, înmulțire, împărțire), efectuând complement față de 2 pentru numere negative.

La ieșire: în A exponentul
primul byte: 00 pentru număr pozitiv, FF pentru număr negativ

2FBA Aduce 2 numere în virgulă flotantă

Rutina este folosită la adunarea, înmulțirea și împărțirea numerelor în virgulă flotantă.

La ieșire: în H', B', C', C, B numărul din vârful stivei-calculator
în L', D', E', D, E numărul al 2-lea din stiva-calculator

2FDD Shift pentru adunare

Potrivește exponentul unui număr în virgulă flotantă pentru adunare.

3004 Adună transportul C

Rutina adună fanionul de transport C (de depășire aritmetică), este utilizată de adunare și înmulțire.

300F Scădere (Calculator-offset 03)

Rutina schimbă semnul scăzătorului, continuă cu adunare (3014).

3014 Adunare (Calculator-offset 0F)

30A9 HL=HL*DE

Înmulțire pe 16 biți.

30C0 Pregătire de înmulțire în virgulă flotantă

La ieșire: în F:C=1 dacă numărul este 0
în A semnul rezultatului
înlocuiește bitul de semn cu 1

30CA Înmulțire (Calculator-offset 04)

31AF Împărțire (Calculator-offset 05)

3214 Trunchere (Calculator-offset 3A)

Rutina rotunjește un număr prin eliminarea părții fracționare

3293 Stivuește 2 întregi

Rutina expandează 2 întregi în forma virgulă flotantă și îi pune pe stivă..

3297 Stivuește un întreg (Calculator-offset 3D)

Rutina expandează un întreg în forma virgulă flotantă și îl pune pe stivă..

1.10 CALCULATORUL ÎN VIRGULĂ FLOTANTĂ

Calcularele în virgulă flotantă se fac prin apelarea rutinei 0028 în forma:

RST 28

DEFB offset

...

DEFB end-calc

unde DEFB semnifică operațiile care se fac cu numerele în virgulă flotantă.
 Pentru operații se folosește o stivă-calculator diferită de stiva microprocesorului.
 Notația stivei-calculator: (înainte— —după), unde vârful stivei este în dreapta (atât înainte de efectuarea operației cât și după).

32C5 Tabela de constante

	Expandat	Const.	
32C5	DEFB 00,B0,00	00,00,00,00,00	0
32CB	DEFB 40,B0,00,01	00,00,01,00,00	1
32CC	DEFB 30,00	80,00,00,00,00	1/2
32CE	DEFB F1,49,0F,DA,A2	81,49,0F,DA,A2	PI/2
32D3	DEFB 40,B0,00,0A	00,00,0A,00,00	10

32D7 Tabela offset operații Calculator

	off.	adr.	nume	acțiune
32D7	00	8F, 36	jump-true	((1/0) — —) se utilizează: DEFB 00; DEFB n; începînd numărătoarea după DEFB n, unde se face saltul
32D9	01	3C, 34	exchange	(x, y — —y, x)
32DB	02	A1, 33	delete	(x — —)
32DD	03	0F, 30	subtract	(x, y — —(x—y))
32DF	04	CA, 30	multiply	(x, y — —x*y)
32E1	05	AF, 31	division	(x, y — —x/y)
32E3	06	51, 38	to-power	(x, y — —x^y)
32E5	07	1B, 35	or	(x, y — —z) z = x dacă y = 0; z = 1 altfel (OR)
32E7	08	24, 35	no-&-no	(x, y — —z) z = x dacă y < > x; z = 0 altfel (AND)
32E9	09	3B, 35	no-l-eql	(x, y — —z) z = 1 dacă x < = y; z = 0 altfel; comparare < = numere
32EB	0A	3B, 35	no-gr-eql	(x, y — —z) z = 1 dacă x > = y; z = 0 altfel; comparare > = numere
32ED	0B	3B, 35	nos-neql	(x, y — —z) z = 1 dacă x < > y; z = 0 altfel; comparare < > numere
32EF	0C	3B, 35	no-grtr	(x, y — —z) z = 1 dacă x > y; z = 0 altfel; comparare > numere
32F1	0D	3B, 35	no-less	(x, y — —z) z = 1 dacă x < y; z = 0 altfel; comparare < numere
32F3	0E	3B, 35	nos-eql	(x, y — —z) z = 1 dacă x = y; z = 0 altfel; comparare = numere
32F5	0F	14, 30	addition	(x, y — —x+y)

	off.	adr.	nume	acțiune
3327	28	4A, 38	sqr	(x— —SQR x)
3329	29	92, 34	sgn	(x— —SGN x)
332B	2A	6A, 34	abs	(x— —ABS x)
332D	2B	AC, 34	peek	(x— —PEEK x)
332F	2C	A5, 34	in	(x— —IN x)
3331	2D	B3, 34	usr-no	(x— —) salt la rutina care începe la adresa INT x
3333	2E	1F, 36	str\$	(— —z\$) evaluează expresia STR\$ x
3335	2F	C9, 35	chr\$	(— —z\$) evaluează expresia CHR\$ x
3337	30	01, 35	not	(x— —z) z=1 dacă x=0; altfel z=0
3339	31	C0, 33	duplicate	(x— —x x)
333B	32	A0, 36	n-mod-m	(n, m— —(n—INT(n/m)), (INT(n/m)))
333D	33	86, 36	jump	(— —) se utilizează DEFB 33, DEFB n; unde n este byte-ul numărul n, începînd număratoarea după DEFB n unde se face saltul relativ, n: număr cu semn în complement față de 2
333F	34	C6, 33	stk-data	(— —x) următorii 5 bytes după DEFB 34 reprezintă un număr x în virgulă flotantă
3341	35	7A, 36	dec-jr-nz	(— —) decrementează BREG, salt relativ dacă BREG < > 0; se utilizează în forma: DEFB 35, DEFB n; unde n este byte-ul numărul n, începînd număratoarea după DEFB n unde se face saltul relativ; n număr cu semn în complement față de 2
3343	36	06, 35	less-0	(x— —y) y=1 dacă x<0; y=0 altfel
3345	37	F9, 34	greater-0	(x— —y) y=1 dacă x>0; y=0 altfel
3347	38	9B, 36	end-calc	(— —) reîntoarcere în rutina apelantă, orice apel RST 20 se termină cu DEFB 38
3349	39	83, 37	get-argt	(x— —v) v: valoare utilizată de funcțiile SIN și COS; not.: $y = x / (2 * \pi) - \text{INT}(x / (2 * \pi) + 0.5)$; atunci: $4 * y$ în $[-1, 1] = > v = 4 * y$; $4 * y$ în $(1, 2] = > v = 2 - 4 * y$; $4 * y$ în $[-2, -1) = > v = -2 - 4 * y$
334B	3A	14, 32	truncate	(x— —y) unde y este obținut din x, prin eliminarea părții fracționare
334D	3B	A2, 33	fp-calc-2	rutină utilizată doar de rutina 26C9 (la adresa 2757)

	off.	adr.	nume	acțiune
334F	3C	4F, 2D	e-to-fp	$(x \rightarrow y)$ transformă un număr din forma exponențială (xEm , m în A) în forma virgulă flotantă y
3351	3D	97, 32	re-stack	$(x \rightarrow x')$ x' este forma completă în virgulă flotantă a numărului x
3353	86 88 8C	49, 34	series	generează serii polinomiale Cebîșev pentru calculul funcțiilor; offset: 86 pentru SIN; 88 pentru EXP; 8C pentru LN și ATN; după fiecare offset urmează 6, 8 respectiv 12 numere în virgulă flotantă
3355	A0 A1 A2 A3 A4	1B, 34	stk-zero stk-one stk-half stk-pi/2 stk-ten	$(\rightarrow 0)$ creează pe stivă constanta respectivă $(\rightarrow 1)$ $(\rightarrow 1/2)$ $(\rightarrow \pi/2)$ $(\rightarrow 10)$
3357	C0 C1 C2 C3 C4 C5	2D, 34	st-mem-0 st-mem-1 st-mem-2 st-mem-3 st-mem-4 st-mem-5	$(x \rightarrow \rightarrow)$ retrage un număr de pe stivă și îl introduce în MEMBOT
3359	E0 E1 E2 E3 E4 E5	0F, 34	get-mem-0 get-mem-1 get-mem-2 get-mem-3 get-mem-4 get-mem-5	introduce un număr în stivă din MEMBOT

Observație: Șirurile sînt date în formă parametrică:

byte 1 nesemnificativ
bytes 2,3 adresă
bytes 4,5 lungime

335B Calculatorul

Rutină apelată de RST 28 (calculatorul în virgulă flotantă)

33A1 Delete (Calculator-offset 02)

33A2 Operație-unică (Calculator-offset 3B)

Rutină utilizată de calculator pentru operații cu un singur parametru.

33A9 Test 5 bytes

Rutina testează dacă există spațiu suficient pe stivă (în memorie) pentru un număr între stivă și memorie.

33B4 Număr pe stivă

Rutină utilizată de rutinele 03F8 și 24FB.

33C0 Deplasează un număr (Calculator-offset 31)

Rutina deplasează un întreg între stivă și memorie.

33C6 Număr pe stivă (Calculator-offset 34)

33F7 Selectează constanta

Rutină utilizată de rutina 341B

3406 Locația de memorie

Rutină de calcul a adreselor celor 6 numere în virgulă flotantă din MEMBOT.

340F Aduce din memorie (Calculator-offset E0...E5)**341B Constantă pe stivă (Calculator-offset A0...A4)****342D Stochează în memorie (Calculator-offset C0...C5)****343C Permută (Calculator-offset C0...C5)****3449 Generator de serii (Calculator-offset 86,88,8C)**

Observație: instrucțiunile CALL 335E și CALL 3362 se pot citi ca RST 28.

346A Funcția ABS (Calculator-offset 2A)**346E Negare (Calculator-offset 1B)****3492 Funcția SGN (Calculator-offset 29)****34A5 Funcția IN (Calculator-offset 2C)****34AC Funcția PEEK (Calculator-offset 2B)****34B3 Funcția USR număr (Calculator-offset 2D)****34BC Funcția USR șir (Calculator-offset 19)****34E9 Test 0**

La ieșire: în F: C=0 dacă numărul de pe stiva-calculator este 0

34F9 Mai mare ca 0 (Calculator-offset 37)**3501 Funcția NOT (Calculator-offset 30)****3506 Mai mic ca 0 (Calculator-offset 36)****350B 0 sau 1**

Rutina setează numărul de pe stivă la valoarea:

0 dacă în F: C=0

1 dacă în F: C=1

351B Funcția OR (Calculator-offset 07)**3524 Funcția AND (Calculator-offset 08 și 10)**

Rutina 3524 pentru no-&-no.

Rutina 352D pentru str-&-no.

353B Comparare (Calculator-offset 09...0E și 11...16)**359C Concatenarea șirurilor (Calculator-offset 17)****35BF Adrese stivă**

La ieșire: în HL: STKEND-5 = adresa primului byte al numărului de pe stiva-calculator

în DE: STKEND = adresa primului byte liber pe stiva-calculator

35C9 Funcția CHR\$ (Calculator-offset 2F)**35DE Funcția VAL și VAL\$ (Calculator-offset 18 și 1D)****361F Funcția STR\$ (Calculator-offset 2E)****3645 Citește stream (Calculator-offset 1A)****3669 Funcția CODE șir (Calculator-offset 1C)****3674 Funcția LEN (Calculator-offset 1E)****367A Buclă cu contor (Calculator-offset 35)****3686 Salt (Calculator-offset 33)****368F Salt dacă adevărat (Calculator-offset 00)****369B Sfârșit calcul (Calculator-offset 38)**

- 36A0 Modul (Calculator-offset 32)
 36AF Funcția INT (Calculator-offset 27)
 36C4 Funcția EXP (Calculator-offset 26)
 3713 Funcția LN (Calculator-offset 25)
 3783 Get-argt (Calculator-offset 39)
 37AA Funcția COS (Calculator-offset 20)
 37B5 Funcția SIN (Calculator-offset 1F)
 37DA Funcția TAN (Calculator-offset 21)
 37E2 Funcția ARCTAN (Calculator-offset 24)
 3833 Funcția ARCSIN (Calculator-offset 22)
 3843 Funcția ARCCOS (Calculator-offset 23)
 384A Funcția SQR (Calculator-offset 28)
 3851 Funcția putere (Calculator-offset 06)
 386E Neutilizat
 3D00 Matricea caracterelor

3D00-3DFF Fiecărui caracter afișabil de la 20 (blank) la 7F (copyright) i se atribuie 8 bytes care definesc matricea 8*8 pixeli cu ajutorul căreia se afișează..

1.11 VARIABILELE DE SISTEM

Notație:	m	variabila poate fi modificată
	N	variabila nu trebuie modificată
	-	variabilă modificată permanent de sistem
	[]	în paranteze sînt adresele rutinelor care le utilizează

Adresă	Byte	Mod	Nume	Observații
5C00	8	N	KSTATE	definește starea tastaturii [02BF, 11B7]
5C08	1	-	LAST-K	conține codul ultimei taste apăsată (02BF)
5C09	1	m	REPDEL	durata maximă de apăsare a unei taste fără repetare este 0.02s*REPDEL [02BF, 11B7]
5C0A	1	m	REPPER	durata între repetiții cînd o tastă este menținută apăsată (0.1s) [0310]
5C0B	2	-	DEFADD	adresa argumentelor FN [27BD]
5C0D	1	-	K-DATA	byte de control, culori tastate [10A8]
5C0E	2	-	TVDATA	byte de control, culoare utilizată de AT și TAB [0A60]
5C10	38	N	STRMS	conține adresele canalelor
5C36	2	m	CHARS	conține adresa tabelului matrice al caracterelor (3D00-0100 = 3C00)
5C38	1	m	RASP	durata semnalului sonor
5C39	1	m	PIP	lungimea semnalului sonor la apăsarea unei taste

5C3A	1	m	ERR-NR	codul erorii întâlnite este ERR-NR + 1, în absența erorii ERR-NR=FF [0053, 2089]
5C3B	1	N	FLAGS	fanioane [02BF, 0A23, 0A4F, 0A6D, 0ADC, 0C55, 11B7, 1795, 1F3A, 1F60, 2089, 2AEE, 35DE]
5C3C	1	N	TV-FLAGS	fanioane [0970, 0ADC, 0C55, 0D4D, 0E44, 11B7, 1795, 2089]
5C3D	1	N	ERR-SP	adresa stivei în caz de eroare [0F2C, 11B7, 1EAC, 1EED, 1F23, 2089]
5C3F	2	-	LIST-SP	conține adresa de retur după LIST [0C55, 1795]
5C41	1	-	MODE	modul cursor: K,L,C,E sau G [02BF, 0F2C, 0F81, 18E1]
5C42	2	m	NEWPPC	numărul liniei BASIC unde trebuie efectuat un salt [0808, 1D03, 1E67]
5C44	2	m	NSPPC	numărul instrucțiunii dintr-o linie BASIC unde trebuie efectuat un salt [0808, 1D03, 1E67]
5C45	2	m	PPC	numărul liniei BASIC în curs de execuție [1D03, 1EED]
5C47	1	m	SUBPPC	numărul instrucțiunii în curs de execuție din linia BASIC [1D03, 1EED]
5C48	1	m	BORDCR	culoarea border-ului și a liniilor de jos ale ecranului [053F, 0D4D, 0E44]
5C49	2	m	E-PPC	numărul liniei în care se găsește cursorul [0FA9, 1795]
5C4B	2	N	VARS	adresa zonei de memorie în care se găsesc variabilele BASIC [0605, 0808, 08B6, 11B7, 1EAC]
5C4D	2	-	DEST	adresa variabilei în curs de asignare [2AFF]
5C4F	2	N	CHANS	adresa datelor asupra diferitelor canale
5C51	2	N	CURCHL	adresa de informații in/out [0AD6, 0D6B, 0DAF, 0FA9, 361F]
5C53	2	N	PROG	adresa de început a programului BASIC [0605, 08B6, 092C, 11B7, 196E]
5C55	2	N	NXTLIN	adresa următoarei linii BASIC de executat [1BB2, 1D03]
5C57	2	N	DATADD	adresa sfârșitului ultimei date (DATA) [11B7, 1DEC, 1E42]
5C59	2	N	E-LINE	conține adresa zonei editor [0605, 0808, 11B7, 19FB, 2AEE, 2C02]
5C5B	2	m	K-CUR	adresa cursorului pe ecran [0F2C, 0F81, 18E1, 20B9, 361F]
5C5D	2	N	CH-ADD	adresa următorului caracter de interpretat [0018, 0074, 19FB, 1D03, 2089, 35DE]
5C5F	2	m	X-PTR	adresa caracterului care precede cursorul de eroare în editare sau INPUT [0808, 092C, 1DEC, 2089]
5C61	2	N	WORKSP	adresa zonei de lucru [0030, 11B7, 169E, 2089]
5C63	2	N	STKBOT	adresa de început a stivei-calculator [11B7, 169E, 2089]

5C65	2	N	STKEND	adresa de sfârșit a stivei-calculator [11B7, 19FB, 1EAC, 1F05]
5C67	1	-	BREG	contor utilizat de o instrucțiune pseudo-DJNZ [0028]
5C68	2	-	MEM	adresa stivei-calculator [1D03]
5C6A	1	m	FLAGS2	fanioane pentru decodarea tastaturii [0333, 0DAF, 1795, 18E1, 2089]
5C6B	1	N	DF-SZ	numărul liniilor din partea de jos a ecranului [0C55, 0D6B, 11B7, 1795, 2089]
5C6C	2	m	S-TOP	numărul liniei superioare într-un LIST, LLIST
5C6E	2	m	OLDPPC	numărul liniei BASIC de unde se continuă programul după CONTINUE
5C70	1	m	OSPPC	numărul instrucțiunii într-o linie de unde se continuă programul după CONTINUE
5C71	1	-	FLAGX	fanioane [0F2C, 0FA9, 2089, 2AEE]
5C72	2	-	STRLEN	numărul caracterelor de interpretat [1D03, 2AEE]
5C74	2	-	T-ADDR	adresa următoarei rubrici în tabela de sintaxă, utilizată de rutinele SAVE și LOAD
5C76	2	m	SEED	baza seriei numerelor pseudo-aleatoare [1E4F]
5C78	3	m	FRAMES	control de ceas real, FRAMES*0.02s= timpul scurs de la pornirea calculatorului (sau reset hard) [0038, 1E4F]
5C7B	2	m	UDG	adresa primului caracter grafic
5C7D	1	m	COORDS	coordonata x a ultimului pixel afișat [22DC, 2320, 2382]
5C7E	1	m	COORDS	coordonata y a ultimului pixel afișat [22DC, 2320, 2382]
5C7F	1	m	P-POSN	poziția capului de imprimantă
5C80	1	m	PR-CC	adresa următorului caracter de imprimat
5C81	1	m	-	neutilizat
5C82	2	m	ECHO-E	numărul ultimei coloane utilizată la un INPUT [0ADC]
5C84	2	m	DF-CC	adresa următoarei poziții de scriere PRINT [0ADC]
5C86	2	m	DFCCL	adresa următoarei poziții de scriere în partea de jos a ecranului
5C88	1	N	S-POSN	numărul coloanei următoarei instrucțiuni PRINT
5C89	1	N	S-POSN	numărului liniei următoarei instrucțiuni PRINT
5C8A	2	N	SPOSNL	numărul coloanei și liniei de afișare pentru partea de jos a ecranului
5C8C	1	m	SCR-CT	contor al numărului de linii înainte de scroll? [0C55, 0DAF, 2089]
5C8D	1	m	ATTR-P	numărul culorii permanente [0DAD, 0D6B, 0E44, 11B7]
5C8E	1	m	MASK-P	culori transparente permanente
5C8F	1	-	ATTR-T	numărul culorii temporare [0BDB, 0C55, 0D4D, 11B7, 18C1]

5C90	1	-	MASK-T	culori transparente temporare
5C91	1	m	P-FLAG	fanoane [0A3D, 0BD3, 0C55, 18C1, 22DC]
5C92	30	N	MEMBOT	6 locații-registre pentru numere în virgulă flotantă (5 bytes fiecare) [0028]
5CB0	2	m	NMIADD	inutilizată [0066]
5CB2	2	m	RAMTOP	adresa ultimului byte utilizat de BASIC [11B7, 1EAC]
5CB4	2	m	P-RAMT	adresa ultimului byte fizic

1.12 HARTA MEMORIEI TOTALE

0000	ROM
4000	Harta de pixeli
5800	Atribute
5B00	Tampon de imprimantă
5C00	Variabile de sistem
5CB6	Harta microdrive
CHANS	Informații de canale
PROG	Program BASIC
VARs	Variabile
E-LINE	Comanda sau linia BASIC în curs de editare
WORKSP	Input date
	Spațiu de lucru temporar
STKBOT	Stiva-calculator
STKEND	Liber
SP-register	Stiva microprocesorului
	Liber
	Stiva GO SUB
RAMTOP	UDG
FFFF	-

1.13 STRUCTURI BASIC

Structura liniei BASIC

byte-inf	byte-sup	lungime-text + 1	text	cod Enter
număr	de	linie		

Observație: Numărul de linie pe 2 bytes este singurul pentru care nu este

respectată convenția Intel (byte-ul mai puțin semnificativ la adresa inferioară și byte-ul mai semnificativ la adresa superioară).

Variabilă numerică, nume = 1 literă

011	litera-1—60	exponent	bit-semn, mantisă = 4 bytes
-----	-------------	----------	-----------------------------

Variabilă numerică, nume = k litere ($k > 1$)

101	litera-1—60	litera-2	...	1 litera-k	valoare = 5 bytes
-----	-------------	----------	-----	------------	-------------------

Variabilă tablou numerică, k dimensiuni

1 0 0	litera-1-60	lungime-totală-elemente + dimensiuni + 1 = 2bytes
-------	-------------	---

numărul dimensiunilor	dimensiune-1 = 2 bytes	...
-----------------------	------------------------	-----

dimensiune-k = 2 bytes	valori = 5 bytes fiecare
------------------------	--------------------------

Ordinea este $x(1,1...)$, $x(2,1...)$... (invers lexicografică).

Variabilă șir, 1 dimensiune

010	litera-60	lungime = 2 bytes	text
-----	-----------	-------------------	------

Variabilă tablou șir, k dimensiuni

1 0 0	litera-1-60	lungime-totală-elemente + dimensiuni + 1 = 2 bytes
-------	-------------	--

numărul dimensiunilor	dimensiune-1 = 2 bytes	...
-----------------------	------------------------	-----

dimensiune-k = 2 bytes	elemente = 1 byte fiecare
------------------------	---------------------------

Variabilă de control pentru FOR-NEXT

111	litera—60	valoare = 5 bytes	limita = 5 bytes	treapta = 5 bytes
-----	-----------	-------------------	------------------	-------------------

byte-inf	byte-sup	număr-instrucțiune
număr linie	FOR	în linie FOR

1.14 SETUL DE CARACTERE SPECTRUM

Observație: corespunde cu codurile ASCII între 32 și 127

	0	1	2	3	4	5	6	7	8	9
0							PRINT comma	EDIT control	cursor stringa	cursor dreapta
10	cursor jos	cursor sus	Delete	Enter	number		INK control	PAPER control	FLASH control	BRIGHT control
60		=		?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[\]	^	-	£	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	{		}	~	copy- right	udg-1	udg-2
130	udg-3	udg-4	udg-5	udg-6	udg-7	udg-8	udg-9	udg-10	udg-11	udg-12
140	udg-13	udg-14	udg-15	udg-16	udg-A	udg-B	udg-C	udg-D	udg-E	udg-F
150	udg-G	udg-H	udg-I	udg-J	udg-K	udg-L	udg-M	udg-N	udg-O	udg-P
160	udg-Q	udg-R	udg-S	udg-T	udg-U	RND	INKEY\$	PI	FN	POINT
170	SCRN.	ATTR	AT	TAB	VAL\$	CODE	VAL	LEN	SIN	COS
180	TAN	ASN	ACS	ATN	LN	EXP	INT	SQR	SGN	ABS
190	PEEK	IN	USR	STR\$	CHR\$	NOT	BIN	OR	AND	< =
200	> =	< >	LINE	THEN	TO	STEP	DEF FN	CAT	FOR. MAT	MOVE
210	ERASE	OPEN #	CLOSE #	MERGE	VERIFY	BEEP	CIRCLE	INK	PAPER	FLASH
220	BRIGHT	INVER- SE	OVER	OUT	LPRINT	LLIST	STOP	READ	DATA	RESTO- RE
230	NEW	BOR- DER	CONTI- NUE	DIM	REM	FOR	GO TO	GO SUB	INPUT	LOAD
240	LIST	LET	PAUSE	NEXT	POKE	PRINT	PLOT	RUN	SAVE	RAND.
250	IF	CLS	DRAW	CLEAR	RE- TURN	COPY				

2 ASAMBLORUL/DEZASAMBLORUL

2.1 MICROPROCESORUL Z80

Regiștrii microprocesorului Z80 sînt:

A acumulator (registru principal) (8 biți)

F flags (registru fanioane acumulator) (8 biți) conține:

bit 7	S semn	bit 3	* nedefinit
bit 6	Z zero	bit 2	P/V paritate/depășire
bit 5	* nedefinit	bit 1	N adunare/scădere
bit 4	H half carry	bit 0	C carry

B,C,D,E,H,L regiștri de uz general (8 biți)

pot fi utilizați în pereche BC, DE, HL pentru a forma o adresă set alternat de regiștri; operează la fel ca analogii lor

A, F, B, C,

D, E, H, L

IX, IY

SP

PC

R, I

Notații:

regiștrii index (16 biți)

stack-pointer: adresa vârfului stivei care crește în jos (16 biți)

program-counter: adresa următoarei instrucțiuni (16 biți)

regiștri cu destinație specială (8 biți)

r,s regiștri (8 biți)

rp pereche de regiștri (16 biți)

d,n,e dată numerică (0-FF) (8 biți)

nn dată numerică (0-FFFF) (16 biți)

b bit (7-0)

Fc bitul C din F

Fz bitul Z din F

c condiție

T perioadă de tact, ceas

fanion în F: ! afectat conform rezultatului

. neafectat

* nedefinit

<-> permutarea conținutului a 2 regiștri

<- asignare
 (rp) conținutul byte-ului de memorie a cărei adresă se află în rp
 (nn) conținutul byte-ului de memorie a cărei adresă este nn
 A0~A15 magistrala de adrese
 D0~D7 magistrala de date

Convenția Intel: byte-ul mai puțin semnificativ la adresa inferioară
 byte-ul mai semnificativ la adresa superioară

2.2 ÎNCĂRCARE PE 8 BIȚI

Format	Operație	Flags	Cod	T	Observații
LD r,s	r <- s	..*.*...	01-r-s-	4	r,s Reg
LD r,n	r <- n	..*.*...	00-r-100 -n-	7	000 B 001 C
LD r, (HL)	r <- (HL)	..*.*...	01-r-110	7	010 D
LD r, (IX+d)	r <- (IX+d)	..*.*...	11011101 01-r-110 -d-	19	011 E 100 H 101 L
LD r, (IY+d)	r <- (IY+d)	..*.*...	11111101 01-r-110 -d-	19	111 A
LD (HL), n	(HL) <- n	..*.*...	00110110 -n-	10	
LD (IX+d), n	(IX+d) <- n	..*.*...	11011101 00110110 -d- -n-	19	
LD (IY+d), n	(IY+d) <- n	..*.*...	11111101 00110110 -d- -n-	19	
LD A, (BC)	A <- (BC)	..*.*...	00001010	7	
LD A, (DE)	A <- (DE)	..*.*...	00011010	7	
LD A, (nn)	A <- (nn)	..*.*...	00111010 -n- -n-	13	
LD (BC), A	(BC) <- A	..*.*...	00000010	7	
LD (DE), A	(DE) <- A	..*.*...	00010010	7	
LD (nn), A	(nn) <- A	..*.*...	00110010 -n- -n-	13	

Format	Operație	Flags	Cod	T	Observații
LDA, I	A ← I	!!*0b0.	11101101 01010111	9	b = IFF, bit de întrerupere
LDA, R	A ← R	!!*0*b0.	11101101 01011111	9	
LDI, A	I ← A	..**...	11101101 01000111	9	
LD R, A	R ← A	..**...	11101101 01001111	9	
LD (HL), r	(HL) ← r	..**...	01110-r-	7	
LD (IX+d), r	(IX+d) ← r	..**...	11011101 01110-r- -d-	19	
LD (IY+d), r	(IY+d) ← r	..**...	11111101 01110-r- -d-	19	

2.3 ÎNCĂRCARE PE 16 BIȚI

Format	Operație	Flags	Cod	T	Observații
LD rp, nn	rp ← nn	..**...	00rp001 -n- -n-	10	rp Reg.p. 00 BC 01 DE
LD IX, nn	IX ← nn	..**...	11011101 00100001 -n- -n-	14	10 HL 11 SP
LD IY, nn	IY ← nn	..**...	11111101 00100001 -n- -n-	14	
LD HL, (nn)	H ← (nn + 1) L ← (nn)	..**...	00101010 -n- -n-	16	
LD rp, (nn)	rph ← (nn + 1) rpl ← (nn)	..**...	11101101 01rp1011 -n- -n-	20	

Format	Operație	Flags	Cod	T	Observații
LD IX,(nn)	IX _h <- (nn + 1) IX _l <- (nn)	..*.*...	11011101 00101010 -n- -n-	20	
LD IY, (nn)	IY _h <- (nn + 1) IY _l <- (nn)	..*.*...	11111101 00101010 -n- -n-	20	
LD (nn), HL	(nn + 1) <- H (nn) <- L	..*.*...	00100010 -n- -n-	16	
LD (nn), rp	(nn + 1) <- rp _h (nn) <- rp _l	..*.*...	11101101 01rp0011 -n- -n-	20	
LD (nn), IX	(nn + 1) <- IX _h (nn) <- IX _l	..*.*...	11011101 00100010 -n- -n-	20	
LD (nn),IY	(nn + 1) <- IY _h (nn) <- IY _l	..*.*...	11111101 00100010 -n- -n-	20	
LD SP, HL	SP <- HL	..*.*...	11111001	6	
LD SP,IX	SP <- IX	..*.*...	11011101 11111001	10	
LD SP, IY	SP <- IY	..*.*...	11111101 1111101	10	
PUSH rp	(SP-2) <- rp _l (SP-1) <- rp _h SP <- SP-2	..*.*...	11rp0101	11	
PUSH IX	(SP-2) <- IX _h (SP-1) <- IX _l SP <- SP-2	..*.*...	11011101 11100101	15	
PUSH IY	(SP-2) <- IY _h (SP-1) <- IY _l SP <- SP-2	..*.*...	11111101 11100101	15	
POP rp	rp _h <- (SP + 1) rp _l <- (SP) SP <- SP + 2	..*.*...	11rp0001	10	
POP IX	IX _h <- (SP + 1) IX _l <- (SP) SP <- SP + 2	..*.*...	11011101 11100001	14	
POP IY	IY _h <- (SP + 1) IY _l <- (SP) SP <- SP + 2	..*.*...	11111101 11100001	14	

2.4 SCHIMB, TRANSFER DE BLOCURI ȘI CĂUTARE

Format	Operație	Flags	Cod	T	Observații
EX DE, HL	DE <-> HL	..*.*...	11101011	4	comutare pe setul
EX AF, AF'	AF <-> AF'	!!*!*!!!	00001000	4	alternat de
EXX	BC <-> BC' DE <-> DE' HL <-> HL'	..*.*...	11011001	4	registri
EX (SP), HL	H <-> (SP + 1) L <-> (SP)	..*.*...	11100011	19	
EX (SP), IX	IXh <-> (SP + 1) IXl <-> (SP)	..*.*...	11011101 11100011	23	
EX (SP), IY	IYh <-> (SP + 1) IYl <-> (SP)	..*.*...	11111101 11100011	23	
LDI	(DE) <- (HL) DE <- DE + 1 HL <- HL + 1 BC <- BC - 1	..*0*!0.	11101101 10100000	16	P/V = 0 < = > BC - 1 = 0; altfel P/V = 1
LDIR	LDI pînă cînd BC = 0	..*0*00.	11101101 10110000	21 16	dacă BC < > 0 dacă BC = 0
LDD	(DE) <- (HL) DE <- DE - 1 HL <- HL - 1 BC <- BC - 1	..*0*!0.	11101101 10101000	16	P/V = 0 < = > BC - 1 = 0; altfel P/V = 1
LDDR	LDI pînă cînd BC = 0	..*0*00.	11101101 10111000	21 16	dacă BC < > 0 dacă BC = 0
CPI	A comparat cu (HL) HL <- HL + 1 BC <- BC - 1	!!*!*!1.	11101101 10100001	16	Z = 1 dacă A = (HL); P/V = 0 dacă BC - 1 = 0
CPIR	CPI pînă cînd A = (HL) sau BC = 0	!!*!*!1.	11101101 10110001	21 16	dacă A < > (HL) și BC < > 0 dacă A = (HL) sau BC = 0 Z, P/V conform CPI
CPD	A comparat cu (HL) HL <- HL - 1 BC <- BC - 1	!!*!*!1.	11101101 10101001	16	Z = 1 dacă A = (HL); P/V = 0 dacă BC - 1 = 0

Format	Operație	Flags	Cod	T	Observații
CPDR	CPD pînă cînd A = (HL) sau BC = 0	!!*!*11.	11101101 10111001	21 - 16	dacă A < > (HL) și BC < > 0 dacă A = (HL) sau BC = 0 Z, P/V conform CPI

2.5 ARITMETICĂ ȘI LOGICĂ PE 8 BIȚI

Format	Operație	Flags	Cod	T	Observații
ADD A, r	$A \leftarrow A+r$!!*!*V0!	10000-r-	4	r Reg.
ADD A, n	$A \leftarrow A+n$!!*!*V0!	11000110 -n-	7	000 B 001 C
ADD A, (HL)	$A \leftarrow A+(HL)$!!*!*V0!	10000110	7	010 D
ADD A, (IX+d)	$A \leftarrow A+(IX+d)$!!*!*V0!	11011101 10000110 -d-	19	011 E 100 H 101 L
ADD A, (IY+d)	$A \leftarrow A+(IY+d)$!!*!*V0!	11111101 10000110 -d-	19	111 A
ACD A, r	$A \leftarrow A+r+Fc$!!*!*V0!	10001-r-	4	
ACD A, n	$A \leftarrow A+n+Fc$!!*!*V0!	11001110 -n-	7	
ACD A, (HL)	$A \leftarrow A+(HL)+Fc$!!*!*V0!	10001110	7	
ACD A, (IX+d)	$A \leftarrow A+(IX+d)+Fc$!!*!*V0!	11011101 10001110 -d-	19	
ACD A, (IY+d)	$A \leftarrow A+(IY+d)+Fc$!!*!*V0!	11111101 10001110 -d-	19	
SUB r	$A \leftarrow A-r$!!*!*V1!	10010-r-	4	
SUB n	$A \leftarrow A-n$!!*!*V1!	11010110 -n-	7	
SUB (HL)	$A \leftarrow A-(HL)$!!*!*V1!	10010110	7	
SUB (IX+d)	$A \leftarrow A-(IX+d)$!!*!*V1!	11011101 10010110 -d-	19	
SUB (IY+d)	$A \leftarrow A-(IY+d)$!!*!*V1!	11111101 10010110 -d-	19	

Format	Operație	Flags	Cod	T	Observații
SBC A, r	A <- A—r—F _c	!!*!*V1!	10011-r- 11011110 -n-	4	
SBC A, n	A <- A—n—F _c	!!*!*V1!	11011110 -n-	7	
SBC A, (HL)	A <- A—(HL)—F _c	!!*!*V1!	10011110	7	
SBC A, (IX+d)	A <- A—(IX+d)—F _c	!!*!*V1!	11011101 10011110 -d-	19	
SBC A, (IY+d)	A <- A—(IY+d)—F _c	!!*!*V1!	11111101 10011110 -d-	19	
AND r	A <- A și r	!!*!*P00	10100-r-	4	
AND n	A <- A și n	!!*!*P00	11100110 -n-	7	
AND (HL)	A <- A și (HL)	!!*!*P00	10100110	7	
AND (IX+d)	A <- A și (IX+d)	!!*!*P00	11011101 10100110 -d-	19	
AND (IY+d)	A <- A și (IY+d)	!!*!*P00	11111101 10100110 -d-	19	
OR r	A <- A sau r	!!*!*P00	10110-r-	4	
OR n	A <- A sau n	!!*!*P00	11110110 -n-	7	
OR (HL)	A <- A sau (HL)	!!*!*P00	10110110	7	
OR (IX+d)	A <- A sau (IX+d)	!!*!*P00	11011101 10110110 -d-	19	
OR (IY+d)	A <- A sau (IY+d)	!!*!*P00	11111101 10110110 -d-	19	
XOR r	A <- A sau-exclusiv r	!!*!*P00	10101-r-	4	
XOR n	A <- A sau-exclusiv n	!!*!*P00	11101110 -n-	7	
XOR (HL)	A <- A sau-exclusiv (HL)	!!*!*P00	10101110	7	
XOR (IX+d)	A <- A sau-exclusiv (IX+d)	!!*!*P00	11011101 10101110 -d-	19	
XOR (IY+d)	A <- A sau-exclusiv (IY+d)	!!*!*P00	11111101 10101110 -d-	19	
CP r	A comparat cu r	!!*!*V1!	10111-r-	4	
CP n	A comparat cu n	!!*!*V1!	11111110 -n-	7	
CP (HL)	A comparat cu (HL)	!!*!*V1!	10111110	7	

Format	Operație	Flags	Cod	T	Observații
CP (IX+d)	A comparat cu (IX+d)	!!*!*V!	11011101 10111110 -d-	19	
CP (IY+d)	A comparat cu (IY+d)	!!*!*V!	11111101 10111110 -d-	19	
INC r	$r <- r+1$!!*!*V0.	00-r-100	4	
INC (HL)	$(HL) <- (HL)+1$!!*!*V0.	00110100	11	
INC (IX+d)	$(IX+d) <- (IX+d)+1$!!*!*V0.	11011101 00110100 -d-	23	
INC (IY+d)	$(IY+d) <- (IY+d)+1$!!*!*V0.	11111101 00110100 -d-	23	
DEC r	$r <- r-1$!!*!*V0.	00-r-101	4	
DEC (HL)	$(HL) <- (HL)-1$!!*!*V0.	00110101	11	
DEC (IX+d)	$(IX+d) <- (IX+d)-1$!!*!*V0.	11011101 00110101 -d-	23	
DEC (IY+d)	$(IY+d) <- (IY+d)-1$!!*!*V0.	11111101 00110101 -d-	23	

2.6 CONTROL

Format	Operație	Flags	Cod	T	Observații
DAA	ajustare zecimală	!!*!*P.I	00100111	4	bit 3-0: dacă 9 sau H=1 adaugă 6 bit 4-7: dacă 9 sau C=1 adaugă 6+H
CPL	$A <- \text{neg } A$..*1*.1.	00101111	4	complement față de 1
NEG	$A <- (\text{neg } A)+1$!!*!*V!	11101101	8	complement față de 2
CCF	$F_c <- \text{neg } F_c$..***.0	00111111	4	
SCF	$F_c <- 1$..*.*...	00110111	4	
NOP	nici o operație	..*.*...	00000000	4	

Format	Operație	Flags	Cod	T	Observații
HALT	microprocesor stop	..*.*..	01110110	4	așteaptă întrerupere sau restart hard
DI	IFF < - 0	..*.*..	11110011	4	dezactivează întreruperile mascabile
EI	IFF < - 1	..*.*..	11111011	4	activează întreruperile mascabile
IM0	mod 0 întreruperi	..*.*..	11101101 01000110	8	
IM1	mod 1 întreruperi	..*.*..	11101101 01010110	8	
IM2	mod 2 întreruperi	..*.*..	11101101 01011110	8	

2.7 ARITMETICĂ PE 16 BIȚI

Format	Operație	Flags	Cod	T	Observații
ADD HL, ss	HL <- HL + ss	..***.0!	00ss1001	11	ss Reg.
ADC HL, ss	HL <- HL + ss + Fc	!!***V0!	11101101 01ss1010	15	00 BC 01 DE
SBC HL, ss	HL <- HL - ss - Fc	!!***V1!	11101101	15	10 HL 11 SP
ADD IX, pp	IX <- IX + pp	..***.0!	11011101 00pp1001	15	pp Reg. 00 BC 01 DE 10 IX 11 SP
ADD IY, rr	IY <- IY + rr	..***.0!	11111101 00rr1001	15	rr Reg. 00 BC
INC ss	ss <- ss + 1	..*.*..	11ss1011	6	01 DE
INC IX	IX <- IX + 1	..*.*..	11011101 00100011	10	10 IY 11 SP
INC IY	IY <- IY + 1	..*.*..	11111101 00100011	10	
DEC ss	ss <- ss - 1	..*.*..	00ss1011	6	
DEC IX	IX <- IX - 1	..*.*..	11011101 00101011	10	
DEC IY	IY <- IY - 1	..*.*..	11111101 00101011	10	

2.8 ROTAȚIE ȘI SHIFT

Format	Operație	Flags	Cod	T	Observații	
RLCA	shift A spre stînga bit 0(A); $F_0 \leftarrow$ bit 7(A)	..*0*.0!	00000111	4		
RLA	shift A spre stînga $F_0 \leftarrow$ bit 7(A) bit 0(A) $\leftarrow F_0$..*0*.0!	00010111	4		
RRCA	shift A spre dreapta bit 7(A); $F_0 \leftarrow$ bit 0(A)	..*0*.0!	00001111	4		
RRA	shift A spre dreapta bit 7(A) $\leftarrow F_0$..*0*.0!	00011111	4		
RLC r	shift r spre stînga bit 0(r); $F_0 \leftarrow$ bit 7(r)	!!*0*P0!	11001011 00000-r-	8		r 000 B
RLC (HL)	shift (HL) spre stînga bit 0((HL)); $F_0 \leftarrow$ bit 7((HL))	!!*0*P0!	11001011 00000110	15		001 C 010 D 011 E
RLC (IX+d)	shift (IX+d) spre stînga bit 0((IX+d)); $F_0 \leftarrow$ bit 7((IX+d))	!!*0*P0!	11011101 11001011 -d-	23		100 H 101 L 111 A
RLC (IY+d)	shift (IY+d) spre stînga bit 0((IY+d)); $F_0 \leftarrow$ bit 7((IY+d))	!!*0*P0!	11111101 11001011 -d-	23		
RL r	shift r spre stînga $F_0 \leftarrow$ bit 7(r) bit 0(r) $\leftarrow F_0$!!*0*P0!	11001011 00010-r-	8		
RL (HL)	shift (HL) spre stînga $F_0 \leftarrow$ bit 7((HL)) bit 0((HL)) $\leftarrow F_0$!!*0*P0!	11001011 00010110	15		
RL (IX+d)	shift (IX+d) spre stînga $F_0 \leftarrow$ bit 7((IX+d)) bit 0((IX+d)) $\leftarrow F_0$!!*0*P0!	11011101 11001011 -d-	23		
RL (IY+d)	shift (IY+d) spre stînga $F_0 \leftarrow$ bit 7((IY+d)) bit 0((IY+d)) $\leftarrow F_0$!!*0*P0!	11111101 11001011 -d-	23		
RRC r	shift r spre dreapta bit 7(r); $F_0 \leftarrow$ bit 0(r)	!!*0*P0!	11001011 00001-r-	8		
RRC (HL)	shift (HL) spre dreapta bit 7((HL)); $F_0 \leftarrow$ bit 0((HL))	!!*0*P0!	11001011 00001110	15		

Format	Operație	Flags	Cod	T	Observații
RRC (IX + d)	shift (IX + d) spre dreapta bit 7((IX + d)); F_c <- bit 0((IX + d))	!!*0*P0!	11011101 11001011 -d- 00001110	23	
RRC (IY + d)	shift (IY + d) spre dreapta bit 7((IY + d)); F_c <- bit 0((IY + d))	!!*0*P0!	11111101 11001011 -d- 00001110	23	
RR r	shift r spre dreapta bit 7(r) <- F_c	!!*0*P0!	11001011 00011-r-	8	
RR (HL)	shift (HL) spre dreapta bit 7((HL)) <- F_c	!!*0*P0!	11001011 00011110	15	
RR (IX + d)	shift (IX + d) spre dreapta bit 7((IX + d)) <- F_c	!!*0*P0!	11011101 11001011 -d- 00011110	23	
RR (IY + d)	shift (IY + d) spre dreapta bit 7((IY + d)) <- F_c	!!*0*P0!	11111101 00011110 -d- 00011110	23	
SRA r	shift r spre stînga F_c <- bit 7(r) bit 0(r) <- 0	!!*0*P0!	11001011 00100-r-	8	
SLA (HL)	shift (HL) spre stînga F_c <- bit 7((HL)) bit 0((HL)) <- 0	!!*0*P0!	11001011 00100110	15	
SLA (IX + d)	shift (IX + d) spre stînga F_c <- bit 7((IX + d)) bit 0((IX + d)) <- 0	!!*0*P0!	11011101 11001011 -d- 00100110	23	
SLA (IY + d)	shift (IY + d) spre stînga F_c <- bit 7((IY + d)) bit 0((IY + d)) <- 0	!!*0*P0!	11111101 11001011 -d- 00100110	23	
SRA r	shift r spre stînga F_c <- bit 0(r) bit 7(r) nu se modifică	!!*0*P0!	11001011 00101-r-	8	
SRA (HL)	shift (HL) spre stînga F_c <- bit 0((HL)) bit 7((HL)) nu se modifică	!!*0*P0!	11001011 00101110	15	
SRA (IX + d)	shift (IX + d) spre stînga F_c <- bit 0((IX + d)) bit 7((IX + d)) nu se modifică	!!*0*P0!	11011101 11001011 -d- 00101110	23	

Format	Operație	Flags	Cod	T	Observații
SRA (Y+d)	shift (Y+d) spre stînga F _c <- bit 0((Y+d)) bit 7((Y+d)) nu se modifică	!!*0*P0!	11111101 11001011 -d- 00101110	23	
SRL r	shift r spre dreapta F _c <- bit 0(r) bit 7(r) <- 0	!!*0*P0!	11001011 00111-r-	8	
SRL (HL)	shift (HL) spre dreapta F _c <- bit 0((HL)) bit 7((HL)) <- 0	!!*0*P0!	11001011 00111110	15	
SRL (X+d)	shift (X+d) spre dreapta F _c <- bit 0((X+d)) bit 7((X+d)) <- 0	!!*0*P0!	11011101 11001011 -d- 00111110	23	
SRL (Y+d)	shift (Y+d) spre dreapta F _c <- bit 0((Y+d)) bit 7((Y+d)) <- 0	!!*0*P0!	11111101 11001011 -d- 00111110	23	
RLD	bit0-3(HL) <- bit0-3(A) bit4-7(HL) <- bit0-3(HL) bit0-3(A) <- bit4-7(HL) permutare circulară	!!*0*P0.	11101101 01101111	18	
RRD	bit0-3(HL) <- bit4-7(HL) bit4-7(HL) <- bit0-3(A) bit0-3(A) <- bit0-3(HL)	!!*0*P0.	11101101 01100111	18	

2.9 SET, RESET ȘI TEST PE BIȚI

Format	Operație	Flags	Cod	T	Observații
BIT b, r	F _z <- r _b	*!*1*.0.	11001011 01-b- r-	8	r 000 Reg. B
BIT b, (HL)	F _z <- (HL) _b	*!*1*.0.	11001011 01-b-110	12	001 C 010 D
BIT b, (X+d)	F _z <- (X+d) _b	*!*1*.0.	11011101 11001011 -d- 01-b-110	10	011 E 100 H 101 L 111 A

Format	Operație	Flags	Cod	T	Observații	
BIT b, (IY+d)	$F_z < - (IY+d)_b$	**1*.0.	11111101	20	b	Bit.
			11001011		000	0
			-d-		001	1
			01-b-110		010	2
SET b, r	$r_b < - 1$..*.*..	11001011	8	011	3
			11-b- r-		100	4
SET b, (HL)	$(HL)_b < - 1$..*.*..	11001011	15	101	5
			11-b-110		110	6
SET b, (IX+d)	$(IX+d)_b < - 1$..*.*..	11011101	23	111	7
			11001011			
			-d- 11-b-110			
SET b, (IY+d)	$(IY+d)_b < - 1$..*.*..	11111101	23		
			11001011			
			-d- 11-b-110			
RES b, r	$r_b < - 0$..*.*..	11001011	8		
			10-b- r-			
RES b, (HL)	$(HL)_b < - 0$..*.*..	11001011	15		
			10-b-110			
RES b, (IX+d)	$(IX+d)_b < - 0$..*.*..	11011101	23		
			11001011			
			-d- 10-b-110			
RES b, (IY+d)	$(IY+d)_b < - 0$..*.*..	11111101	23		
			11001011			
			-d- 10-b-110			

2.10 TRANSFER

Format	Operație	Flags	Cod	T	Observații	
JP nn	PC < - nn	..*.*..	11000011	10	c	Cond. în F
			-n-		000	NZ (not 0)
			-n-		001	Z (=0)
JP c, nn	dacă c adevărat atunci PC < - nn	..*.*..	11-c-010	10	010	NC (C=0)
			-n-		011	C (C=1)
			-n-		100	PO (impar)
					101	PE (par)
					110	P (pozitiv)
	111	M (negativ)				

Format	Operație	Flags	Cod	T	Observații
JR e	PC <- PC+e	..*.*...	00011000 -e-	12	e număr cu semn în complement față de 2
JR C, e	dacă C=0 continuă altfel PC <- PC+e	..*.*...	00111000 e-2	7 12	dacă C=0 dacă C=1
JR NC, e	dacă C=1 continuă altfel PC <- PC+e	..*.*...	00110000 e-2	7 12	dacă C=1 dacă C=0
JR Z, e	dacă Z=0 continuă altfel PC <- PC+e	..*.*...	00101000 e-2	7 12	dacă Z=0 dacă Z=1
JR NZ, e	dacă Z=1 continuă altfel PC <- PC+e	..*.*...	00100000 e-2	7 12	dacă Z=1 dacă Z=0
JP (HL)	PC <- HL	..*.*...	11101001	4	
JP (IX)	PC <- IX	..*.*...	11011101 11101001	8	
JP (IY)	PC <- IY	..*.*...	11111101 11101001	8	
DJNZ	B <- B-1 dacă B=0 continuă altfel PC <- PC+e	..*.*...	00010000	8 13	dacă B=0 dacă B <> 0

2.11 SUBRUTINE

Format	Operație	Flags	Cod	T	Observații
CALL nn	SP <- SP-2 (SP+1) <- PC _H (SP+2) <- PC _L PC <- nn	..*.*...	11001101 -n- -n-	17	c Cond. în F 000 NZ (not 0) 001 Z (=0) 010 NC (C=0) 011 C (C=1) 100 PO (impar) 101 PE (par) 110 P (pozitiv) 111 M (negativ)
CALL c, nn	dacă c fals: continuă dacă c adevărat: SP <- SP-2 (SP+1) <- PC _H (SP+2) <- PC _L PC <- nn	..*.*...	11-c-100 -n- -n-	10 17	dacă c fals dacă c adevărat
RET	PC _L <- (SP) PC _H <- (SP+1) SP <- SP+2	..*.*...	11001101	10	

Format	Operație	Flags	Cod	T	Observații
RET c	dacă c fals: continuă dacă c adevărat: $PC_i \leftarrow (SP)$ $PC_h \leftarrow (SP+1)$ $SP \leftarrow SP+2$..*.*...	11-c-000	5 10	dacă c fals dacă c adevărat
RETI	$PC_i \leftarrow (SP)$ $PC_h \leftarrow (SP+1)$ $SP \leftarrow SP+2$..*.*...	11101101 01001101	14	ret. din întreruperi mascabile
RETN	$PC_i \leftarrow (SP)$ $PC_h \leftarrow (SP+1)$ $SP \leftarrow SP+2$ restaurează IFF	..*.*...	11101101 01000101	14	ret. din întreruperi nemascabile
RST p	$SP \leftarrow SP-2$ $(SP+1) \leftarrow PC_h$ $(SP+2) \leftarrow PC_i$ $PC_h \leftarrow 0$ $PC_i \leftarrow p$..*.*...	11-t-111	11	t p 000 00h 001 08h 010 10h 011 18h 100 20h 101 28h 110 30h 111 38h

2.12 INPUT ȘI OUTPUT

Format	Operație	Flags	Cod	T	Observații
IN A, (n)	$A \leftarrow (n)$..*.*...	11011011	11	n în A0 ~ A7 A în A8 ~ A15
IN r, (C)	$r \leftarrow (C)$!!*!*P0.	11101101 01-r-000	12	C în A0 ~ A7 B în A8 ~ A15 r Reg. 000 B 001 C 010 D 011 E 100 H 101 L 111 A
INI	$(HL) \leftarrow (C)$ $B \leftarrow B-1$ $HL \leftarrow HL+1$	*!*****1*	11101101 10100010	16	C în A0 ~ A7 B în A8 ~ A15 Fz conform B-1
INIR	INI pfnă cfnđ B=0	*!*****1*	11101101 10110010	21 16	dacă B < > 0 dacă B = 0

Format	Operație	Flags	Cod	T	Observații
IND	(HL) < - (C) B < - B-1 HL < - HL-1	*1****1*	11101101 10101010	16	C în A0 - A7 B în A8 - A15 Fz conform B-1
INDR	IND pînă cînd B=0	*1****1*	11101101 10111010	21 16	dacă B < > 0 dacă B = 0
OUT (n), A	(n) < - A	..*.*..	11010011 -n-	11	n în A0 - A7 A în A8 - A15
OUT (C), r	(C) < - r	..*.*..	11101101 01-r-001	12	C în A0 - A7 B în A8 - A15
OUTI	(C) < - (HL) B < - B-1 HL < - HL+1	*1****1*	11101101 10100011	16	C în A0 - A7 B în A8 - A15
OTIR	OUTI pînă cînd B=0	*1****1*	11101101 10110011	21 16	dacă B < > 0 dacă B = 0
OUTD	(C) < - (HL) B < - B-1 HL < - HL-1	*1****1*	11101101 10101011	16	C în A0 - A7 B în A8 - A15 Fz conform B-1
OTDR	OUTD pînă cînd B=0	*1****1*	11101101 10111011	21 16	dacă B < > 0 dacă B = 0

Observație: Pe lângă setul de instrucțiuni standard mai există și altele, avînd codurile în locurile rămase necompletate, din tabelele de dezasamblare. Acestea nu sînt garantate de producător, nu sînt recunoscute de asamblare, deci utilizarea lor nu este recomandabilă.

Atenție: Unele instrucțiuni pot fi codificate în mai multe moduri, avînd timpi de execuție și număr de bytes diferiți. Atenție la modul de codificare al asamblorului folosit, în procesele cu timp critic, calculat (input/output pe bandă magnetică, difuzor, border etc.).

2.13 TABELELE DE DEZASAMBLARE

2.13.1 Tabela 1 (principală), fără prefix

	0	1	2	3	4	5	6	7
00	NOP	LD BC, nn	LD (BC), A	INC BC	INC B	DEC B	LD B, n	RLCA
10	DJNZ d	LD DE, nn	LD (DE), A	INC DE	INC D	DEC D	LD D, n	RLA
20	JR NZ, d	LD HL, nn	LD (nn), HL	INC HL	INC H	DEC H	LD H, n	DAA
30	JR NC, d	LD SP, nn	LD (nn), A	INC SP	INC (HL)	DEC (HL)	LD (HL), n	SCF
40	LD B, B	LD B, C	LD B, D	LD B, E	LD B, H	LD B, L	LD B, (HL)	LD B, A
50	LD D, B	LD D, C	LD D, D	LD D, E	LD D, H	LD D, L	LD D, (HL)	LD D, A
60	LD H, B	LD H, C	LD H, D	LD H, E	LD H, H	LD H, L	LD H, (HL)	LD H, A
70	LD (HL), B	LD (HL), C	LD (HL), D	LD (HL), E	LD (HL), H	LD (HL), L	HALT	LD (HL), A
80	ADD A, B	ADD A, C	ADD A, D	ADD A, E	ADD A, H	ADD A, L	ADD A, (HL)	ADD A, A
90	SUB B	SUB C	SUB D	SUB E	SUB H	SUB L	SUB (HL)	SUB A
A0	AND B	AND C	AND D	AND E	AND H	AND L	AND (HL)	AND A
B0	OR B	OR C	OR D	OR E	OR H	OR L	OR (HL)	OR A
C0	RET NZ	POP BC	JP NZ, nn	JP nn	CALL NZ, nn	PUSH BC	ADD A, n	RST 00h
D0	RET NC	POP DE	JP NC, nn	OUT (n), A	CALL NC, nn	PUSH DE	SUB n	RST 10h
E0	RET PO	POP HL	JP PO, nn	EX (SP), HL	CALL PO, nn	PUSH HL	AND n	RST 20h
F0	RET P	POP AF	JP P, nn	DI	CALL P, nn	PUSH AF	OR n	RST 30h

	8	9	A	B	C	D	E	F
00	EX AF, AF'	ADD HL, BC	LD A, (BC)	DEC BC	INC C	DEC C	LD C, n	RRCA
10	JR d	ADD HL, DE	LD A, (DE)	DEC DE	INC E	DEC E	LD E, n	RRA
20	JR Z, d	ADD HL, HL	LD HL, (nn)	DEC HL	INC L	DEC L	LD L, n	CPL
30	JR C, d	ADD HL, SP	LD A, (nn)	DEC SP	INC A	DEC A	LD A, n	CCF
40	LD C, B	LD C, C	LD C, D	LD C, E	LD C, H	LD C, L	LD C, (HL)	LD C, A
50	LD E,	LD E,	LD E,	LD E,	LD E,	LD E,	LD E,	LD E,
60	LD L, B	LD L, C	LD L, D	LD L, E	LD L, H	LD L, L	LD L, (HL)	LD L, A
70	LD A, B	LD A, C	LD A, D	LD A, E	LD A, H	LD A, L	LD A, (HL)	LD A, A
80	ADC A, B	ADC A, C	ADC A, D	ADC A, E	ADC A, H	ADC A, L	ADC A, (HL)	ADC A, A
90	SBC A, B	SBC A, C	SBC A, D	SBC A, E	SBC A, H	SBC A, L	SBC A, (HL)	SBC A, A
A0	XOR B	XOR C	XOR D	XOR E	XOR H	XOR L	XOR (HL)	XOR A
B0	CP B	CP C	CP D	CP E	CP H	CP L	CP (HL)	CP A
C0	RET Z	RET	JP Z, nn	prefix	CALL Z, nn	CALL nn	ADC A, nn	RST 8h
D0	RET C	EXX	JP C, nn	IN A, (n)	CALL C, nn	prefix	SBC A, n	RST 18h
E0	RET PE	JP (HL)	JP PE, nn	EX DE, HL	CALL PE, nn	prefix	XOR n	RST 28h
F0	RET M	LD SP, (HL)	JP M, nn	EI	CALL PE, nn	prefix	CP n	RST 38h

2.13.2 Tabela 2; prefix: CB

	0	1	2	3	4	5	6	7
00	RLC B	RLC C	RLC D	RLC E	RLC H	RLC L	RLC (HL) A	RLC A
10	RL B	RL C	RL D	RL E	RL H	RL L	RL (HL) A	RL A
20	SLA B	SLA C	SLA D	SLA E	SLA H	SLA L	SLA (HL) A	SLA A
30	-	-	-	-	-	-	-	-
40	BIT 0, B	BIT 0, C	BIT 0, D	BIT 0, E	BIT 0, H	BIT 0, L	BIT 0, (HL)	BIT 0, A
50	BIT 2, B	BIT 2, C	BIT 2, D	BIT 2, E	BIT 2, H	BIT 2, L	BIT 2, (HL)	BIT 2, A
60	BIT 4, B	BIT 4, C	BIT 4, D	BIT 4, E	BIT 4, H	BIT 4, L	BIT 4, (HL)	BIT 4, A
70	BIT 6, B	BIT 6, C	BIT 6, D	BIT 6, E	BIT 6, H	BIT 6, L	BIT 6, (HL)	BIT 6, A
80	RES 0, B	RES 0, C	RES 0, D	RES 0, E	RES 0, H	RES 0, L	RES 0, (HL)	RES 0, A
90	RES 2, B	RES 2, C	RES 2, D	RES 2, E	RES 2, H	RES 2, L	RES 2, (HL)	RES 2, A
A0	RES 4, B	RES 4, C	RES 4, D	RES 4, E	RES 4, H	RES 4, L	RES 4, (HL)	RES 4, A
B0	RES 6, B	RES 6, C	RES 6, D	RES 6, E	RES 6, H	RES 6, L	RES 6, (HL)	RES 6, A
C0	SET 0, B	SET 0, C	SET 0, D	SET 0, E	SET 0, H	SET 0, L	SET 0, (HL)	SET 0, A
D0	SET 2, B	SET 2, C	SET 2, D	SET 2, E	SET 2, H	SET 2, L	SET 2, (HL)	SET 2, A
E0	SET 4, B	SET 4, C	SET 4, D	SET 4, E	SET 4, H	SET 4, L	SET 4, (HL)	SET 4, A
F0	SET 6, B	SET 6, C	SET 6, D	SET 6, E	SET 6, H	SET 6, L	SET 6, (HL)	SET 6, A

	8	9	A	B	C	D	E	F
00	RRC B	RRC C	RRC D	RRC E	RRC H	RRC L	RRC (HL)	RRC A
10	RR B	RR C	RR D	RR E	RR H	RR L	RR (HL)	RR A
20	SRA B	SRA C	SRA D	SRA E	SRA H	SRA L	SRA (HL)	SRA A
30	SRL B	SRL C	SRL D	SRL E	SRL H	SRL L	SRL (HL)	SRL A
40	BIT 1, B	BIT 1, C	BIT 1, D	BIT 1, E	BIT 1, H	BIT 1, L	BIT 1, (HL)	BIT 1, A
50	BIT 3, B	BIT 3, C	BIT 3, D	BIT 3, E	BIT 3, H	BIT 3, L	BIT 3, (HL)	BIT 3, A
60	BIT 5, B	BIT 5, C	BIT 5, D	BIT 5, E	BIT 5, H	BIT 5, L	BIT 5, (HL)	BIT 5, A
70	BIT 7, B	BIT 7, C	BIT 7, D	BIT 7, E	BIT 7, H	BIT 7, L	BIT 7, (HL)	BIT 7, A
80	RES 1, B	RES 1, C	RES 1, D	RES 1, E	RES 1, H	RES 1, L	RES 1, (HL)	RES 1, A
90	RES 3, B	RES 3, C	RES 3, D	RES 3, E	RES 3, H	RES 3, L	RES 3, (HL)	RES 3, A
A0	RES 5, B	RES 5, C	RES 5, D	RES 5, E	RES 5, H	RES 5, L	RES 5, (HL)	RES 5, A
B0	RES 7, B	RES 7, C	RES 7, D	RES 7, E	RES 7, H	RES 7, L	RES 7, (HL)	RES 7, A
C0	SET 1, B	SET 1, C	SET 1, D	SET 1, E	SET 1, H	SET 1, L	SET 1, (HL)	SET 1, A
D0	SET 3, B	SET 3, C	SET 3, D	SET 3, E	SET 3, H	SET 3, L	SET 3, (HL)	SET 3, A
E0	SET 5, B	SET 5, C	SET 5, D	SET 5, E	SET 5, H	SET 5, L	SET 5, (HL)	SET 5, A
F0	SET 7, B	SET 7, C	SET 7, D	SET 7, E	SET 7, H	SET 7, L	SET 7, (HL)	SET 7, A

	8	9	A	B	C	D	E	F
00	-	-	-	-	-	-	-	-
10	-	-	-	-	-	-	-	-
20	-	-	-	-	-	-	-	-
30	-	-	-	-	-	-	-	-
40	IN C, (C)	OUT (C), C	ADC HL, BC	LD BC, (nn)	-	RETI	-	LD R, A
50	IN E, (C)	OUT (C), E	ADC HL, DE	LD DE, (nn)	-	-	IM 2	LD A, R
60	IN L, (C)	OUT (C), L	ADC HL, HL	LD HL, (nn)	-	-	-	RLD
70	IN A, (C)	OUT (C), A	ADC HL, SP	LD SP, (nn)	-	-	-	-
80	-	-	-	-	-	-	-	-
90	-	-	-	-	-	-	-	-
A0	LDD	CPD	IND	OUTD	-	-	-	-
B0	LDDR	CPDR	INDR	OTDR	-	-	-	-
C0	-	-	-	-	-	-	-	-
D0	-	-	-	-	-	-	-	-
E0	-	-	-	-	-	-	-	-
F0	-	-	-	-	-	-	-	-

2.13.4 Tabela 4: prefix DD/FD pentru registrul IX/IY

Cod	Instrucțiune	Cod	Instrucțiune
09	ADD I_{xy} , BC	CBd06	RLC ($I_{xy}+d$)
19	ADD I_{xy} , DE	CBd0E	RRC ($I_{xy}+d$)
21	LD I_{xy} , nn	CBd16	RL ($I_{xy}+d$)
22	LD (nn), I_{xy}	CBd1E	RR ($I_{xy}+d$)
23	INC I_{xy}	CBd26	SLA ($I_{xy}+d$)
29	ADD I_{xy} , I_{xy}	CBd2E	SRA ($I_{xy}+d$)
2A	LD I_{xy} , (nn)	CBd3E	SRL ($I_{xy}+d$)
2B	DEC I_{xy}	CBd46	BIT 0, ($I_{xy}+d$)

34	INC ($I_{xy} + d$)	CBd4E	BIT 1, ($I_{xy} + d$)
35	DEC ($I_{xy} + d$)	CBd56	BIT 2, ($I_{xy} + d$)
36	LD ($I_{xy} + d$), n	CBd5E	BIT 3, ($I_{xy} + d$)
39	ADD I_{xy} , SP	CBd66	BIT 4, ($I_{xy} + d$)
46	LD B, ($I_{xy} + d$)	CBd6E	BIT 5, ($I_{xy} + d$)
4E	LD C, ($I_{xy} + d$)	CBd76	BIT 6, ($I_{xy} + d$)
56	LD D, ($I_{xy} + d$)	CBd7E	BIT 7, ($I_{xy} + d$)
5E	LD E, ($I_{xy} + d$)	CBd86	RES 0, ($I_{xy} + d$)
66	LD H, ($I_{xy} + d$)	CBd8E	RES 1, ($I_{xy} + d$)
6E	LD L, ($I_{xy} + d$)	CBd96	RES 2, ($I_{xy} + d$)
70	LD ($I_{xy} + d$), B	CBd9E	RES 3, ($I_{xy} + d$)
71	LD ($I_{xy} + d$), C	CBdA6	RES 4, ($I_{xy} + d$)
72	LD ($I_{xy} + d$), D	CBdAE	RES 5, ($I_{xy} + d$)
73	LD ($I_{xy} +$), E	CBdB6	RES 6, ($I_{xy} + d$)
74	LD ($I_{xy} + d$), H	CBdBE	RES 7, ($I_{xy} + d$)
75	LD ($I_{xy} + d$), L	CBdC6	SET 0, ($I_{xy} + d$)
77	LD ($I_{xy} + d$), A	CBdCE	SET 1, ($I_{xy} + d$)
7E	LD A, ($I_{xy} + d$)	CBdD6	SET 2, ($I_{xy} + d$)
86	ADD A, ($I_{xy} + d$)	CBdDE	SET 3, ($I_{xy} + d$)
8E	ADC A, ($I_{xy} + d$)	CBdE6	SET 4, ($I_{xy} + d$)
9B	SUB ($I_{xy} + d$)	CBdEE	SET 5, ($I_{xy} + d$)
9E	SBC A, ($I_{xy} + d$)	CBdF6	SET 6, ($I_{xy} + d$)
A6	AND ($I_{xy} + d$)	CBdFE	SET 7, ($I_{xy} + d$)
AE	XOR ($I_{xy} + d$)	E1	POP I_{xy}
B6	OR ($I_{xy} + d$)	E3	EX (SP), I_{xy}
BE	CP ($I_{xy} + d$)	E5	PUSH I_{xy}
		E9	JP (I_{xy})
		F9	LD SP, I_{xy}

2.13.5 Tabele de conversie: binar/zecimal/hexazecimal

Hex.	Binar	Hex.	Binar
0	0000	8	1000
1	0001	9	1001
2	0010	A	1010
3	0011	B	1011
4	0100	C	1100
5	0101	D	1101
6	0110	E	1110
7	0111	F	1111

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
10	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
20	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
30	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
40	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
50	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
60	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
70	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
80	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
90	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
A0	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
B0	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
C0	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
D0	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
E0	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
F0	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255

2.14 ASAMBLORUL GENS*

2.14.1 Instalarea

Asamblorul GENS este cel mai puternic asamblor Z80 pentru calculatoarele compatibile SPECTRUM, având următoarele caracteristici:

este relocabil, putînd fi încărcat la orice adresă în memorie
editorul este puternic și eficient
are instrucțiuni macro

Pentru încărcare și lansare în execuție se folosește secvența:

LOAD " CODE număr : RANDOMIZEUSR număr

unde număr este adresa de încărcare care trebuie să fie cu aproximativ 10 kbytes mai mică decît ultima adresă fizică.

Apeluri ulterioare: număr+56: inițializare totală
număr+61: inițializare cu păstrarea textului-sursă

Taste funcționale:

E	Enter:	Carriage return
CS1	Caps shift & 1:	ieșire din comanda-editor
CS0	Delete = Caps shift & 0:	șterge un caracter
CS8	Cursor-> = Caps shift & 8:	avans tab
CS5	Cursor<- = Caps shift & 5:	șterge toată linia

*GENS3M21 este produs înregistrat al firmei HISOFT

2.14.2 Editorul

Prompt-ul > așteaptă o comandă de tipul:

comandă număr-1, număr-2, șir-1, șir-2

unde *număr-1, număr-2* sînt numere întregi între 1 și 32767; *șir-1, șir-2* sînt șiruri de caractere de lungime între 0 și 20. Blank-ul nu este semnificativ decît în șiruri; , este separator.

GENERAREA TEXTULUI-SURSĂ

I *număr-1, număr-2*

Introduce numărătoarea automată a liniilor.

Valori implicite: *număr-1* = 10, *număr-2* = 10.

O linie text-sursă este de forma:

număr-linie etichetă instrucțiune operanzi ; comentarii

La fiecare structură, cu excepția primelor 2, se ajunge cu tab-ul; la terminarea unei linii se apasă E.

Orice linie nouă introdusă, cu comanda I sau direct după prompt-ul > se suprapune peste textul anterior astfel încît liniile anterioare cu același număr de linii se șterg.

LISTARE

L *număr-1, număr-2*

Listează liniile textului-sursă avînd numerele între *număr-1* și *număr-2*; orice tastă efectuează scroll, cu excepția CS1.

Valori implicite: *număr-1* = 1, *număr-2* = 32767

K *număr*

Setează numărul de linii de listat înainte de scroll.

Valori implicite: *număr* = 16

EDITARE

D *număr-1, număr-2*

Șterge liniile textului-sursă, avînd numerele între *număr-1* și *număr-2*.

Valori implicite: fără

Dacă *număr-1* > *număr-2* sau lipsesc argumente comanda este ignorată..

M *număr-1, număr-2*

Introduce linia *număr-1* la linia *număr-2*; linia *număr-1* rămîne în text; linia *număr-2* anterioară se șterge.

Valori implicite: fără

N *număr-1, număr-2*

Renumerotează textul-sursă începînd cu *număr-1* și cu pasul *număr-2*

Valori implicite: fără

F număr-1, număr-2, șir-1, șir-2

șir-1 este căutat în liniile cu numerele între număr-1 și număr-2; dacă este găsit atunci se intră în editorul de linie (comanda E) cu cursorul pe primul caracter al șir-1; șir-2 substituie șir-1.

Valori implicite: cele aflate în buffer la o utilizare anterioară, altfel fără

E număr

Editor de linie.

Valori implicite: fără

Dacă număr lipsește, comanda este ignorată.

Sub-comenzi:

CS0 cursor stînga

CS8 avans tab

E terminarea editării, cu modificări

C supra-scriere, începînd de la poziția curentă a cursorului

F caută următorul șir-1 din comanda F

I înserează caractere la poziția curentă a cursorului

K șterge un caracter la poziția cursorului

L listează restul liniei în curs de editare

Q terminarea editării, modificările sînt anulate

R anulează modificările

S înlocuiește șir-1 din comanda F

X avansează cursorul la sfîrșitul liniei și introduce sub-comanda I

Z șterge toate caracterele de la cursor la sfîrșitul liniei

COMENZI DE BANDĂ MAGNETICĂ**P număr-1, număr-2, șir**

Salvează pe bandă magnetică liniile din textul-sursă între număr-1 și număr-2, într-un fișier cu numele șir.

Valori implicite: cele aflate în buffer la o utilizare anterioară, altfel fără.

G , , șir

Încarcă de pe bandă magnetică fișierul (text-sursă) șir la sfîrșitul textului-sursă și liniile sînt renumerotate.

Valori implicite: dacă șir lipsește se încarcă primul fișier

T număr-1, număr-2, șir

Salvează liniile, avînd numărul între număr-1 și număr-2 cu numele șir în formatul utilizabil de directiva de asamblare *F.

COMENZI DE UZ GENERAL**B**

Retur în BASIC Spectrum.

C

Interoghează utilizatorul asupra buffer-ului utilizat de directiva de asamblare *F (Include buffer) și a celui utilizat de instrucțiunile Macro (Macro buffer). Bufferele

R

Rulează codul asamblat avînd adresa de start, data de directiva de asamblare ENT. Cu ajutorul instrucțiunii RET se poate reintra în asamblor, la sfîrșitul rulării, dacă stiva microprocesorului este restaurată la ieșire.

2.14.3 DIRECTIVE DE ASAMBLARE**EXPRESII**

O expresie este o listă de termeni separați de operatori, unde:

termen:	constantă zecimală
	constantă hexazecimală precedată de #
	constantă binară precedată de %
	caracter încadrat între ghilimele
	etichetă, maxim 6 caractere, care începe cu o literă
	\$, semnifică adresa curentă
operator:	+ adunare
	- scădere
	& operația AND
	@ operația OR
	* înmulțire între întregi
	/ împărțire între întregi
	?, operația MODULO: $x?y = x - (x/y)*y$

Din numere se consideră partea MODULO 65535 (#10000); expresiile sînt evaluate de la stînga la dreapta, fără priorități; o expresie în paranteză, ca operand, semnifică o adresă..

DIRECTIVE**DEFB *expresie-1, ...***

Fiecare expresie,evaluată la un byte, este stocată la adresa curentă a DEFB.

DEFS *expresie*

Rezervă, începînd de la adresa curentă, un număr de bytes egal cu valoarea *expresie*.

DEFS "*șir*"

Stochează în memorie, începînd de la adresa curentă, codurile ASCII ale caracterelor care formează șirul (în ordinea în care apar în șir).

DEFW *expresie-1,...*

Fiecare expresie, evaluată la 2 bytes, este stocată la adresa curentă a DEFW.

ELSE

Dacă liniile de dinainte de ELSE nu se assemblează (ca urmare a IF) atunci liniile de după ELSE vor fi asamblate și reciproc; se utilizează numai în structurile IF ... ELSE ... END, unde este opțional.

END

Marchează sfârșitul structurii IF ... ELSE ... END.

ENT *expresie*

Adresa de start, pentru comanda R, are valoarea *expresie*.

eticheta EQU *eticheta*

Expresia, de tip număr, poate fi înlocuită cu eticheta.

IF *expresie*

Dacă *expresie*=0 liniile care urmează după IF, pînă la ELSE, END sau sfârșitul textului, nu sînt asamblate.

ORG *expresie*

Următoarele linii de text-sursă vor fi asamblate începînd de la adresa *expresie*, pînă la întîlnirea unei noi ORG sau pînă la sfârșitul textului-sursă.

; *comentariu*

Caracterele care urmează după ; sînt ignorate la asamblare (se consideră comentariu).

Observație: IF/ELSE/END se utilizează în mod normal în forma:

```
IF expresie
  text-sursa-1
ELSE
  text-sursa-2
END
```

MACRO-INSTRUCȚIUNI

Definirea macro-instrucțiunilor:

```
nume MAC
  ...
  ... (corpul definiției)
  ...
  ENDM
```

O definiție macro poate avea pînă la 32 de parametri, notați în definiție, cu simbolurile =0, ..., =31.

Apelul unei macro-instrucțiuni se face astfel:

```
nume parametru-0, ...
```

parametrii putînd fi utilizați și în expresii.

2.14.4 Codul erorilor generate

- | | |
|----|---|
| 1 | eroare în linie |
| 2 | mnemonic nerecunoscut |
| 3 | declarație eronată |
| 4 | simbol definit de mai multe ori |
| 5 | caracter ilegal în linia respectivă |
| 6 | operand ilegal în linia respectivă |
| 7 | simbolul utilizat este cuvânt rezervat (mnemonic, nume-registru sau directivă-asamblor) |
| 8 | registri nepotriviți |
| 9 | prea mulți registri |
| 10 | o expresie care ar fi trebuit să fie evaluată la un byte are o valoare mai mare |
| 11 | instrucțiunea JP (IX+d) sau JP (IY+d) nu există |
| 12 | eroare în directiva asamblor |
| 13 | simbol în EQU care nu a fost definit încă |
| 14 | împărțire cu 0 |
| 15 | depășire la înmulțire |
| 16 | lipsește definiția macro-instrucțiunii |
| 17 | numele nu este identificator al unei macro-instrucțiuni |
| 18 | lipsește un apel macro |
| 19 | lipsește declarația condițională |

2.15 DEZASAMBLORUL MONS*

2.15.1 Instalarea

Dezasamblorul MONS este destinat să lucreze în tandem cu asamblorul GENS astfel încât este relocabil.

Pentru încărcare și lansare în execuție se folosește secvența:

LOAD "" CODE număr : RANDOMIZE USR număr

unde număr este adresa de încărcare. Necesarul de memorie pentru MONS este de aproximativ 6Kbytes.

Apel ulterior: **USR număr + 24**

Ecranul principal are structura următoare:

*MONS3M21 este produs înregistrat al firmei HISOFT

adresa-curentă	cod	Instrucțiune			
PC	număr-1	nr-1	...	nr-7	(7 numere , de la adr. PC)
SP	număr-2	.		.	
IV	număr-3	.		.	
IX	număr-4	.		.	
HL	număr-5				
DE	număr-6				
BC	număr-7	nr-42	...	nr-49	(7 numere, de la adr. BC)
AF	număr-8	ZS H P/V NC	(sint afișate		
IR	număr-9		numai		
			faniioanele		
			setate)		
adresa-1	n-1		adresa-9	n-9	adresa-17 n-17
.			> adr. curentă	n13<	
.					
adresa-8	n-8		adresa-16	n-16	adresa-24 n-24
>					

Prompt-ul inferior semnifică așteptarea unei comenzi.

2.15.2 Comenzi

Enter

Incrementează adresa curentă cu 1.

CS și 5 (cursor ^)

Decrementează adresa curentă cu 8.

CS și 7 (cursor ->)

Decrementează adresa curentă cu 1.

CS și 8 (cursor <-)

Incrementează adresa curentă cu 8.

SS și 3

Comută zecimal/hexazecimal modul de afișare al adreselor și regiștrilor PC, ..., BC.

SS și 4

Comută ecran principal/pagina de cod dezasamblat în mod scroll.

SS și P

Tipărește un bloc de memorie, la imprimantă, de la adresa curentă și caracterele corespunzătoare, în standardul ASCII (pentru codurile mai mari decât 127 bitul 7 este resetat apoi pentru codurile mai mici decât 32 se afișează .). CS și 5 efectuează reîntoarcere la ecranul principal; CS și 1 retur în BASIC Spectrum; orice alta tastă efectuează scroll.

SS și T

Introduce un punct de control după instrucțiunea curentă și continuă execuția. Punctele de control introduse de comanda W se pierd. PC și adresa curentă trebuie să coincidă.

SS și Z

Realizează rularea programului instrucțiune cu instrucțiune; PC și adresa curentă trebuie să coincidă.

Adresa curentă devine egală cu numărul de pe stivă.

G

Caută șir: fiecare byte hexazecimal este dat pe rând prin tastarea Enter; șirul se termină cu orice caracter care nu este cifră hexazecimală.

H

Convertește un număr decimal în hexazecimal; numărul se termină cu orice caracter care nu este număr hexazecimal.

I

Copiază un bloc de memorie; **First:** adresa de început, **Last:** adresa de sfârșit, **To:** adresa destinație. Se copiază inclusiv blocurile care rescriu peste vechiul bloc.

J

Salt și execuție la adresa specificată; dacă se dorește întoarcerea în dezasamblor se introduce un punct de control (comanda W). Comanda J nu păstrează conținutul actual al regiștrilor.

K

Continuă execuția de la adresa curentă, nu afectează conținutul regiștrilor, dacă se dorește reîntoarcerea în dezasamblor se introduce un punct de control (comanda W).

L

Listează un bloc de memorie, de la adresa curentă, și caracterele corespunzătoare, în standardul ASCII (pentru codurile mai mari decât 127 bitul 7 este resetat apoi pentru codurile mai mici decât 32 se afișează .). CS și 5 efectuează reîntoarcere la ecranul principal; CS și 1 retur în BASIC Spectrum; orice alta tastă efectuează scroll.

M

Modifică adresa curentă conform celei furnizate de utilizator în hexazecimal.

N

Caută locul următor în care se găsește șirul specificat de ultima comandă G.

O

Byte-ul de la adresa curentă se consideră o adresare relativă (în instrucțiunea JR) și adresa curentă se modifică în conformitate cu acesta (reîntoarcere cu comanda U).

P

Umple un bloc de memorie de la adresa **First:** la adresa **Last:** cu byte-ul **With:**.

Q

Comută pe setul alternat de regiștri.

T

Dezasamblează un bloc de memorie de la adresa **First:** la adresa **Last:** la imprimantă dacă se introduce y sau pe ecran fără scroll. **Text:** dezassemblează text, Enter, simplu, anulează comanda.

U

Reîntoarcere la adresa curentă (inițială) după comanda O.

V

Reîntoarcere la adresa curentă (inițială) după comanda X.

W

Introduce un punct de control (oprire) la adresa curentă, aceasta fiind de forma **CALL număr** (3 bytes). După utilizarea acestui punct de control cei 3 bytes sînt refăcuți. Se pot utiliza oricîte puncte de control, fiecare necesitînd 5 bytes spațiu liber în memorie după MONS.

X

Cei 2 bytes de la adresa curentă se consideră o adresa de salt (instrucțiunile CALL și JP) și adresa curentă se modifică în conformitate cu aceștia (reîntoarcere cu comanda V).

Y

Permite introducerea unui șir alfanumeric care este introdus începînd de la adresa curentă a memoriei. Șirul se termina cu CS și 5. Delete permite ștergerea caracterelor. Adresa curentă devine prima adresă după șir.

Modificarea memoriei

Conținutul adresei curente poate fi modificat direct prin introducerea unui număr hexazecimal. După introducerea numărului adresa curentă se incrementează. Numărul nu trebuie să se termine cu un simbol care reprezintă o comandă.

Modificarea regiștrilor

. deplaseaza cursorul registru (ecranul principal: stînga sus) prin permutare circulară. *număr* . introduce *număr* în registrul curent.

3.1 BASIC SPECTRUM

3.1.1 Constante, variabile, expresii

Constantă numerică

Format standard:

semn cifră. cifră...cifră E semn cifră cifră

semn este + (opțional) sau -. *cifră* este un element din mulțimea {0, ..., 9}. Cifrele din fața punctului zecimal și exponentul pot lipsi dacă sînt egale cu 0. Numerele sînt afișate în formatul standard.

Constante-șir

Format:

șir

Unde *șir* este o serie de caractere afișabile (cu cursorul L, C, E, G cu sau fără SS).

Variabile numerice

Variabilele care înlocuiesc o valoare numerică unică pot fi formate din mai multe litere și cifre; trebuie să înceapă cu o literă; nu se face distincția între literele mari și mici.

Variabilele care înlocuiesc tablouri-numerice pot fi formate dintr-o singură literă; nu se face distincția între literele mari și mici.

Variabile-șir

Variabilele care înlocuiesc șiruri sau tablouri-șir pot fi formate dintr-o singură literă; nu se face distincția între litere mari și mici.

Observație: numele unei variabile numerice, șir sau contor pot coincide fără a exista confuzii/erori.

Expresii

Expresiile sînt liste de constante, variabile, funcții BASIC și funcții definite de programator, separate de operatori și paranteze (,).

Expresiile sînt evaluate de la stînga la dreapta, conform ordinii de prioritate a operatorilor.

Parantezele (,) se utilizează pentru a semnaliza inversarea ordinii de aplicare a priorității operatorilor. Modul de evaluare a unei expresii în ordinea descrescătoare a priorităților este:

- se evaluează subexpresiile din paranteze
- se evaluează funcțiile
- se evaluează subexpresiile aritmetice; ordinea de prioritate este:
 - ^ ridicare la putere (prioritate maximă)
 - *, / înmulțire, împărțire
 - +, - adunare, scădere (prioritate minimă)
- se evaluează subexpresiile operaționale; operatorii operaționali:
 - >, =, <, <=, >=, <> (au aceeași ordine de prioritate)
- se evaluează subexpresiile logice; ordine de prioritate:
 - NOT (prioritate maximă)
 - AND
 - OR (prioritate minimă)

Expresiile sînt de 2 tipuri după rezultatul evaluării:

expresii-numerice generează numere
expresii-șir generează șiruri

3.1.2 Cuvinte cheie BASIC Spectrum

Cuvintele cheie BASIC Spectrum sînt de 3 tipuri: comenzi, funcții și operatori (cu destinații diverse).

ABS

Format:

ABS constantă / variabilă-numerică sau
ABS (expresie-numerică)

ABS este funcție și returnează valoarea absolută (modulul) numărului.

ACS

Format:

ACS constantă / variabilă-numerică sau
ACS (expresie-numerică)

ACS este funcție și returnează arccosinusul numărului (între -1 și 1) în radiani.

AND

Format:

condiție-1 AND condiție-2 (a) sau
expresie-numerică-1 AND expresie-numerică-2 (b) sau
expresie-șir AND expresie-numerică (c)

unde condiție-1/2 sînt expresii care prin evaluare (a operatorilor operaționali sau logici) pot fi false sau adevărate. AND este funcție și returnează:

(a) adevărat (funcție logică) dacă condiție-1/2 sînt amîndouă adevărate; altfel fals

- (b) *valoare-numerică-1* a expresiei dacă *valoare-numerică-2* a expresiei-2 este nenulă
- (c) *valoare-șir* a expresiei-șir dacă *valoare-numerică* a expresiei-numerice este nenulă

ASN

Format:

ASN *constantă/variabilă-numerică* sau**ASN** (*expresie-numerică*)

ASN este funcție și returnează arcsinusul numărului (între -1 și 1) în radiani.

AT

Format:

AT *linie, coloană*

AT este operator și stabilește poziția de afișare în comenzile INPUT și PRINT. *linie* este numărul liniei, între 0 și 21, de sus în jos; *coloană* este numărul coloanei, între 0 și 31, de la stînga la dreapta. *linie* și *coloană* pot fi constante, variabile sau expresii numerice, rotunjite dacă este cazul la numărul întreg cel mai apropiat.

ATN

Format:

ATN *constantă/variabilă-numerică* sau**ATN** (*expresie-numerică*)

ATN este funcție și returnează arctangentul numărului în radiani.

ATTR

Format:

ATTR (*linie, coloană*)

ATTR este funcție și returnează valoarea atributului aflat în poziția (*linie, coloană*); *linie* este numărul liniei, între 0 și 21, de sus în jos; *coloană* este numărul coloanei, între 0 și 31, de la stînga la dreapta. *linie* și *coloană* pot fi constante, variabile sau expresii numerice, rotunjite dacă este cazul la numărul întreg cel mai apropiat. Atributul unui caracter este:

Cod culoare:		Cod BRIGHT	Cod FLASH
0 negru	4 verde	0 normal	0 normal
1 roșu	5 cyan	1 strălucitor	1 cliptor
2 albastru	6 galben		
3 magenta	7 alb		

BEEP

Format:

BEEP *expresie-numerică-1, expresie-numerică-2*

BEEP este comandă și emite un sunet la difuzor. *expresie-numerică-1* reprezintă durata în secunde, între 0 și 10; *expresie-numerică-2* reprezintă înălțimea notei, între -60 și 69. Diferența de o unitate între înălțimile a 2 note reprezintă o diferență de un semiton. Nota DO, inferior, din cheia SOL are înălțimea 0.

BIN

Format:

BIN *număr-binar*

BIN este funcție și returnează valoarea numărului binar în zecimal (între 0 și 65535 zecimal = număr binar cu maxim 16 cifre binare, 0 sau 1).

BORDER

Format:

BORDER *expresie-numerică-pozitivă*

BORDER este comandă și schimbă culoarea marginii conform valorii expresiei numerice, între 0 și 7, eventual aceasta fiind rotunjită la întregul cel mai apropiat.

Cod culoare:

0 negru	4 verde
1 roșu	5 cyan
2 albastru	6 galben
3 magenta	7 alb

BRIGHT

Format:

BRIGHT *expresie-numerică-positivă*

BRIGHT este comandă/operator și permite comandarea intensității luminoase a afișării conform valorii expresiei numerice, eventual aceasta fiind rotunjită la valoarea întreagă cea mai apropiată:

<i>expresie-numerică</i> = 0	intensitate luminoasă normală
<i>expresie-numerică</i> = 1	intensitate luminoasă marită (strălucitor)
<i>expresie-numerică</i> = 8	opțiunile BRIGHT 0/1 făcute anterior asupra pozițiilor de afișare rămân valabile la afișări ulterioare

CAT

Comandă microdrive.

CHR\$

Format:

CHR\$ *constantă /variabilă-numerică-positivă* sau

CHR\$ (*expresie-numerică-positivă*)

CHR\$ este funcție și returnează caracterul al cărui cod este dat de *număr*. Eventual numărul este rotunjit la numărul întreg cel mai apropiat.

CIRCLE

Format:

CIRCLE *listă-opțiuni expresie-numerică-positivă-1, expresie-numerică-positivă-2, expresie-numerică-positivă-3*

unde *listă-opțiuni* este opțională și este de forma:

opțiune-afișare ; ...

și *opțiune-afișare* este de tipul: PAPER, INK, BRIGHT, OVER, FLASH sau INVERSE.

CIRCLE este comandă și trasează pe ecran un cerc aproximativ:

expresie-numerică-positivă-1 reprezintă coordonata x a centrului; valori extreme de la 0 la 255, de la stînga la dreapta

expresie-numerică-positivă-2 reprezintă coordonata y a centrului; valori extreme de la 0 la 175, de jos în sus

expresie-numerică-positivă-3 reprezintă raza, valori extreme de la 0 la 87

Expresiile numerice sînt eventual rotunjite la valoarea întreagă cea mai apropiată și trebuie să aibă valorile astfel încît cercul să se încadreze în ecran. Cercul este trasat din extremitatea stîngă în sens orar.

CLEAR

Format:

CLEAR *expresie-numerică-positivă*

CLEAR este o comandă și realizează următoarele operații:

- șterge toate variabilele BASIC
- șterge ecranul
- atribuie variabilei-sistem RAMTOP valoarea *expresiei-numerică* eventual rotunjită la valoarea întreagă cea mai apropiată; expresia numerică poate lipsi caz în care se consideră valoarea implicită 65367.

RAMTOP reprezintă adresa ultimei locații utilizabile de programul BASIC.

CLOSE#

Comandă microdrive.

CLS

Format:

CLS

CLS este comandă și realizează ștergerea ecranului și resetarea atributelor conform culorilor INK și PAPER permanente.

CODE

Format:

CODE constantă /variabilă-șir

CODE (expresie-șir)

CODE expresie-numerică-pozitivă-1, expresie-numerică-pozitivă-2

(a)

(b)

(c)

CODE este funcție/operator și realizează:

(a),(b) funcție care returnează codul primului caracter al șirului (codurile caracterelor afișabile)

(c) operator utilizat în comenzile LOAD, SAVE și VERIFY pentru manipularea blocurilor de bytes pe bandă magnetică
 expresie-numerică-pozitivă-1 reprezintă adresa primului byte
 expresie-numerică-pozitivă-2 reprezintă numărul de bytes

CONTINUE

Format:

CONTINUE

CONTINUE este comandă și este utilizată pentru continuarea rulării programului după o comandă STOP sau BREAK.

COPY

Format:

COPY

COPY este comandă și realizează copia ecranului la imprimantă.

COS

Format:

COS constantă /variabilă-numerică

sau

COS (expresie-numerică)

COS este funcție și returnează valoarea cosinusului numărului considerat unghi măsurat în radiani.

DATA

Format:

DATA expresie-numerică /șir , ...

(a)sau

DATA literă ()

(b)sau

DATA literă\$ ()

(c)

DATA este comandă/operator și realizează:

(a) comandă: utilizată pentru generarea datelor citite de comanda READ; pot fi puse oriunde în program, respectînd ordinea lor de succesiune

(b),(c) operator: utilizat de comenzile LOAD, SAVE și VERIFY pentru manipularea tablourilor numerice/șir pe bandă magnetică

Variabilele din DATA trebuie să aibă valori în momentul citirii READ dacă DATA este utilizată ca o comandă.

DEF FN

Format:

DEF FN literă listă-variabile = expresie-numerică sau

DEF FN literă \$ listă-variabile = expresie-șir

unde *listă-variabile*, opțională, are formatul:

(*variabilă-numerică /șir , ...*)

DEF FN este comandă și este utilizată pentru definirea unei funcții-utilizator numerice sau șir. Funcția definită depinde de variabilele din listă.

DIM

Format:

DIM literă (dimensiune-1, ...) sau

DIM literă \$ (dimensiune-1, ...)

unde *dimensiune* este de forma:

expresie-numerică-pozitivă

în care eventualele variabile trebuie să aibă valori în momentul evaluării; expresiile numerice sînt rotunjite la numărul întreg cel mai apropiat; numărul dimensiunilor este nelimitat.

DIM este comandă și realizează creerea și inițializarea variabilelor tablou numerice/șir cu 0/blank-uri.

DRAW

Format:

DRAW listă-opțiuni expresie-numerică-1, expresie-numerică-2 (a) sau

DRAW listă-opțiuni expresie-numerică-1, expresie-numerică-2, (b)

expresie-numerică-3

unde *listă-opțiuni* este opțională și este de forma:

opțiune-afișare ; ...

și *opțiune-afișare* este de tipul PAPER, INK, BRIGHT, OVER, FLASH sau INVERSE.

DRAW este comandă și efectuează pe ecran:

(a) trasarea unei drepte între punctele de coordonate (X0, Y0) și

(X0 + *expresie-numerică-1*, Y0 + *expresie-numerică-2*)

(b) trasarea unui arc de cerc între punctele de coordonate (X0, Y0) și

(X0 + *expresie-numerică-1*, Y0 + *expresie-numerică-2*) a cărui unghi la

centru măsurat în radiani este *expresie-numerică-3*; dacă *expresie-numerică-3* > 0 trasarea se face în sens trigonometric, altfel în sens orar.

unde (X0, Y0) reprezintă coordonatele ultimului pixel trasat de o comandă PLOT, DRAW, CIRCLE anterioară, (0, 0) inițial sau după CLS.

x măsurat de la stînga la dreapta, de la 0 la 255

y măsurat de jos în sus, de la 0 la 175

ERASE

Comandă microdrive.

FLASH

Format:

FLASH expresie-numerică-pozitivă

FLASH este comandă/operator și permite comandarea afișării clipitoare conform valorii expresiei numerice, eventual rotunjită la valoarea întreagă cea mai apropiată.

expresie-numerică = 1: afișare clipitoare

expresie-numerică = 0: afișare normală

expresie-numerică = 8: opțiunile făcute anterior asupra pozițiilor de afișare rămîn neschimbate la afișări ulterioare.

FN

Format:

FN literă listă-expresii sau
FN literă \$ listă-expresii

unde *listă-variabile* are formatul:

(*expresie-numerică* / *șir* , ...)

FN este funcție și returnează valoarea funcției definite prin DEFN respectiv. În momentul evaluării variabilelor din *lista-variabile* (DEFN) li se atribuie valorile obținute în urma evaluării expresiilor, în ordinea din liste.

FOR

Format:

FOR literă = expresie-numerică-1 TO expresie-numerică-2 opțiune-pas

unde *opțiune-pas* este de tipul STEP.

FOR este comandă și permite realizarea buclelor cu contor.

Variabila-contor este literă și este incrementată cu 1, dacă nu a fost specificat alt pas prin *opțiune-pas*; începînd cu *expresie-numerică-1*, atîta timp cît este mai mică sau egală cu *expresie-numerică-2*. Corpul buclei este cuprins între instrucțiunea FOR și instrucțiunea NEXT corespunzătoare. Sînt permise oricîte bucle îmbricate, buclele încrucișate sînt interzise.

FORMAT

Comandă microdrive.

GO SUB

Format:

GO SUB expresie-numerică-positivă

GO SUB este comandă și permite saltul la o subrutină care începe la numărul de linie egal cu *expresie-numerică* care eventual este rotunjită la numărul întreg cel mai apropiat. Dacă nu există un astfel de număr de linie atunci programul continuă cu linia avînd numărul imediat următor. Reîntoarcerea în rutina apelantă se execută cu RETURN.

GO TO

Format:

GO TO expresie-numerică-positivă

GO TO este comandă și permite saltul la o linie care are numărul egal cu *expresie-numerică* care este eventual rotunjită la numărul întreg cel mai apropiat. Dacă nu există un astfel de număr de linie atunci programul continuă cu linia avînd numărul de linie imediat următor.

IF

Format:

IF condiție THEN listă-instrucțiuni

unde *condiție* este o expresie care poate fi adevărată sau falsă și *listă-instrucțiuni* are formatul:

instrucțiune : ...

IF este comandă și permite execuția condiționată a instrucțiunilor:

- a) condiție adevărată: se execută *listă-instrucțiuni*
 - b) condiție falsă: nu se execută *listă-instrucțiuni*
- listă-instrucțiuni* este opțională.

IN

Format:

IN constantă / variabilă-numerică sau

IN (expresie-numerică)

IN este funcție și returnează ca valoare byte-ul citit la portul de intrare a cărui adresă este numărul respectiv.

Porturile tastaturii au adresa de tipul: $256*(255-2n) + 254$; n: {0,...,7}

0 CS...V	4 0...6
1 A...G	5 P...Y
2 Q...T	6 Enter...H
3 1...5	7 Space...B

INK

Format:

INK *expresie-numerică*

INK este comandă/operator și permite controlul culorii de scriere, conform *expresiei-numerică* care eventual este rotunjită la valoarea întregă cea mai apropiată.

Culori:	0 negru	5 cyan
	1 albastru	6 galben
	2 roșu	7 alb
	3 magenta	8 transparent
	4 verde	9 contrast

transparent: culoarea INK nu se modifică la afișări ulterioare

contrast: culoarea INK este negru/alb vizibil pe culoarea PAPER

INKEY\$

Format:

INKEY\$

INKEY\$ este funcție și returnează ca valoare un șir cu un caracter, dat de tasta care este apăsată în momentul evaluării; dacă nu a fost apăsată nici o tastă se returnează șirul vid; nu se așteaptă apăsarea unei taste; se detectează combinații de taste cu CS și SS.

INPUT

Format:

INPUT *element-input separator ...*

unde *element-input* este de forma:

element separator ... sau
(element separator ...)

a) (,) determină afișarea informațiilor dintre paranteze

b) element este de forma:

constantă / variabilă-numerică / șir

opțiune-afișare: PAPER, INK, BRIGHT, OVER, FLASH sau INVERSE

LINE *constantă / variabilă-șir*: determină suprimarea ghilimelelor la afișare

AT *linie, coloană* : pentru poziția de afișare

și separator este de forma:

; poziția următoare de afișare este în continuare, este separator pentru opțiunile de afișare

, poziția următoare de afișare este în coloana 0/16

, poziția următoare de afișare este în linia următoare

INPUT este comandă și permite introducerea datelor de la tastatură de către utilizator, în timpul rulării programului.

INT

Format:

INT *constantă / variabilă-numerică* sau

INT (*expresie-numerică*)

INT este funcție și returnează partea întregă a numărului.

INVERSE

Format:

INVERSE *expresie-numerică-pozitivă*

INVERSE este comandă/operator și permite afișarea în culori inversate (culoarea INK se înlocuiește cu PAPER și reciproc):

- 0 afișare normală
- 1 afișare inversată

LEN

Format:

LEN *constantă/variabilă-șir* sau
LEN (*expresie-șir*)

LEN este funcție și returnează lungimea șirului.

LET

Format:

LET *variabilă-numerică/șir = expresie-numerică/șir*

LET este comandă și permite asignarea variabilelor cu valorile expresiilor.

LINE

Format:

LINE *constantă/variabilă-șir* (a) sau
LINE *expresie-numerică-pozitivă* (b)

LINE este opțiune și permite:

- (a) suprimarea ghilimelelor de încadrare a șirului în comandă INPUT
- (b) autorularea programelor, la încărcare (LOAD), la numărul de linie dat de expresie, eventual rotunjită la numărul întreg cel mai apropiat, dacă acestea au fost salvate (SAVE) cu această opțiune; dacă un astfel de număr de linie nu există atunci programul începe cu linia având numărul imediat următor

LIST

Format:

LIST *expresie-numerică-pozitivă*

LIST este comandă și permite listarea programului pe ecran începând cu numărul de linie dat de *expresie-numerică*, eventual aceasta fiind rotunjită la numărul întreg cel mai apropiat. *Expresia-numerică* este opțională, valoare implicită 0. Poziționează cursorul de selectare pentru editarea liniei.

LLIST

Format:

LLIST *expresie-numerică-pozitivă*

LLIST este comandă și permite listarea programului la imprimantă, începând cu numărul de linie dat de *expresie-numerică*, eventual aceasta fiind rotunjită la numărul întreg cel mai apropiat. *Expresia-numerică* este opțională, valoare implicită 0.

LN

Format:

LN *constantă/variabilă-numerică-pozitivă* sau
LN (*expresie-numerică-pozitivă*)

LN este funcție și returnează ca valoare logaritmul natural al numărului.

LOAD

Format:

LOAD "*expresie-șir*" *opțiune-date*

unde *opțiuni-date* sînt opționale și sînt de tipul CODE, DATA sau SCREEN\$.

LOAD este comandă și permite încărcarea fișierelor de pe bandă magnetică. *Expresie-șir* este opțională și dacă lipsește se încarcă primul fișier de pe bandă magnetică. Dacă *opțiunea-date* lipsește se încarcă un program BASIC. Dacă parametrii opțiunii CODE lipsesc atunci blocul de bytes se încarcă conform parametrilor salvați.

LPRINT

Format:

LPRINT *listă-opțiuni element-imprimare separator ...*

listă-opțiuni este de forma:

opțiune-imprimare ; ...

opțiune-imprimare este de tipul AT, TAB, INVERSE sau OVER

element-imprimare este de forma:

expresie-numerică /șir

separator este:

; poziția următoare de tipărire este în continuare

, poziția următoare de tipărire este în coloana 0/16

' poziția următoare de tipărire este în linia următoare

LPRINT este comandă și permite tipărirea la imprimantă.

MERGE

Format:

MERGE "*expresie-șir*"

MERGE este comandă și permite concatenarea programului BASIC din memorie cu unul înregistrat pe bandă magnetică. MERGE încarcă un program de pe bandă magnetică și îl concatenează cu cel din memorie, intercalînd, în ordine, numerele de linie. Linile din programul din memorie se șterg dacă există în programul de pe bandă linii cu același număr de linie. MERGE stopează autorularea programelor.

MOVE

Comandă microdrive.

NEW

Format:

NEW

NEW este comandă care realizează restart-ul soft al calculatorului.

NEXT

Format:

NEXT *literă*

NEXT este comandă și închide bucla deschisă de comanda FOR, căreia îi corespunde, corespondența făcîndu-se prin numele variabilei-contor de buclă.

NOT

Format:

NOT *condiție*

unde condiție este o expresie care prin evaluare poate fi falsă sau adevărată (prin evaluarea operatorilor operaționali sau logici).

NOT este funcție și returnează:

a) fals dacă *condiție* este adevărată

b) adevărat dacă *condiție* este falsă

OPEN#

Comandă microdrive

OR

Format:

condiție-1 OR condiție-2

unde *condiție-1/2* sînt expresii care prin evaluare pot fi false sau adevărate (prin evaluarea operatorilor operaționali sau logici).

OR este funcție logică și returnează fals dacă *condiție-1/2* sînt amîndouă false, altfel adevărat.

OUT

Format:

OUT *expresie-numerică-positivă-1, expresie-numerică-positivă-2*

OUT este comandă și permite trimiterea unui byte dat de evaluarea *expresiei-numerică-2* la portul de adresă *expresie-numerică-1*; expresiile fiind eventual rotunjite la numărul întreg cel mai apropiat.

Adrese de porturi:

254	BIN D7D6D5D4D3D2D1D0:	D4 difuzor D3 microfon D2D1D0 culoare BORDER
251	imprimantă	
247,239	porturi rezervate	

OVER

Format:

OVER *expresie-numerică-positivă*

OVER este comandă/operator și permite suprainprimarea conform valorii *expresiei-numerică*, eventual rotunjită la numărul întreg cel mai apropiat.

1 permite suprainprimarea

0 imprimare normală

PAPER

Format:

PAPER *expresie-numerică*

PAPER este comandă/operator și permite controlul culorii fondului ecranului conform *expresiei numerice* care eventual este rotunjită la valoarea întregă cea mai apropiată.

Culori:	0 negru	5 cyan
	1 albastru	6 galben
	2 roșu	7 alb
	3 magenta	8 transparent
	4 verde	9 contrast

transparent: culoarea PAPER nu se modifică la afișări ulterioare

contrast: culoarea PAPER este negru/alb vizibil pe culoarea INK

PAUSE

Format:

PAUSE *expresie-numerică*

PAUSE este comandă și permite oprirea rulării programului pentru un timp determinat de numărul rezultat în urma evaluării *expresiei numerice* care eventual este rotunjită la numărul întreg cel mai apropiat.

număr: 0 pauză pînă la apăsarea unei taste (nu BREAK)

n timpul de așteptare este $t = n * 20\text{ms}$ (n între 1 și 65535)

PEEK

Format:

PEEK *constantă / variabilă-numerică* sau
PEEK (*expresie-numerică*)

PEEK este funcție și returnează ca valoare conținutul numeric al byte-ului de adresă dată de *expresie-numerică* care eventual este rotunjită la numărul întreg cel mai apropiat.

PI

Format:

PI

PI este funcție și returnează ca valoare numărul PI (3.1415927).

PLOT

Format:

PLOT *listă-opțiuni expresie-numerică-1, expresie-numerică-2*

unde *listă-opțiuni* este opțională și este de forma:

opțiune-afișare ; ...

opțiune-afișare este de tipul: PAPER, INK, BRIGHT, OVER,, FLASH sau INVERSE.

PLOT este comandă și desenează pe ecran un punct de coordonate (X= *expresie-numerică-1*, Y= *expresie-numerică-2*), unde *expresiile-numerică* sînt eventual rotunjite la numărul întreg cel mai apropiat.

X măsurat de la stînga la dreapta, de la 0 la 255

Y măsurat de jos în sus, de la 0 la 175

POINT

Format:

POINT (*expresie-numerică-1, expresie-numerică-2*)

POINT este funcție și returnează: 0 pentru pixel de culoare PAPER (stins)

1 pentru pixel de culoare INK (aprins)

unde pixel-ul are coordonatele: X= *expresie-numerică-1*; Y= *expresie-numerică-2* *expresiile-numerică* fiind eventual rotunjite la numărul întreg cel mai apropiat

X măsurat de la stînga la dreapta, de la 0 la 255

Y măsurat de jos în sus, de la 0 la 175

POKE

Format:

POKE *expresie-numerică-1, expresie-numerică-2*

POKE este comandă și permite încărcarea unui byte de adresă dată de *expresie-numerică-1* cu valoare dată de *expresie-numerică-2*; eventual *expresiile numerice* sînt rotunjite la numărul întreg cel mai apropiat.

Observație: bytes avînd adresele între 0 și 16383 (memoria ROM) nu pot fi modificați.

PRINT

Format:

PRINT *listă-opțiuni element-afișare separator ...*

listă-opțiuni este de forma:

opțiune-afișare ; ...

unde *opțiune-afișare* este de tipul: AT, TAB, PAPER, INK, BRIGHT, FLASH sau INVERSE; USR.

element-afișare este de forma:

expresie-numerică / și r

separator este:

; poziția următoare de afișare este în continuare

, poziția următoare de afișare este în coloana 0/16

' poziția următoare de afișare este în linia următoare

PRINT este comandă și permite afișarea pe ecran.

RANDOMIZE (RAND)

Format:

RANDOMIZE *expresie-numerică* (a) sau
RANDOMIZE USR *expresie-numerică* (b)

RANDOMIZE este comandă și execută:

- (a) introducerea *expresiei-numerică* opționale, eventual rotunjită la numărul întreg cel mai apropiat, în variabila de sistem SEED (utilizată de RND); dacă *expresia-numerică* lipsește sau este 0 în SEED se introduce variabila de sistem FRAMES (ceas de timp real) modulo 65536.
- (b) salt la rutina în cod-mașină care începe la adresa dată de *expresia-numerică*

READ

Format:

READ *variabilă-numerică / șir , ...*

READ este comandă și atribuie *variabilelor-numerică / șir* valorile din instrucțiunile DATA începând din locul unde s-a efectuat ultima citire READ anterioară (prima citire se face din prima instrucțiune DATA) dacă nu există o comandă RESTORE. Toate instrucțiunile DATA sînt considerate ca formînd un singur șir de date (în ordinea numerelor de linie).

REM

Format:

REM *text*

REM este comandă și permite introducerea comentariilor în programe. La rulare instrucțiunea REM are efect nul.

RESTORE

Format:

RESTORE *expresie-numerică-pozitivă*

RESTORE este comandă și permite forțarea citirii READ de la începutul unei instrucțiuni DATA al cărei număr de linie este dat de *expresia-numerică*, eventual rotunjită la numărul întreg cel mai apropiat. Dacă nu există un astfel de număr atunci se citește din instrucțiunea DATA, avînd numărul de linie imediat următor. *Expresie-numerică* este opțională, valoare implicită/inițială: 0.

RETURN

Format:

RETURN

RETURN este comandă și permite reîntoarcerea în programul apelant după un apel GO SUB.

RND

Format:

RND

RND este funcție și returnează ca valoare un număr pseudo-aleator calculat cu formula:

$$75 * (\text{SEED} + 1) / 65536 - \text{INT}(75 * (\text{SEED} + 1) / 65536)$$

unde SEED este variabilă-sistem.

RUN

Format:

RUN *expresie-numerică-pozitivă*

RUN este comandă și permite lansarea în execuție a programelor de la numărul de linie dat de *expresia-numerică*, eventual rotunjită la numărul întreg cel mai

apropiat. Dacă nu există un astfel de număr programul începe la numărul de linie imediat următor. Expresia-numerică este opțională, valoare implicită: 0.

SAVE

Format:

SAVE "expresie-șir" opțiuni-date

unde opțiuni-date sînt opționale și sînt de tipul CODE, DATA și SCREEN\$. SAVE este comandă și permite salvarea fișierelor pe bandă magnetică. Dacă opțiuni-date lipsesc se salvează programul BASIC împreună cu valorile variabilelor sale, dacă acestea nu au fost șterse cu CLEAR.

SCREEN\$

Format:

SCREEN\$ (linie, coloană) (a) sau
SCREEN\$ (b)

SCREEN\$ este funcție/operator care realizează:

- (a) funcție care returnează caracterul-șir, aflat în ecran în poziția (linie, coloană). linie este numărul liniei, între 0 și 21 de sus în jos coloană este numărul coloanei, între 0 și 31 de la stînga la dreapta linie și coloană pot fi constante, variabile sau expresii numerice eventual rotunjite la numărul întreg cel mai apropiat.
- (b) opțiune utilizată în comenzile LOAD, SAVE sau VERIFY pentru manipularea fișierului imagine-ecran pe bandă magnetică.

SGN

Format:

SGN constantă/variabilă-numerică sau
SGN (expresie-numerică)

SGN este funcție și returnează semnul numărului:

- 1 dacă numărul este pozitiv
- 0 dacă numărul este 0
- 1 dacă numărul este negativ

SIN

Format:

SIN constantă/variabilă-numerică
SIN (expresie-numerică)

SIN este funcție și returnează valoarea sinusului numărului considerat unghi măsurat în radiani.

SQR

Format:

SQR constantă-variabilă-numerică-pozitivă
SQR (expresie-numerică-pozitivă)

SQR este funcție și returnează ca valoare rădăcina pătrată (radicalul) numărului (pozitiv).

STEP

Format:

STEP expresie-numerică

STEP este operator și permite stabilirea pasului de incrementare a variabilei de ciclare FOR, acesta fiind numărul dat de expresia-numerică.

STOP

Format:

STOP

STOP este comandă și realizează oprirea rulării programului în locul respectiv.

STR\$

Format:

STR\$ constantă /variabilă-numerică sau
STR\$ (expresie-numerică)

STR\$ este funcție și permite transformarea numerelor în șiruri.

TAB

Format:

TAB coloană

TAB este operator și stabilește poziția de afișare în comenzile **INPUT** și **PRINT**. *coloană* este numărul coloanei, între 0 și 31, numerotarea făcându-se de la stînga la dreapta. *coloana* poate fi constantă, variabilă sau expresie numerică pentru care se efectuează operațiile de rotunjire la numărul întreg cel mai apropiat apoi modulo 32 dacă este cazul.

TAN

Format:

TAN constantă /variabilă-numerică sau
TAN (expresie-numerică)

TAN este funcție și returnează valoarea tangentei numărului considerat unghi măsurat în radiani.

THEN

Format:

THEN instrucțiune : ...

THEN este operator utilizat de **IF** care realizează secvența de comenzi care urmează, dacă condiția este adevărată.

TO

Format:

TO expresie-numerică (a)
 constantă /variabilă-șir (expresie-numerică-1 **TO** expresie-numerică-2) (b)
 (expresie-șir) (expresie-numerică-1 **TO** expresie-numerică-2) (c)

TO este operator /funcție și realizează:

- (a) operator în **FOR**, *expresie-numerică* reprezintă valoarea limită a variabilei de ciclare
- (b),(c) funcție de separare a subșirurilor. Din șirul dat de expresie se separă subșirul care începe cu caracterul cu numărul de ordine dat de *expresie-numerică-1* și se termină cu caracterul cu numărul de ordine dat de *expresie-numerică-2* eventual expresiile numerice fiind rotunjite la numerele întregi cele mai apropiate. *expresie-numerică-1/2* sînt opționale, valori implicite: primul/ultimul caracter al șirului.

USR

Format:

USR constantă /variabilă-numerică (a) sau
USR (expresie-numerică) (b) sau
USR constantă /variabilă-șir (c)

USR este operator /funcție și realizează:

- (a),(b) apelul unei subrutine în cod-mașină care începe la adresa specificată de număr. *Expresie-numerică* este eventual rotunjită la numărul întreg cel mai apropiat. Dacă se dorește reîntoarcerea în programul **BASIC** subrutina trebuie să se termine cu instrucțiunea **RET**, să nu modifice stiva și **IY** iar **HL** trebuie încărcat cu 2758h. Apelul poate fi făcut cu **RANDOMIZE** sau **PRINT**, (caz în care la retur se afișează conținutul **BC**).

(c) funcție care returnează adresa caracterului UDG respectiv

VAL

Format:

VAL *constantă / variabilă-șir*

VAL este funcție și returnează ca valoare expresie-numerică obținută prin eliminarea ghilimelelor șirului.

VAL\$

Format:

VAL\$ *variabilă-șir*

VAL\$ este funcție și returnează ca valoare expresie-șir obținută prin eliminarea ghilimelelor din variabila șir.

VERIFY

Format:

VERIFY "*expresie-șir*" *opțiuni-date*

VERIFY este comandă și permite verificarea fișierelor de pe bandă magnetică (salvate cu comanda LOAD) prin comparare cu cele aflate în memorie. *Expresie-șir* este opțională și dacă lipsește se verifică programe BASIC.

3.1.3 Structura programului BASIC

Un program BASIC are formatul:

linie-program-1

linie-program-2

...

O linie program are formatul:

număr-linie instrucțiune-1 : instrucțiune-2 : ...

unde *număr-linie* poate fi un întreg între 0 și 9999 și instrucțiune are formatul:

comandă parametri-comandă

putând fi de la 1 la 255 instrucțiuni într-o linie-program.

În mod normal la rulare liniile-program se execută în ordinea crescătoare a *numerelor-linie* și în cadrul unei linii-program instrucțiunile se execută în ordine de la stînga la dreapta, cu excepția instrucțiunilor GO SUB, GO TO, IF, LOAD, NEXT, RETURN, STOP.

Linile-program introduse fără număr-linie sînt executate imediat.

3.1.4 Mesaje de eroare

Un mesaj de eroare are formatul:

cod-eroare mesaj, număr-linie : număr-instrucțiune

unde *cod-eroare* este codul erorii întîlnite 0 ... 9, A...R; *număr-linie* este numărul liniei în care a apărut eroarea; *număr-instrucțiune* este numărul instrucțiunii din linie în care a apărut eroarea; *mesaj* explică tipul erorii întîlnite.

0 OK

Program terminat normal

1 NEXT without FOR

Variabila de control NEXT nu a fost definită de FOR

2 Variable not found

Variabilă nedefinită (lipsește LET/INPUT/DIM)

3 Subscript wrong

Indice eronat la variabila-tablou (indexată)

4 Out of memory

Depășire de memorie

5 Out of screen

Linie incorectă (23) în INPUT sau parametri incorecți în PRINT AT

6 Number too big

Încercarea de a calcula un număr mai mare decât 10^{38} generează eroare

7 RETURN without GO SUB

A fost înțilnită RETURN dar nu GO SUB anterior

8 End of file

Eroare microdrive

9 STOP statement

La execuția unei instrucțiuni STOP

A Invalid argument

Argumentul unei funcții matematice este incorect

B Integer out of range

O expresie numerică ce trebuie evaluată la intervalul (0, 255) sau (0, 65535) a generat o valoare în afara intervalului

C Nonsense in BASIC

Instrucțiune nevalidă

D BREAK-CONT repeats

A fost apăsată tasta BREAK, CONTINUE repetă instrucțiunea oprită

E Out of DATA

Instrucțiuni DATA insuficiente pentru READ sau lipsește RESTORE

F Invalid file name

Nume fișier incorect pentru comanda SAVE

G No room for line

Nu există memorie suficientă pentru a introduce linia BASIC editată

H STOP in INPUT

În decursul efectuării instrucțiunii INPUT a fost introdusă comanda STOP, CONTINUE repetă INPUT.

I FOR without NEXT

Lipsește instrucțiunea NEXT corespunzătoare

J Invalid I/O device

Eroare microdrive

K Invalid colour

Parametru incorect în PAPER, INK, BRIGHT, OVER, FLASH sau INVERSE

L BREAK into PROGRAM

A fost apăsată tasta BREAK, CONTINUE reia rularea programului de la instrucțiunea următoare

M RAMTOP no good

Valoarea specificată pentru variabila-sistem RAMTOP este incorectă (prea mare/mică)

N Statement lost

Tentativă de salt la o linie-program inexistentă

O Invalid stream

Eroare microdrive

P FN without DEF

Funcția-utilizator respectivă nu a fost definită

Q Parameter error

Parametrii funcției-utilizator nu corespund; conform listei-variabile din definiția ei

R Tape loading error

Eroare la citirea unui fișier de pe bandă magnetică, în LOAD/VERIFY

3.2 COMPILATORUL HIBASIC*

3.2.1 Instalarea

HIBASIC este cel mai bun compilator BASIC, fiind totodată și complet. Instalarea se face prin simpla încărcare a programului.

Comenzi

*C	compilează programul BASIC
*R	rulează codul compilat
*X	șterge codul compilat
*T	listează, după rularea programului cu comanda RUN, numele variabilelor și tipul lor
*ERASE	șterge programul BASIC (înlocuiește NEW)
*D	analog *C, memorează numai DATA
*E	analog *C, nu memorează DATA

3.2.2 Directive

Directivele de compilare sînt puse în program la începutul acestuia și au formatul:

număr-linie REM : directivă

unde directiva poate fi:

GO SUB număr

*HIBASIC este produs înregistrat al firmei HISOFT (Cameron Hayne)

declară subrutina care începe la linia *număr*
INT *variabilă-numerică*
 declară variabila de tip număr întreg
INT+ *variabilă-numerică*
 declară variabila de tip număr întreg pozitiv
LEN *variabilă-șir < = număr*
 declară lungimea maximă a variabilei ca fiind *număr*, valoare implicită 255
LINE *număr*
 afișează la compilare adresa liniei compilate
LIST
 afișează la compilare variabilele
OPEN# *număr-linie-1*, **CLOSE#** *număr-linie-2*
 definește prima respectiv ultima linie compilată

3.3 FIFTH*

3.3.1 Instalarea

Fifth este o extensie BASIC puternică specializată pe efecte grafice și sonore deosebite.

Încărcarea programului se face cu secvența:

CLEAR 61029 : LOAD "" CODE

Fifth nu ocupă zona de memorie UDG.

Orice program care utilizează Fifth începe cu secvența:

RANDOMIZE număr : RANDOMIZE USR 61030

unde număr este numărul de bytes necesari comenzilor OBJECT.

Tasta Break poate fi dezactivată/activată cu POKE 65230,1/0.

Toate comenzile Fifth au formatul:

număr-linie **REM** *comandă parametri \ ...*

unde \ (back-slash) este separator de comenzi (analogul : din BASIC)

Comenzile sînt introduse literă cu literă și separate de spații, nu se face distincție între majuscule și minuscule.

3.3.2 Erori

8 programul s-a terminat corect
0 salt la un număr de linie inexistent
Q comandă Fifth incorectă sau, normal, eroare BASIC

*Fifth este produs înregistrat al firmei Computer Rentals Ltd. (Richard M. Taylor)

A,B parametrii comenzii Fifth incorecți sau, normal, eroare BASIC

3.3.3 Comenzi

*

Format:

REM * text

* permite introducerea comentariilor-text în instrucțiunea REM.

ALL

Format:

REM ALL c/v-obiect

unde *c/v-obiect* este un șir (fără "") sau o variabilă-șir

ALL stabilește ca toate obiectele *nume-curent* (pentru toți indicii) să devină curente (se lucrează în continuare cu ele).

COLOUR

Format:

PRINT opțiune-afișare ; ...

REM COLOUR c/v-obiect sau

REM TEMPS\COLOUR c/v-obiect

unde *opțiune-afișare* poate fi PAPER , INK , BRIGHT , FLASH , INVERSE sau OVER.

c/v-obiect este un șir (fără "") sau o variabilă-șir

COLOUR stabilește culoarea de afișare pentru nume-obiect conform instrucțiunii PRINT vidă sau TEMPS; culoare 9 nu este utilizată.

DISABLE

Format:

REM DISABLE c/v-obiect

unde *c/v-obiect* este un șir (fără "") sau o variabilă-șir.

DISABLE oprește mișcarea obiectului curent pe ecran.

ENABLE

Format:

REM ENABLE c/v-obiect

unde *c/v-obiect* este un șir (fără "") sau o variabilă-șir.

ENABLE reactivează mișcarea obiectului curent care a fost oprită cu DISABLE.

ERASE

Format:

REM ERASE c/v-obiect

unde *c/v-obiect* este variabilă-literă sau nume-obiect

În general un caracter este deplasat (MOVE) prin ștergerea lui din poziția actuală (afișarea unui blank) apoi reafișarea caracterului în poziția deplasată. Dacă caracterul are o margine de pixeli de lățime egală sau mai mare cu numărul de pixeli de deplasare definiți în SPEED (parametrul 3) atunci obiectul nu mai este șters înainte de deplasare. Fifth detectează automat această situație, mărinde viteza de rulare a programului și eliminând efectul de clipire al caracterelor la afișare (dacă numărul lor este prea mare). Dar prin acest mod de lucru nu se detectează coliziunea obiectului respectiv cu altul. ERASE determină ștergerea caracterului din poziția actuală înainte de afișarea lui în noua poziție.

FILL

Format:

PRINT PAPER *expresie-numerică* ; **INK** *expresie-numerică***REM FILL**

sau

REM TEMPS\FILL

FILL preia culorile definite în instrucțiunea PRINT vidă sau conform TEMPS și le modifică pe tot ecranul fără a șterge conținutul lui. Operatorii PAPER și INK din PRINT sînt opționali.

FIND

Format:

REM FIND *v/ff-1* , *v/ff-2*unde *v/ff-1/2* sînt variabile-literă sau funcții Fifth.

FIND determină numele obiectului aflat la poziția dată.

v/ff-1 reprezintă coordonata x, de la 0 la 255, de la stînga la dreapta*v/ff-2* reprezintă coordonata y, de la 0 la 175, de jos în susNumele obiectului este returnat în variabila rezervată **j\$**, dacă nu există nici un obiect se returnează șirul nul "". Șirul **j\$** este format din litere minuscule.**GET**

Format:

REM GET *v/ff-1* , *v/ff-2* , *v/ff-3* , *v/ff-4* , *variabilă-șir*unde *v/ff* sînt variabile-literă sau funcții Fifth.GET salvează în *variabilă-șir* setul de caractere afișate pe ecran într-un dreptunghi definit prin coordonatele sale:*v/ff-1* coordonata x a colțului stînga-sus*v/ff-2* coordonata y a colțului stînga-sus*v/ff-3* coordonata x a colțului dreapta-jos*v/ff-4* coordonata y a colțului dreapta-jos

unde coordonatele x reprezintă liniile, între 0 și 21, de la stînga la dreapta; coordonatele y reprezintă coloanele, de la 0 la 31, de sus în jos; se salvează și atributele de culoare.

INTERACT

Format:

REM INTERACT *v/ff*unde *v/ff* este variabilă-literă sau funcție Fifth.

INTERACT determină salt la o subrutină a programului în momentul în care 2 obiecte intră în coliziune (pe ecran ocupă același loc). Subrutina trebuie să se termine cu CONTINUE (nu RETURN), *v/ff* între 0 și 65535, dacă *v/ff* > 9999 acțiunea subrutinei este nulă. Sînt permise pînă la 16 interacțiuni simultane, altfel eroare.

INTPARAM

Format:

REM INTPARAM

INTPARAM returnează, pentru 2 obiecte intrate în coliziune (pe ecran ocupă același loc) în variabilele rezervate:

h\$, **i\$** numele-obiectelor care se ciocnesc**h**, **i** indexul respectiv al obiectelor

Dacă nu există ciocnire atunci șirurile sînt vide iar indexul nul

LARGE

Format:

PRINT *opțiune-afișare* ; ...

REM LARGE sau
REM TEMPS\ LARGE

LARGE permite afișarea șirurilor de caractere mărite, la scară. Opțiunile selectate care pot fi PAPER, INK, BRIGHT, FLASH, OVER sînt stabilite în instrucțiunea PRINT vidă. TEMPS sau PRINT selectează culorile.

Variabilele rezervate utilizate de LARGE sînt:

- a\$** variabilă care conține șirul de afișat
- x** variabilă care conține poziția x al colțului stînga-sus, măsurată de la stînga la dreapta, de la 0 la 255
- y** variabilă care conține poziția y al colțului stînga-sus, măsurată de sus în jos, de la 0 la 175
- t** variabilă care conține scara pe verticală, între 1 și 22
- w** variabilă care conține scara pe orizontală, între 1 și 32

Valorile numerice sînt rotunjite la numărul întreg cel mai apropiat.

LET

Format:

REM LET *variabilă-literă = funcție-Fifth*

LET atribuie unei variabile-literă numerice valoarea funcției Fifth (în scopul prelucrării ulterioare în expresii).

LIMIT

Format:

REM LIMIT *v/ff*

unde *v/ff* este variabilă-literă sau funcție Fifth.

LIMIT determină salt la o subrutină a programului în momentul în care obiectul a atins marginea ecranului. Subrutina trebuie să se termine cu CONTINUE (nu RETURN). *v/ff* între 0 și 65535, dacă *v/ff* > 9999 acțiunea subrutinei este nulă. Sînt permise pînă la 16 obiecte care ating simultan limita, altfel eroare.

LTPARAM

Format:

REM LTPARAM

LTPARAM returnează, pentru un obiect care a atins limita ecranului:

- în **i** 0 = margine-sus
 - 1 = margine-dreapta
 - 2 = margine-jos
 - 3 = margine-stînga
 - în **h\$** *nume-obiect*
 - în **h** *indexul numelui-obiect*
- i, h\$, h** sînt variabile rezervate.

MOVE

Format:

REM MOVE *c/v-obiect, v/ff-1, v/ff-2*

unde: *c/v-obiect* este un șir (fără '"') sau o variabilă-șir
v/ff-1/2 sînt variabile literă sau funcții Fifth

MOVE deplasează obiectul curent conform VECTOR și SPEED, doar în timpul rulării programului, pînă la marginea ecranului, de la pozițiile inițiale *v/ff-1/2*. MOVE poate fi folosită și pentru a deplasa obiecte într-o buclă FOR-NEXT deoarece șterge imaginea anterioară.

v/ff-1 reprezintă coordonata x, de la 0 la 255, de la stînga la dreapta

v/ff-2 reprezintă coordonata y, de la 0 la 175, de jos în sus

OBJECT

Format:

REM OBJECT *c/v-obiect, v/ff*

unde *v/ff* este variabila-litera/functie-Fifth

c/v-obiect este un șir (fără "") sau o variabilă-șir

OBJECT definește un set de obiecte, de tip tablou unidimensional, al căror număr este dat de expresia-Fifth, care pot fi manipulate pe ecran (analog DIM). Obiectele sînt caractere alfanumerice, grafice sau UDG, cu excepția blank-ului; toate obiectele dintr-un tablou fiind de același tip. Numele reprezintă un șir alfanumeric.

Spațiul alocat obiectelor este definit de prima instrucțiune a programului; memoria ocupată de un obiect se calculează cu expresia:

*lungime-nume + 10 + 6*dimensiune-tablou*

PRINT

Format:

REM PRINT *c/v-obiect, caracter*

unde *c/v-obiect* este un șir (fără "") sau o variabilă-șir

PRINT definește obiectul ca fiind caracter. Sînt permise toate caracterele alfanumerice, grafice și UDG cu excepția blank-ului.

PUT

Format:

REM PUT *v/ff-1, v/ff-2, variabilă-șir*

unde *v/ff* sînt variabile-literă sau funcții Fifth.

PUT afișează pe ecran variabila-șir salvată cu **GET** în poziția:

v/ff-1 coordonata x a colțului stînga-sus a dreptunghiului; linia între 0 și 21, de la stînga la dreapta

v/ff-2 coordonata y a colțului stînga-sus a dreptunghiului; coloana, între 0 și 31, de sus în jos

variabilă-șir șirul de afișat, salvat anterior cu **GET**

PUT poate fi precedată de **TEMPS** sau o instrucțiune **PRINT** vidă care stabilește culorile de afișare.

Culoare: 8 **PUT** reface culorile conform salvării **GET**

9 asigură contrastul **PAPER/INK**

INVERSE 1 inversează culorile din salvarea **GET**

REPLACE

Format:

PRINT PAPER *expresie-numerică* ; **INK** *expresie-numerică*

REM REPLACE

sau

REM REPLACE\FILL

REPLACE înlocuiește culorile temporare specificate în instrucțiunea **PRINT** vidă cu culorile permanente, selectiv, în toate locurile de pe ecran unde acestea există (dacă există). **TEMPS\REPLACE** înlocuiește culorile permanente tot cu ele.

Culoare: 8 culoarea respectivă **PAPER/INK** este ignorată la căutare

9 fără utilizare

RMOVE

Format:

REM RMOVE *c/v-obiect, v/ff-1, v/ff-2*

unde *c/v-obiect* este un șir (fără "") sau o variabilă-șir

v/ff-1/2 sînt variabile-literă sau funcții Fifth

RMOVE deplasează obiectul curent conform deplasărilor relative *v/ff-1/2* față de o poziție anterioară.

$v/ff-1$ reprezintă coordonata x , de la 0 la 255, de la stînga la dreapta
 $v/ff-2$ reprezintă coordonata y , de la 0 la 175, de jos în sus
 Obiectul nu poate ieși din ecran: o coordonată relativă mai mare decît 255/175
 reprezintă o deplasare relativă în sens contrar (coordonele circulare). REMOVE
 sterge imaginea anterioară (poate fi folosită într-o buclă FOR-NEXT pentru
 deplasarea obiectelor).

SOUND

Format:

REM SOUND $v/ff-1, v/ff-2, v/ff-3, v/ff-4$

unde v/ff sînt variabile-literă sau funcții Fifth.

SOUND emite o serie de sunete, cu tonalitați (în creștere sau în descreștere).

$v/ff-1$ reprezintă numărul de sunete din serie, între 0 și 255.

$v/ff-2$ reprezintă durata, între 0 și 65535

$v/ff-3$ reprezintă tonul inițial, între 0 și 65535

$v/ff-4$ reprezintă treapta de incrementare, între 0 și 32767 pozitivă, între
 65535 și 32768 negativă.

SPEED

Format:

REM SPEED $c/v-obiect, v/ff-1, v/ff-2$

unde $v/ff-1/2$ sînt variabile-literă sau funcții Fifth

$c/v-obiect$ este un șir (fără "") sau o variabilă-șir

SPEED stabilește viteza de deplasare a obiectului curent:

$v/ff-1$ reprezintă timpul după care obiectul este deplasat: $v/ff-1 * 20ms$; între 0 și
 255

$v/ff-2$ reprezintă numărul de pixeli săriți de obiect la fiecare deplasament; între 0
 și 255

TEMPS

Format:

TEMPS

TEMPS stabilește culorile (temporare) de afișare pentru următoarele comenzi
 Fifth, conform celor stabilite anterior prin comenzile PAPER, INK.

USE

Format:

REM USE $c/v-obiect, v/ff$

unde $c/v-obiect$ este un șir (fără "") sau funcție Fifth

v/ff este o variabilă-literă sau funcție Fifth

USE stabilește obiectul curent și indicele curent (cu care se va lucra în
 continuare).

VECTOR

Format:

REM VECTOR $c/v-obiect, v/ff$

unde $c/v-obiect$ este un șir (fără "") sau o variabilă-șir

v/ff este variabilă-literă sau funcție Fifth

VECTOR stabilește direcția următoare de deplasare pentru obiectul curent
 (definit de USE sau ALL).

Există 16 direcții de deplasare, numerotate de la 0 la 15: 0 este în sus,
 următoarea direcție este decalată față de cea anterioară cu 22.50 (2PI/16 rad) în
 sens orar.

3.3.4 Funcții

Funcțiile Fifth sînt utilizate ca parametri în comenzile Fifth.

ATTR

Format:

ATTR *c/v-obiect*

unde *c/v-obiect* este un șir (fără '"') sau o variabilă-șir

ATTR returnează ca valoare codul de attribute-culoare al obiectului în formatul ATTR BASIC Spectrum.

COLUMN

Format:

COLUMN *c/v-obiect*

unde *c/v-obiect* este un șir (fără '"') sau o variabilă-șir

COLUMN returnează ca valoare numărul coloanei în care se găsește obiectul curent, definit de USE.

CURRENT

Format:

CURRENT *c/v-obiect*

unde *c/v-obiect* este un șir (fără '"') sau o variabilă-șir

CURRENT returnează indicele curent al obiectului. Sub ALL (toți indicii sînt curenți) se returnează valoarea 0.

DIRECTION

Format:

DIRECTION *c/v-obiect*

unde *c/v-obiect* este un șir (fără '"') sau o variabilă-șir

DIRECTION returnează ca valoare direcția de deplasare a obiectului (parametrul 2, VECTOR).

INTERACT

Format:

INTERACT

INTERACT returnează numărul liniei unde începe subrutina de tratare a comenzii INTERACT. Dacă *număr* > = 9999 atunci subrutina nu există.

LIMIT

Format:

LIMIT

LIMIT returnează numărul liniei unde începe subrutina de tratare a comenzii LIMIT. Dacă *număr* > = 9999 atunci subrutina nu există.

LINE

Format:

LINE *c/v-obiect*

unde *c/v-obiect* este un șir (fără '"') sau o variabilă-șir

LINE returnează ca valoare numărul liniei în care se găsește obiectul curent, definit de USE.

MASK

Format:

MASK *c/v-obiect*

unde *c/v-obiect* este un șir (fără '"') sau o variabilă-șir

MASK returnează ca valoare byte-ul de mascare pentru culoare 8 (bit 1 înseamnă culoare nemodificată).

NO

Format:

NO *c/v-obiect*

unde *c/v-obiect* este un șir (fără "") sau o variabilă-șir

NO returnează ca valoare numărul obiectelor-nume (parametrul 2, OBJECT).

SCREEN

Format:

SCREEN *c/v-obiect*

unde *c/v-obiect* este un șir (fără "") sau o variabilă-șir

SCREEN returnează ca valoare codul caracterului-obiect.

STATUS

Format:

STATUS *c/v-obiect*

unde *c/v-obiect* este un șir (fără "") sau o variabilă-șir

STATUS returnează ca valoare, pentru obiectul curent (USE):

1 dacă ENABLE

0 dacă DISABLE

VELOCITY

Format:

VELOCITY *c/v-obiect*

unde *c/v-obiect* este un șir (fără "") sau o variabilă-șir

VELOCITY returnează ca valoare numărul de pixeli săriți de obiectul curent (USE) la fiecare deplasament (MOVE) (parametrul 3, SPEED).

3.4 BETA BASIC*

3.4.1 Instalarea

Beta BASIC este o extensie BASIC puternică care introduce o serie de comenzi și funcții noi, accesibile analog cuvintelor-cheie BASIC Spectrum, precum și facilități de structurare și editare a programelor. Instalarea se face simplu, cu **LOAD ""**. Pentru dezactivarea sunetului de tastă apăsată se utilizează **POKE 23609,0**. Beta BASIC introduce o linie 0, rezidentă în toate programele, de definiție a comenzilor și funcțiilor proprii astfel încât toate programele care rulează sub Beta BASIC trebuie încărcate cu **MERGE ""**.

*Beta Basic este produs înregistrat al firmei Beta Soft

3.4.2 Comenzi

Cuvintele-cheie comenzi sînt obținute, în general, în modul cursor G (GRAPHICS) prin apăsarea unei taste literare sau cifre.

ALTER (G:A)

Format:

ALTER *listă-opțiuni-1 TO listă-opțiuni-2*

unde *lista-opțiuni* are formatul:

opțiune-afișare , ...

și *opțiune-afișare* este de tipul PAPER, INK, BRIGHT sau FLASH. ALTER modifică atributele de culoare pe tot ecranul, fără să îl șteargă:

- a) conform *listă-opțiuni-2* dacă *listă-opțiuni-1* lipsește
- b) înlocuiește atributele de culoare din *listă-opțiuni-1* cu cele din *listă-opțiuni-2*

AUTO

Format:

AUTO *număr-întreg-1 , număr-întreg-2*

AUTO introduce facilitatea de editare automată a numerelor de linie începînd de la *număr-întreg-1* cu pasul *număr-întreg-2*.

Numerele-întregi-1/2 sînt opționale, valori implicite 10, 10.

Dacă în AUTO se modifică numărul de linie, acesta este preluat în continuare de AUTO. Ieșirea din AUTO se face cu tasta BREAK apăsată minim 1s.

BREAK-*(SS și blank)

Apăsarea tastelor SS și blank (spațiu) minim 1s determină oprirea oricărui program care rulează sub Beta BASIC și afișarea cursorului *. Pentru aceasta trebuie ca programul să nu dezactiveze întreruperile mascabile.

CLOCK (G:C)

Format:

CLOCK *expresie-numerică* (a)sau

CLOCK *expresie-șir* (b)

CLOCK efectuează servicii de ceas real: afișează timpul curent, alarma sonoră la un moment determinat, apelul unei subrutine la un moment determinat.

(a) dacă numărul este între 0 și 7 acesta definește modul de lucru:

număr	subrutină	alarmă	afișare
0	nu	nu	nu
1	nu	nu	da
2	nu	da	nu
3	nu	da	da
4	da	nu	nu
5	da	nu	da
6	da	da	nu
7	da	da	da

dacă numărul este între 8 și 9999 acesta reprezintă numărul de linie unde începe subrutina de apelat în momentul în care timpul curent coincide cu timpul-alarmă

(b) poziționează ceasul:

cifra-1 cifra-2 (ora) : cifra-3 cifra-4 (minutul) : cifra-5 cifra-6 (secunda) conform primelor 6 cifre din șir, alte caractere fiind ignorate, cu excepția "a", "A"; cifrele lipsă se consideră 0 ; "a", "A" la începutul șirului

DPOKE plasează în memorie la adresa dată de *expresie-numerică-1* byte-ul inferior al *expresiei-numerică-2* și la adresa imediat următoare (+1) byte-ul superior.

EDIT (Enter apoi 0)

Format:

EDIT *expresie-numerică*

EDIT lansează direct în editare linia a cărei număr este dat de *expresia-numerică* indiferent de poziția cursorului în listing. *expresie-numerică* este opțională, valoarea implicită este numărul de linie indicat de cursor în listing.

ELSE(G:E)

Format:

ELSE *instrucțiune-1* : ...

ELSE definește instrucțiunile care se execută în IF dacă condiția este falsă, ELSE trebuie plasată în aceeași linie cu IF, THEN și este precedată de : . ELSE este opțională și în cazul condiționalelor IF imbricate aparține celei mai apropiate IF anterioare.

END PROC (G:3)

Format:

END PROC

END PROC marchează sfârșitul definiției unei proceduri (DEF PROC) și reîntorcerea la programul apelant. O definiție de procedură poate avea mai multe puncte de ieșire (retur) dar în acest caz ea trebuie separată de restul programului deoarece nu se mai asigură neexecutarea ei.

EXIT IF (G:I)

Format:

EXIT IF *condiție*

EXIT IF este utilizată în interiorul unei bucle DO-LOOP și determină saltul la prima instrucțiune după LOOP (părăsirea buclei) dacă *condiție* este adevărată.

FILL(G:F)

Format:

FILL *expresie-numerică-1* , *expresie-numerică-2*

FILL INK *culoare* ; *expresie-numerică-1* ; *expresie-numerică-2* (a)sau

FILL PAPER *culoare* ; *expresie-numerică-1* ; *expresie-numerică-2* (b)sau (c)

unde *culoare* este o expresie-numerică care reprezintă o culoare validă și *expresie-numerică-1/2* reprezintă coordonata punctului x/y de unde începe operația, punct oarecare în zonă:

x între 0 și 255 de la stînga la dreapta

y între 0 și 175 de jos în sus

FILL umple o zonă ecran delimitată de o curbă închisă sau marginile sale astfel:

(a) pixel culoare PAPER devine pixel culoare INK permanentă

(b) pixel culoare PAPER devine pixel culoare INK temporară

(c) pixel culoare INK devine pixel culoare PAPER

GET (G:G)

Format:

GET *variabilă-numerică* / *șir*

GET așteaptă apăsarea unei taste și atribuie variabilei:

numerică: valoarea cifrei apăsate (0 ... 9) sau un cod numeric pentru celelalte taste (A= 10 ... a=42...) conform modului cursor (L, C, E, G)

șir: șirul cu un unic element caracterul-tastă-apăsată conform modului cursor (L, C, E, G)

JOIN (G:SS și 6)

Format:

JOIN *expresie-numerică*JOIN concatenează la sfârșitul liniei BASIC al cărei număr este dat de *expresie numerică* conținutul liniei BASIC următoare**KEY IN (G:SS și 4)**

Format:

KEY IN *variabilă-șir*KEY IN introduce o linie BASIC, în program, în timpul rulării acestuia, care este dată de *variabilă-șir*. KEY IN nu poate fi utilizată decât în cadrul unui program.**KEYWORDS (G:8)**

Format:

KEYWORDS *expresie-numerică*

KEYWORDS comută afișarea caractere grafice/cuvinte-cheie Beta BASIC conform numărului:

0 caractere grafice , cu excepția KEYWORDS

1 cuvinte-cheie Beta BASIC

KEYWORDS 0 afectează listingul programului dar nu rularea (funcționalitatea) acestuia.

LIST

Format:

LIST *expresie-numerică-1 TO expresie-numerică-2*LIST efectuează comanda BASIC Spectrum normală și, conform formatului, listarea liniilor avînd numerele între *expresie-numerică-1* și *expresie-numerică-2*. *Expresie-numerică-1/2* sînt opționale, valori implicite prima/ultima linie de program.**LLIST**

Format:

LLIST *expresie-numerică-1 TO expresie-numerică-2*LLIST efectuează comanda BASIC Spectrum normală și, conform formatului, listarea liniilor avînd numerele între *expresie-numerică-1* și *expresie-numerică-2*. *Expresie-numerică-1/2* sînt opționale, valori implicite prima/ultima linie de program.**LOOP (G:L)**

Format:

LOOP (a) sau**LOOP WHILE** *condiție* (b) sau**LOOP UNTIL** *condiție* (c)

LOOP marchează sfârșitul unei bucle fără contor:

(a) utilizat de DO/DO LOOP/DO UNTIL

WHILE: dacă *condiție* este adevărată se efectuează salt la prima instrucțiune de după DO, altfel continuăUNTIL: dacă *condiție* este falsă se efectuează salt la prima instrucțiune de după DO, altfel continuă(b) utilizat de DO, dacă *condiție* este adevărată se efectuează salt la prima instrucțiune de după DO , altfel continuă(c) utilizat de DO, dacă *condiție* este falsă se efectuează salt la prima instrucțiune de după DO, altfel continuă

Buclele DO-LOOP WHILE/UNTIL sînt parcurse cel puțin odată.

ON (G:O)

Format:

GO TO ON *expresie-numerică ; număr-linie-1 , ...* sau
GO SUB ON *expresie-numerică ; număr-linie-1 , ...*

unde *numerele-linie* sînt expresii numerice

ON determină, conform comenzii, salt la linia a cărei număr are ordinul dat de *expresia-numerică*. Dacă numărul nu se încadrează în intervalul 1, ..., *numarul-de-elemente-în-listă*, comanda este ignorată.

ON ERROR(G:N)

Format:

ON ERROR *expresie-numerică*

ON ERROR permite mascarea erorilor prin apelarea unei subrutine (tip GO SUB), care începe la numărul de linie dat de *expresie-numerică*, la întîlnirea unei erori, cu excepția erorilor 0 și 9. Comanda este dezactivată prin **ON ERROR 0** și în timpul rulării subrutinei proprii (de apel). ON ERROR utilizează 3 variabile-numerice rezervate:

LINE numărul liniei în care a intervenit eroarea
STAT numărul instrucțiunii din linie în care a intervenit eroarea
ERROR codul erorii întîlnite
PLOT

Format:

PLOT *listă-opțiuni ; expresie-numerică-1 ,
 expresie-numerică-2 , expresie-numerică-3*

PLOT efectuează comanda BASIC Spectrum normală și, conform formatului, afișarea unui șir într-o poziție dată. *Expresie-numerică-1/ 2* definesc coordonatele x/y ale colțului dreapta-sus al primului caracter al șirului:

x între 0 și 255, de la stînga la dreapta

y între 0 și 175, de jos în sus

Fragmentul de șir care depășește marginea dreaptă a ecranului este afișat în stînga (coordonate circulare).

POKE

Format:

POKE *expresie-numerică , expresie-șir*

POKE efectuează comanda BASIC Spectrum normală și, conform formatului, stocarea unui șir în memorie, începînd de la adresa dată (în formatul codurilor ASCII).

POP (G:C)

Format:

POP *variabilă-numerică*

POP retrage un număr de linie (adresa de retur) din stiva GO SUB/PROC/DO-LOOP. *Variabila-numerică* este opțională și dacă este dată capătă valoarea numărului-de-linie. POP permite leșirea din subrutine, proceduri, bucle fără încărcarea stivei respective.

PROC (G:2)

Format:

PROC *expresie-șir*

PROC apelează procedura definită cu DEF PROC respectiv.

RENUM (G:4)

Format:

RENUM (*număr-linie-1 TO număr-linie-2*)
LINE *număr-linie-3 STEP expresie-numerică*

RENUM renumerează un bloc de linii-program care începe la *număr-linie-1* și se termină la *număr-linie-2*. Prima linie a blocului primește *număr-linie-3*, următoarele primind numere de linie conform pasului dat de *expresie-numerică*.

Număr-linie-1/2 sînt opționale, valori implicite 10, 10; dacă lipsesc expresia dintre paranteze poate lipsi.

LINE *număr-linie-3*, **STEP** *expresie-numerică* sînt opționale, valori implicite: 10, 10.

RENUM acordează toate instrucțiunile care folosesc numere de linie, cu excepția CLOCK, în conformitate cu noua numerotare, dacă numerele de linie sînt date explicit; variabilele/expresiile-nerice generează eroare Y și vechea linie nu este ștersă.

ROLL (G:R)

Format:

ROLL *expresie-numerică-cod* , *expresie-numerică-pixel* ;
expresie-numerică-x ; *expresie-numerică-y* ;
expresie-numerică-ferastră-x ; *expresie-numerică-ferastră-y*

ROLL efectuează deplasarea unei ferestre-ecran, ceea ce iese din ecran printr-o margine intră prin marginea opusă; conform codului:

cod	direcție	deplasează
1	stînga	atribute
2	jos	atribute
3	sus	atribute
4	dreapta	atribute
5	stînga	pixeli
6	jos	pixeli
7	sus	pixeli
8	dreapta	pixeli
9	stînga	pixeli și atribute
10	jos	pixeli și atribute
11	sus	pixeli și atribute
12	dreapta	pixeli și atribute

Expresie-numerică-pixel este opțională, valoare implicită 1; definește numărul de pixeli cu care se deplasează fereastra.

Expresie-numerică-x, ..., *expresie-numerică-ferastră-y* sînt opționale și definesc fereastra de deplasat, implicit întregul ecran.

x,y coordonatele *x* , *y* ale colțului stînga-sus al ferestrei

x de la 0 la 255, de la stînga la dreapta

y de la 0 la 175, de jos în sus

ferastră-x lungimea *x* a ferestrei, în coloane
 între 1 și 32, de la stînga la dreapta

ferastră-y înălțimea *y* a ferestrei în pixeli
 între 0 și 176, de sus în jos

Atributele nu pot fi deplasate decît în pași multiplu de 8 pixeli, indiferent de numărul de pixeli-deplasament.

SCROLL (G:S)

Format:

SCROLL *expresie-numerică-cod* , *expresie-numerică-pixel* ;
expresie-numerică-x ; *expresie-numerică-y* ;
expresie-numerică-ferastră-x ; *expresie-numerică-ferastră-y*

SCROLL efectuează deplasarea unei ferestre-ecran, ceea ce iese din ecran printr-o margine se pierde; conform codului:

cod	direcție	deplasează
1	stînga	atribute
2	jos	atribute
3	sus	atribute
4	dreapta	atribute
5	stînga	pixeli
6	jos	pixeli
7	sus	pixeli
8	dreapta	pixeli
9	stînga	pixeli și atribute
10	jos	pixeli și atribute
11	sus	pixeli și atribute
12	dreapta	pixeli și atribute

Expresie-numerică-pixel este opțională, valoare implicită 1; definește numărul de pixeli cu care se deplasează fereastra.

Expresie-numerică-x, ..., expresie-numerică-fereastră-y sînt opționale și definesc fereastra de deplasat, implicit întregul ecran.

x, y coordonatele x, y ale colțului stînga-sus al ferestrei

x de la 0 la 255, de la stînga la dreapta

y de la 0 la 175, de jos în sus

fereastră-x lungimea x a ferestrei, în coloane între 1 și 32, de la stînga la dreapta

fereastră-y înalțimea y a ferestrei în pixeli între 0 și 176, de sus în jos

Atributele nu pot fi deplasate decît în pași multiplu de 8 pixeli, indiferent de numărul de pixeli-deplasament.

SORT (G:M)

Format:

SORT *v-n/s opțiune-indice opțiune-indice* (a)sau

SORT INVERSE *v-n/s opțiune-indice opțiune-indice* (b)

unde *v-n/s* este o variabilă-numerică/șir

și *opțiune-indice* sînt opționale, de forma:

(*indice-1 TO indice-2*)

(1)sau

(*indice*)

(2)sau

()

(3)

SORT sortează variabile-tablou, uni sau bi-dimensionale, în ordine directă (a) sau inversă (b) (ordinea este: pentru numere: numerală; pentru șiruri: codul ASCII):

(1) sortează între *indice-1* și *indice-2*; *indicii-1/2* sînt opționali, valori implicite primul /ultimul element

(2) sortează partea de tablou definită de *indice*

(3) *indicii* (pe poziția respectivă) sînt indiferenți

SPLIT (SS și W)

Introducerea caracterului <> într-o linie BASIC imediat după : determină separarea liniei în 2 părți: prima este introdusă în listing iar a 2-a în zona editare, amîndouă cu numărul de linie respectiv.: trebuie sa nu fie precedat de o instrucțiune REM.

TRACE (G:T)

Format:

TRACE *expresie-numerică*

TRACE permite depanarea programelor, apelînd o subrutină înaintea execuției fiecărei instrucțiuni BASIC. Numărul definește:

0 dezactivează TRACE, rulare normală
 1...9999 activează TRACE, numărul reprezintă numărul de linie de unde începe subrutina de apelat

TRACE folosește 2 variabile rezervate:

LINE numărul liniei curente

STAT numărul instrucțiunii curente din linia curentă

UNTIL (G:K)

Format:

UNTIL *condiție*

Utilizat în DO/LOOP determină execuția ciclului atâta timp cât condiția este falsă.

USING (G:U)

Format:

USING *format ; constantă /variabilă-numerică*

unde format este #...#.#... sau 0...0.0...

USING este utilizat în PRINT pentru a stabili formatul de afișare al numerelor: numărul de cifre înainte și după . .

suprimă 0-urile (redundante) din fața numărului

0 afișează 0-urile (redundante) din fața și coada numărului

% semnalizează depășire, cifrele inferioare (mai puțin semnificative) sînt suprimate conform formatului, ultima cifră afișată este eventual rotunjită.

WHILE (G:J)

Format:

WHILE *condiție*

Utilizat în DO/LOOP determină execuția ciclului atâta timp cât condiția este falsă.

XOS, YOS, XRG, YRG

Variabile rezervate pentru comenzile PLOT, DRAW, CIRCLE, FILL:

XOS originea x a sistemului de coordonate, inițial 0 (stînga jos)

YOS originea y a sistemului de coordonate, inițial 0 (stînga jos)

XRG scala axei x, inițial 256

YRG scala axei y, inițial 176

3.4.3 Funcții

Funcțiile se obțin prin tastarea FN *literă simbol* unde *simbol* este (pentru funcții-nerice sau \$ pentru funcțiile-șir.

În momentul tastării simbolului FN *litera* se șterge și este înlocuită cu denumirea funcțiilor. Funcțiile sînt definite într-o linie 0 introdusă de Beta BASIC în toate programele.

AND (FN A)

Format:

AND (*expresie-numerică-1 , expresie-numerică-2*)

AND efectuează operația logică și pe biți între *expresie-numerică-1* și *expresie-numerică-2* care pot fi între 0 și 65535.

BIN\$ (FN B)

Format:

BIN\$ (*expresie-numerică*)

Returnează un șir format din 0 și 1, cu 8 sau 16 caractere, care reprezintă forma binară a *expresiei-numerice* evaluată la un byte (8 biți) sau 2 bytes (16 biți).

CHAR\$ (FN C\$)

Format:

CHAR\$ (*expresie-numerică*)

CHAR\$ transformă *expresia-numerică*, între 0 și 65535, într-un șir de 2 caractere conform codurilor ASCII. Primul caracter are codul: INT (*expresie-numerică / 256*), al 2-lea caracter are codul *expresie-numerică—256*INT(expresie-numerică / 256)*.

COSE (FN C())

Format:

COSE (*expresie-numerică*)

COSE returnează cosinusul unghiului exprimat în radiani cu o aproximație de numai 4 zecimale exacte dar aproximativ de 6 ori mai rapid decât COS.

DEC (FN D())

Format:

DEC (*expresie-șir*)

DEC convertește într-un număr zecimal șirul citit ca un număr hexazecimal cu minim 1 și maxim 4 cifre.

DPEEK (FN P())

Format:

DPEEK (*expresie-numerică*)

DPEEK citește un număr pe 2 bytes din memorie de la adresa dată de *expresie-numerică*, în convenția Intel, astfel încât valoarea este PEEK (*expresie-numerică*) + 256 * PEEK (*expresie-numerică + 1*).

FILLED (FN F())

Format:

FILLED ()

FILLED returnează numărul de pixeli manipulați de ultima comandă FILL.

HEX (FN H\$)

Format:

HEX\$ (*expresie-numerică*)

HEX\$ convertește un număr zecimal, între -255 și 65535, în echivalentul său hexazecimal (pentru numere negative în complement față de 2).

INSTRING (FN I())

Format:

INSTRING (*expresie-numerică* , *expresie-șir-1* , *expresie-șir-2*)

INSTRING caută în șirul dat de *expresie-șir-1* subșirul dat de *expresie-șir-2*, începînd cu caracterul al cărui număr de ordine este dat de *expresie-numerică* și returnează poziția primului caracter al subșirului sau 0 dacă acesta nu a fost găsit. # poate înlocui în subșir orice caracter, cu excepția primului.

MEM (FN M())

Format:

MEM ()

MEM returnează numărul de bytes liberi din memorie, realizînd aceeași funcție ca PRINT 65535—USR 7962 (care returnează cu 8 bytes mai mult).

MEMORY\$ (FN M\$)

Format:

MEMORY\$()

MEMORY\$ returnează conținutul memoriei, mai puțin prima și ultimele 3 locații, ca un șir, astfel încît are lungimea 65532.

MOD (FN V)

Format:

MOD (*expresie-numerică-1*, *expresie-numerică-2*)

MOD returnează restul împărțirii (întregi) *expresie-numerică-1* la *expresie-numerică-2*.

NUMBER (FN N)

Format:

NUMBER (*expresie-șir*)

NUMBER transformă un șir de 2 caractere într-un număr, între 0 și 65535, conform codurilor ASCII:

$num\bar{a}r = 256 * CODE\ \bar{s}ir\ (1) + CODE\ \bar{s}ir\ (2)$

OR (FN O)

Format:

OR (*expresie-numerică-1*, *expresie-numerică-2*)

OR efectuează operația logică sau între *expresie-numerică-1* și *expresie-numerică-2* care pot fi între 0 și 65535.

RNDM (FN R)

Format:

RNDM (*expresie-numerică*)

RNDM furnizează un număr aleator întreg între 0 și *expresie-numerică* eventual rotunjită la numărul întreg cel mai apropiat. Dacă *expresie-numerică* = 0 atunci se generează numere aleatoare între 0 și 1. RNDM este de 2.5 ori mai rapid decît RND.

SCRN\$ (FN K\$)

Format:

SCRN\$ (*linie*, *coloană*)

SCRN\$ returnează caracterul-șir aflat în ecran în poziția (*linie*, *coloană*). *linie* este numărul liniei între 0 și 21, de sus în jos; *coloană* este numărul coloanei între 0 și 31, de la stînga la dreapta. *linie* și *coloană* pot constante. variabile sau expresii numerice eventual rotunjite la numărul întreg cel mai apropiat. SCRN\$ recunoaște UDG.

SINE (FN S)

Format:

SINE (*expresie-numerică*)

SINE returnează sinusul unghiului exprimat în radiani cu o aproximație de numai 4 zecimale exacte, dar aproximativ de 6 ori mai rapid decît SIN.

STRING\$ (FN S\$)

Format:

STRING\$ (*expresie-numerică*, *expresie-șir*)

STRING\$ returnează un șir format prin repetarea *expresiei-șir* de un număr de ori dat de *expresie-numerică*.

TIME\$

Format:

TIME\$()

TIME\$ returnează ca valoare șirul care reprezintă ora curentă în format CLOCK.

USING\$ (FN U\$)

Format:

USING\$ (format , expresie-numerică)

unde format este #...#.#...# sau 0...0.0...0

USING\$ generează un șir care reprezintă un număr, conform formatului care stabilește numărul de cifre înainte și după . .

suprimă 0-urile (redundante) din fața numărului

0 afișează 0-urile (redundante) din fața și coada numărului

XOR (FN X())

Format:

XOR (expresie-numerică-1 , expresie-numerică-2)

XOR efectuează operația logică sau-exclusiv între expresie-numerică-1 și expresie-numerică-2 care pot fi între 0 și 65535.

3.4.4 Mesaje de eroare

Pe lângă mesajele de eroare BASIC Spectrum, Beta BASIC introduce o serie de mesaje noi, în același format, conform comenzilor și funcțiilor noi introduse.

S Missing LOOP

Lipsește instrucțiunea LOOP

T LOOP without DO

Lipsește instrucțiunea DO

U No such line

Delete utilizat cu un număr de linie inexistent

V No POP data

Stiva GO SUB/DO-LOOP/PROG este vidă

W Missing DEF PROC

Lipsește instrucțiunea DEF PROC

X No END PROC

Lipsește instrucțiunea END PROC

Y Too hard

La execuția comenzii RENUM a fost găsită o instrucțiune de salt la un număr de linie generat printr-o expresie numerică care nu poate fi evaluată.

3.4.5 Setul de caractere Beta BASIC

Cod	Comandă (G:UDG)	Cod	Comandă (G:UDG)
128	KEYWORDS	147	DO
129	DEF PROC	148	ELSE
130	PROC	149	FILL
131	END PROC	150	GET
132	RENUM	151	neutilizat
133	— —	152	EXIT IF
134	AUTO	153	WHILE

135	DELETE	154	UNTIL
136	— —	155	LOOP
137	JOIN	156	SORT
138	EDIT	157	ON ERROR
139	KEYIN	158	ON
140	— —	159	DPOKE
141	— —	160	POP
142	DEFKEY	161	ROLL
143	— —	162	SCROLL
144	ALTER	163	TRACE
145	*(BREAK-*)	164	USING
146	CLOCK		

4.1 INSTALAREA*

Alături de Forth, Pascal este cel mai eficient limbaj de programare pe calculatoarele compatibile Spectrum, fiind însă mai ușor de utilizat. După încărcarea cu **LOAD** "" compilatorul cere **Top of RAM?** adresa maximă RAM, **Top of RAM for T ?** adresa stivei pentru comanda editor T și **Table size?** dimensiunea tabelului de simboluri utilizată de compilator.

Prin tastarea Enter ca răspuns la întrebări se consideră valori implicite: RAMTOP, -, memoria RAM liberă/16.

Din BASIC Spectrum în editorul compilatorului se întră prin:

RANDOMIZE USR 24598	șterge textul-sursă
RANDOMIZE USR 24603	păstrează textul-sursă
RANDOMIZE USR 24608	inițializare

Taste funcționale:

E	Enter:	Carriage return
CS1	Caps shift & 1:	ieșire din comanda-editor
CS0	Delete = Caps shift & 0:	șterge un caracter
CS8	Cursor = Caps shift & 8:	avans tab
CS5	Cursor = Caps shift & 5:	șterge toată linia

4.2 EDITORUL

Promptul așteaptă o comandă de tipul:

comandă număr-1 , număr-2 , șir-1 , șir-2

unde *număr-1*, *număr-2* sînt numere întregi între 1 și 32767; *șir-1*, *șir-2* sînt șiruri de caractere de lungime între 0 și 20. Blank-ul nu este semnificativ decît în șiruri; , este separator

4.2.1 Generarea textului-sursă

I *număr-1 , număr-2*

Introduce numărătoarea automată a liniilor.

Valori implicite: *număr-1* = 10, *număr-2* = 10.

O linie text-sursă este de forma:

număr-linie secvență-program

Lungimea *secvenței-program* este de cel mult 80 de caractere, numărul de linie servește doar pentru facilitarea editării.

Orice linie nou introdusă, cu comanda **I** sau direct după prompt-ul **>**, se suprapune peste textul anterior astfel încît liniile anterioare cu același număr de linie se șterg.

4.2.2 Listare

L *număr-1 , număr-2*

Listează liniile textului-sursă avînd numerele între *număr-1* și *număr-2*; orice tastă efectuează scroll, cu excepția CS1.

Valori implicite: *număr-1* = 1, *număr-2* = 32767

K *număr*

Setează numărul de linii de listat înainte de scroll.

Valori implicite: *număr* = 16

4.2.3 Editare

D *număr-1 , număr-2*

Șterge liniile textului-sursă, avînd numerele între *număr-1* și *număr-2*.

Valori implicite: fără

Dacă *număr-1* > *număr-2* sau lipsesc argumente comanda este ignorată.

M *număr-1* , *număr-2*

Introduce linia *număr-1* la linia *număr-2*; linia *număr-1* rămîne în text; linia *număr-2* anterioară se șterge.

Valori implicite: fără

N *număr-1* , *număr-2*

Renumerotează textul-sursă începînd cu *număr-1* și cu pasul *număr-2*

Valori implicite: fără

F *număr-1* , *număr-2* , *șir-1* , *șir-2*

șir-1 este căutat în liniile cu numerele între *număr-1* și *număr-2*; dacă este găsit atunci se intră în editorul de linie (comanda E) cu cursorul pe primul caracter al *șir-1*; *șir-2* substituie *șir-1*.

Valori implicite: cele aflate în buffer la o utilizare anterioară, altfel fără

E *număr*

Editor de linie.

Valori implicite: fără

Dacă număr lipsește, comanda este ignorată.

Sub-comenzi:

blank cursor dreapta

CS0 cursor stînga

CS8 avans tab

E terminarea editării, cu modificări

C supra-scriere, începînd de la poziția curentă a cursorului

F caută următorul *șir-1* din comanda F

I inserează caractere la poziția curentă a cursorului

K șterge un caracter la poziția cursorului

L listează restul liniei în curs de editare

Q terminarea editării, modificările sînt anulate

R anulează modificările

S înlocuiește *șir-1* din comanda F

X avansează cursorul la sfîrșitul liniei și introduce sub-comanda I

Z șterge toate caracterele de la cursor la sfîrșitul liniei

4.2.4 Comenzi de bandă magnetică

P *număr-1* , *număr-2* , *șir*

Salvează pe bandă magnetică liniile din textul-sursă între *număr-1* și *număr-2*, într-un fișier cu numele *șir*.

Valori implicite: cele aflate în buffer la o utilizare anterioară, altfel fără.

G , , *șir*

Încarcă de pe bandă magnetică fișierul (text-sursă) *șir* la sfîrșitul textului-sursă și liniile sînt renumerotate.

Valori implicite: dacă *șir* lipsește se încarcă primul fișier

T *număr*

Compilează textul-sursă începînd cu linia *număr* după care întreabă **OK?** și la răspunsul **Y** codul-obiect direct executabil este salvat pe bandă magnetică cu nume (implicit) definit de comanda F. Codul poate fi apoi încărcat și rulat din BASIC Spectrum cu **LOAD "" CODE : RANDOMIZEUSR 24608**

W *număr-1* , *număr-2* , *șir*

Salvează liniile, având numărul între *număr-1* și *număr-2*, cu numele *șir*, în format utilizabil de comanda de compilare \$F.

4.2.5 Comenzi de uz general

B

Retur în BASIC Spectrum, reîntoarcere în Pascal cu:

RANDOMIZE USR 24598	șterge textul-sursă
RANDOMIZE USR 24603	păstrează textul-sursă
RANDOMIZE USR 24608	inițializare

S, , d

Schimbă separatorul operanzilor din , în d; d nu poate fi blank.

V

Afișează bufferul.

4.2.6 Compilarea și rularea programelor

C număr

Compilează programul începînd de la linia *număr*.
valoare implicită: *număr* = 1

La întîlnirea unei erori:

E introduce în editare linia curentă

P introduce în editare linia precedentă

O linie compilată are structura afișată:

adresă-linie-compilată *linie-compilată*

R

Rulează codul compilat fără erori de compilare.

4.3 STRUCTURI DE LIMBAJ

Setul de caractere

cifre: 0, ..., 9; litere: a ... z, A ... Z

caractere speciale: + - * / = ^ < > () [] . , ; ; £

simboluri speciale:

operator de atribuire :=

operatori relaționali <=, >=, <>

delimitatori de comentarii {, } sau (*, *)

Separatori

Spațiul, sfârșitul de linie și comentariul sînt separatori. Se pot utiliza oricîți separatori și oriunde cu excepția utilizării lor în interiorul identificatorilor, numerelor și simbolurilor speciale. Acolo unde apar, caracterele speciale sînt de asemenea separatori, ; este separator de instrucțiuni care în cazuri speciale poate lipsi.

Cuvinte-cheie rezervate, identificatori predefiniți

ABS	FALSE	ODD	SIN
ADDR	FOR	OF	SIZE
AND	FORWARD	ORD	SQR
ARCTAN	FRAC	OUT	SQRT
ARRAY	FUNCTION	PAGE	SUCC
BEGIN	GOTO	PACKED	TAN
BOOLEAN	HALT	PEEK	THEN
CASE	IF	POKE	TIN
CHAR	INCH	PRED	TO
CHR	INLINE	PROCEDURE	TOUT
CONST	INP	PROGRAM	TRUE
COS	INTEGER	RANDOM	TRUNC
DIV	LABEL	READ	TYPE
DO	LN	READLN	UNTIL
DOWNTO	MARK	REAL	USER
ELSE	MAXINT	RECORD	VAR
END	MOD	RELEASE	WHILE
ENTIER	NEW	REPEAT	WIDTH
EOLN	NIL	ROUND	WRITE
EXP	NOT	SET	WRITELN

Identificatori (notație: id)

literă literă / cifră ...

Numai primele 10 caractere sînt semnificative. Se face distincție între litere majuscule și minuscule. Cuvintele-cheie trebuie scrise în majuscule, în editare sînt introduse literă cu literă dar în memorie sînt reținute pe un singur byte fiecare.

Întreg fără semn (notație: ifs)

cifră ...

Număr fără semn (notație: nfs)

ifs

ifs . ifs

ifs exp

ifs . ifs exp

£ cifră-hex

unde exp are formatul:

E ifs

Constantă fără semn (notație: cfs)

id-const

nfs

' caracter '

NIL

MAXINT
TRUE
FALSE

unde NIL este constantă de inițializare a pointerilor cu "nici o adresă"
MAXINT = 32767 cel mai mare întreg
TRUE, FALSE constante logice (adevărat, fals)
constantele acestea fiind predefinite.

Constantă (notație: const)

id-const
nfs
' *caracter* '
CHR (*const*)

Tip simplu (notație: tip-s)

INTEGER
REAL
CHAR
BOOLEAN
(*id*, ...)
const ..const

unde INTEGER tip întreg predefinit
REAL tip real predefinit
CHAR tip caracter predefinit
BOOLEAN tip logic predefinit

Un scalar enumerat poate avea maxim 256 elemente.

În tipul *const .. const* constantele trebuie să fie dintr-o mulțime de tip scalar enumerat ante/pre-definită.

Tip (notație: tip)

tip-s (a)
^*id-tip* (b)
ARRAY [*tip-s*, ...] OF *tip* (c)
SET OF *tip-s* (d)
RECORD *id*, ... : *tip*; ... END (e)

unde: (b) utilizat pentru variabile dinamice
(c) definește tablouri de variabile *tip* cu indici (pentru acces) *tip-s*
(d) submulțimi
(e) variabile (tipuri) cu structuri complexe (cîmpuri)

PACKED poate fi folosit în fața tipurilor ARRAY, SET și RECORD dar acțiunea lui este nulă.

Variabile (notație: var)

id-var (a)
id-var [*expresie*, ...] (b)
id-var . *id-cîmp* (c)
id-cîmp (d)
id-cîmp [*expresie*, ...] (e)
id-cîmp . *id-cîmp* ... (f)

unde: (a) definiție/acces la variabilele statice
(b) acces la elementele variabilelor tablou;
la acces/definirea tipului se pot utiliza în mod independent
structurile [*expresie* ...] sau [*expresie*] [...]
(c) acces la cîmpurile variabilelor RECORD

- (d) acces la cîmpurile variabilelor RECORD; în structuri WITH
- (e) acces la cîmpurile tablou ale variabilelor RECORD
- (f) acces la cîmpurile imbricate; în structuri WITH

Variabilele/cîmpurile dinamice sînt postfixate cu . .

Factor (notație: fact)

<i>cfs</i>	(a)
<i>var</i>	(b)
<i>id-funcție</i>	(c)
<i>id-funcție (expresie , ...)</i>	(d)
<i>(expresie)</i>	(e)
NOT factor	(f)
[expresie]	(g)
[expresie .. expresie , ...]	(h)

- unde:
- (c),(d) reprezintă apeluri de funcții cu sau fără parametri
 - (e) utilizarea parantezelor pentru inversarea ordinii de prioritate a operatorilor
 - (f) operație de negare logică sau aritmetică
 - (g),(h) mulțimi și (sub)domenii

Termen (notație: termen)

factor
factor operator factor

unde operator este:

*	înmulțire pe numere întregi, reale; intersecție pe mulțimi
/	împărțire pe numere reale
DIV	împărțire pe numere întregi
MOD	clase de resturi modulo
AND	și logic /aritmetic

Expresie-simplă (notație: expr-s)

termen
termen operator termen ...
semn termen
semn termen operator termen ...

unde:

semn este + sau -

operator este:

+	adunare pe numere întregi, reale; reuniune pe mulțimi
-	scădere pe numere întregi, reale; diferență pe mulțimi
OR	sau logic /aritmetic

Expresie (notație: expresie)

expr-s
expr-s operator expr-s ...

unde operator este:

=	egalitate pe toate tipurile
< >	inegalitate pe toate tipurile
<	mai-mic pe tipuri simple, șiruri de caractere
>	mai-mare pe tipuri simple, șiruri de caractere
< =	mai-mic-egal pe tipuri simple, șiruri de caractere
	incluziune pe mulțimi
> =	mai-mare-egal pe tipuri simple, șiruri de caractere
	incluziune pe mulțimi
IN	apartenență pe mulțimi

Declarații (notație: decl)

LABEL *ifs* , ... ; (a)

CONST *id = const* ; ... ; (b)

TYPE *id = tip* ; ... ; (c)

VAR *id* , ... : *tip* ; ... ; (d)

unde: (a) declară etichetele pentru GOTO, *ifs* număr de maxim 4 cifre

(b) definește constantele

(c) definește tipurile

(d) declară variabilele și tipul lor

Declarațiile sînt valabile în interiorul blocului în care apar, sînt opționale și ordinea lor este: LABEL, CONST, TYPE și VAR.

INSTRUCȚIUNI SIMPLE ȘI STRUCTURATE (notație: instr-s)**Instrucțiunea de atribuire**

id-var := *expresie* (a)

id-func := *expresie* (b)

unde (a) atribuire de valori variabilelor

(b) atribuire de valori funcțiilor; utilizată în blocul de definire al funcției respective

Instrucțiunea GOTO

GOTO *ifs*

Instrucțiune de salt necondiționat. Eticheta este definită în același bloc și saltul se poate face numai în același bloc la același nivel (nu se poate ieși din FOR ... DO, funcții sau proceduri).

Formatul etichetei este:

ifs : *instrucțiune*

Instrucțiunea IF-THEN

IF *expresie* **THEN** *instrucțiune*

Dacă *expresie* = TRUE atunci se execută *instrucțiune*; în orice situație se continuă cu instrucțiunile următoare.

Instrucțiunea IF-THEN-ELSE

IF *expresie* **THEN** *instrucțiune-1* **ELSE** *instrucțiune-2*

Dacă *expresie* = TRUE atunci se execută *instrucțiune-1* altfel se execută *instrucțiune-2*; în orice situație se continuă cu instrucțiunile următoare.

Instrucțiunea CASE-OF

CASE *expresie* **OF** *const* , ... : *instrucțiune* ; ... **END**

Dacă *expresie* are valoarea printre constante atunci se execută instrucțiunea corespunzătoare; în orice situație se continuă cu instrucțiunile de după END.

Nu este permis formatul vid: CASE OF END.

Instrucțiunea CASE-OF-ELSE

CASE *expresie* **OF** *const* , ... : *instrucțiune* ; ... **ELSE** *instrucțiune* **END**

Dacă *expresie* are valoarea printre constante atunci se execută instrucțiunea corespunzătoare; altfel se execută instrucțiunea de după ELSE; în orice situație se continuă cu instrucțiunile de după END.

Instrucțiunea WHILE-DO

WHILE *expresie* **DO** *instrucțiune*

Se efectuează *instrucțiune* atîta timp cît *expresie* = TRUE.

Instrucțiunea REPEAT-UNTIL

REPEAT *instrucțiune* ; ... **UNTIL** *expresie*

Se efectuează secvența de instrucțiuni pînă cînd *expresie* = FALSE dar cel puțin odată.

Instrucțiunea FOR-TO-DO**FOR** *id-var* := *expresie-1* **TO** *expresie-2* **DO** *instrucțiune*

Bucă cu contor, la fiecare trecere *id-var* capătă valoarea următoare **SUCC** (*id-var*), *id-var* (contorul) la valori între *expresie-1* și *expresie-2* inclusiv. Dacă *expresie-1* > *expresie-2* atunci instrucțiunea nu se efectuează. Contorul nu poate fi variabilă de tip RECORD.

Instrucțiunea FOR-DOWNTO-DO**FOR** *id-var* := *expresie-1* **DOWNTO** *expresie-2* **DO** *instrucțiune*

Bucă cu contor, la fiecare trecere *id-var* capătă valoarea anterioară **PRED** (*id-var*), *id-var* (contorul) la valori între *expresie-1* și *expresie-2* inclusiv. Dacă *expresie-1* < *expresie-2* atunci instrucțiunea nu se efectuează. Contorul nu poate fi variabilă de tip RECORD.

Instrucțiunea WITH-DO**WITH** *id-var*, ... **DO** *instrucțiune*

Simplifică operațiile cu variabile-RECORD, în instrucțiune se folosesc numai identificatorii de cîmp. Nu sînt permise imbricările WITH.

Instrucțiunea vidă

```
;;
BEGIN ;
; END
BEGIN BEGIN
BEGIN END
```

unde ;, BEGIN și END sînt delimitatorii instrucțiunii.

Instrucțiunea vidă poate fi etichetată

Apelul de procedură

```
id-procedură
id-procedură listă-parametri
```

Instrucțiune (notație: instrucțiune)

```
instr-s
BEGIN instrucțiune ; ... END
```

Listă de parametri (notație: list-param)

```
(parametru)
(parametru ; ...)
```

Parametru (notație: parametru)

```
id : tip
id, ... : tip
VAR id : tip
VAR id, ... : tip
```

Definiția funcțiilor (notație: def-func)

```
FUNCTION id-func list-param : tip ; bloc ;
```

unde *tip* este tipul valorii (unice) returnate de funcție.

Funcțiile pot fi apelate recursiv. La apelarea unei funcții se face o corespondență biunivocă între lista parametrilor formali și lista parametrilor efectivi din apel și tipurile trebuie să corespundă.

Definiția procedurilor (notație: def-procedură)

PROCEDURE *id-procedură* ; *bloc* ;

PROCEDURE *id-procedură list-param* ; *bloc* ;

Procedurile nu pot fi utilizate în expresii. La apelarea unei proceduri se face o corespondență biunivocă între lista parametrilor formali și lista parametrilor efectivi.

Declararea procedurilor (notație: decl-procedură)

PROCEDURE *id-procedură* ; **FORWARD**

PROCEDURE *id-procedură list-param* ; **FORWARD**

Se utilizează dacă se intenționează utilizarea procedurilor înainte de definirea lor.

Bloc (notație: bloc)

decl-LABEL

decl-CONST

decl-TYPE

decl-VAR

decl-procedură

...

decl-procedură

def-func

...

def-func

def-procedură

...

def-procedură

BEGIN *instrucțiune* ; ... **END**

unde *decl-procedură*, *def-func* și *def-procedură* pot fi utilizate în orice ordine permisă adică nu pot fi utilizate funcțiile nedefinite anterior sau proceduri nedefinite sau nedeclarate anterior; toate secvențele sînt opționale cu excepția secvenței **BEGIN ... END**.

Structura de program complet

PROGRAM *id* ; *bloc* .

unde instrucțiunile din cadrul blocului sînt instrucțiunile principale de execuție.

4.4 Manipularea datelor în memorie

Întregi

stocați pe 2 bytes în complement față de 2

Reali

stocați pe 4 bytes în formă binară:

bit31 = semn: 0 pozitiv/1 negativ

bit30-bit8 = mantisa în formă normalizată: 1

bit7-bit0 = exponentul, întreg în complement față de 2

sînt stocați în regiștrii H, L, D, E iar în memorie în ordine inversă

Tablouri

$nr\text{-bytes} = nr\text{-elemente} * nr\text{-bytes-pe-element}$

Record-uri

se sumează *nr-bytes* pentru fiecare cîmp

Seturi

sînt stocate ca șiruri de biți:

$nr\text{-bytes} = 1 + (nr\text{-total-de-elemente-ale-multimii} - 1) \text{ DIV } 8$

Pointeri

ocupă 2 bytes fiecare, reprezentînd o adresă conform convenției

Intel

Variabilele globale

sînt declarate în programul principal și li se alocă spațiu în memorie pe stivă, de la adresa £8000 în jos, în ordinea declarării lor, atît spațiu cît necesită fiecare

Variabilele locale

sînt declarate în sub-blocuri și sînt stocate de la adresa IX+4 în jos, în ordinea declarării lor, atît spațiu cît necesită fiecare; registrul IX este setat la începutul fiecărui bloc

Transmisia parametrilor și valorilor de retur

sînt stocați de la adresa IX+2 în sus, în ordine inversă celei din list-param, astfel:

constantele sînt transmise prin valoare

variabilele sînt transmise prin referință (adresă: 2 bytes)

valorile de retur ale funcțiilor sînt puse la adresa cea mai mare

4.5 Funcții și proceduri predefinite

Parametri efectivi (notație: param-ef)

expresie

În expresii nu pot apărea proceduri sau funcții (nu sînt valabile ca parametri).

4.5.1 Funcții

FUNCȚII DE TRANSFORMARE A TIPULUI

CHR (*param-ef*)

parametrul este de tip întreg

funcția returnează caracterul corespunzător codului ASCII; de tip char

ENTIER (*param-ef*)

parametrul este de tip întreg sau real

funcția returnează întregul mai mic sau egal cu parametrul; de tip întreg

ORD (*param-ef*)

parametrul este de tip scalar dar nu real

funcția returnează numărul de ordine al parametrului din setul căruia îi aparține;

de tip întreg

ROUND (*param-ef*)

parametrul este de tip întreg sau real

funcția returnează parametrul rotunjit la întreg; de tip întreg

TRUNC (*param-ef*)

parametrul este de tip întreg sau real

funcția returnează partea întreagă a parametrului; de tip întreg

FUNȚII ARITMETICE

Parametrii sînt de tip întreg și/sau real.

ABS (*param-ef*)

funcția returnează valoarea absolută (modulul) parametrului; de același tip

ARCTAN (*param-ef*)

funcția returnează arctangenta unghiului-parametru exprimat în radiani; de tip real

COS (*param-ef*)

funcția returnează cosinusul unghiului-parametru exprimat în radiani; de tip real

EXP (*param-ef*)

funcția returnează valoarea $e^{\text{parametru}}$ ($e=2.71828$, numărul lui Euler); de tip real

FRAC (*param-ef*)

funcția returnează *parametru* - **ENTIER** (*parametru*); de tip real

ODD (*param-ef*)

funcția returnează **TRUE** dacă parametrul este impar, altfel **FALSE**; de tip logic

SIN (*param-ef*)

funcția returnează sinusul unghiului-parametru exprimat în radiani; de tip real

SQR (*param-ef*)

funcția returnează pătratul parametrului; de același tip

SQRT (*param-ef*)

funcția returnează radicalul (rădăcina patrată) parametrului; de tip real

TAN (*param-ef*)

funcția returnează tangenta unghiului-parametru exprimat în radiani; de tip real

FUNȚII DE UZ GENERAL

ADDR (*id-var*)

funcția returnează adresa de început a variabilei în memorie; de tip întreg

EOLN

funcția returnează **TRUE** dacă următorul caracter citit este sfîrșit de linie (**EOL=CHR(13)**), altfel **FALSE**; de tip logic

INCH

funcția citește tastatura și returnează caracterul tastat sau altfel **CHR(0)**; de tip char

INP (*param-ef*)

parametrul este de tip întreg

se realizează instrucțiunile: **LD BC, param-ef; IN A, (C)** și funcția returnează caracterul citit; de tip char

PEEK (*param-ef, tip*)

parametrul este o adresă din memorie, de tip întreg

funcția returnează date din memorie conform tip, care definește tipul funcției

PRED (*param-ef*)

parametrul este de tip scalar cu excepția real

funcția returnează predecesorul parametrului; de același tip

RANDOM

funcția returnează un număr pseudo-aleator între 0 și 255; de tip întreg

SIZE (*id-var*)

funcția returnează numărul de bytes ocupați de variabilă în memorie; de tip întreg

SUCC (*param-ef*)

parametrul este de tip scalar cu excepția real

funcția returnează succesorul parametrului; de același tip

4.5.2 Proceduri

PROCEDURI DE SCRIERE/CITIRE

PAGE

Șterge ecranul respectiv, avansează imprimanta la începutul paginii următoare.

READ (*id-var*)

READ (*id-var*, ...)

Citește date de la buffer-ul de intrare al tastaturii și le atribuie variabilelor în ordinea din lista. **EOLN=TRUE** la începutul programului.

READLN (*id-var*)

READLN (*id-var*, ...)

Analog **READ** dar citește într-un nou buffer de intrare al tastaturii.

WRITE (*param-ef*)

WRITE (*param-ef*, ...)

unde parametrul este de tipul:

expresie-1

(1)

expresie-1 : *expresie-2*

(2)

expresie-1 : *expresie-2* : *expresie-3*

(3)

expresie-1 : *expresie-2* : H

(4)

Scrive *expresie-1* la consolă, conform formatului:

- *expresie-1* de tip întreg: se utilizează (1) sau (2); *expresie-2* specifică numărul de cifre de afișat, dacă *expresie-1* are mai multe cifre atunci *expresie-2* este ignorată

- *expresie-1* de tip întreg hexazecimal: se utilizează (d); *expresie-2* specifică spațiul/numărul de caractere alocat scrierii

- *expresie-1* de tip real: se utilizează (a), (b) sau (c)

- *expresie-3* reprezintă numărul de zecimale ale numărului de afișat în cazul în care *expresie-2* este dimensionată corespunzător

- *expresie-1* este de tip șir: se utilizează (a) sau (b), se afișează șirul, un caracter pe un spațiu, cu blank-uri suplimentare dacă *expresie-2* o permite

- *expresie-1* este de tip logic: se utilizează (a) sau (b), se afișează TRUE/FALSE, pe 4/5 spații, cu blank-uri suplimentare dacă *expresie-2* o permite

Dacă se utilizează (a) sau *expresie-2* <= 7 se afișează numărul în format standard. Dacă *expresie-2* este între 8 și 12 atunci se afișează numărul în format standard cu 1 pînă la 5 zecimale după . . Dacă *expresie-2* >= 13 se afișează blank-uri suplimentare. În general un număr real are nevoie de 12 spații-caracter pentru afișare. Cursorul (poziția următoare de afișare) este pe prima poziție liberă de după datele afișate.

WRITELN (*param-ef*)

WRITELN (*param-ef*, ...)

Analog **WRITE** dar cursorul (poziția următoare de afișare) este pe linia următoare liberă de după datele afișate.

PROCEDURI DE MANIPULARE A VARIABILELOR DINAMICE

MARK (*id-var*)

Memorează într-o variabilă-pointer starea zonei de memorie Heap în care sînt stocate variabilele dinamice.

NEW (*id-var*)

Creează o variabilă dinamică a cărei adresă se află în variabilă.

RELEASE(*id-var*)

Eliberează zona de memorie Heap de variabilele dinamice conform ultimei proceduri MARK.

PROCEDURI DE UZ GENERAL

HALT

Oprește rularea programului.

INLINE (*cod-byte*, ...)

Șirul de bytes reprezintă un program în cod-mașină Z80, care este inserat în programul compilat.

OUT (*param-ef-1*, *param-ef-2*)

param-ef-1 este de tip întreg, *param-ef-2* este de tip caracter

Se realizează instrucțiunile: LD BC, *param-ef-1*; OUT (C), A.

POKE (*param-ef*, *id-var*)

Stochează variabila în memorie, începînd de la adresa dată de parametru, care poate fi întreg în complement față de 2 sau hexazecimal. Variabila poate fi de orice tip cu excepția seturilor.

TIN (*param-ef-1*, *param-ef-2*)

Încarcă de pe bandă magnetică un fișier al cărui nume este dat de *param-ef-1* (de tip șir) la adresa dată de *param-ef-2* (de tip întreg)

TOUT(*param-ef-1*, *param-ef-2*, *param-ef-3*)

Salvează pe bandă magnetică într-un fișier numit *param-ef-1* (șir de maxim 8 caractere) o zonă de memorie a cărei adresă de început este dată de *param-ef-2*, numărul de bytes (lungimea) fiind dat de *param-ef-3*.

USER (*param-ef*)

Apelează o subrutină în cod-mașină a cărei adresă de început este dată de parametrul de tip întreg, pentru valori mai decît 32767 se utilizează întregi negativi (întregii sînt memorați în complement față de 2). Subrutina trebuie să se termine cu instrucțiunea RET și să nu afecteze registrul IX. Se pot utiliza și constante hexazecimale.

4.6 Directive de compilare

Directivele de compilare se introduc în program separate de numărul de linie cu un blank.

\$A +verifică ca indexul unei variabile-tablou să fie în domeniul declarat,

- implicit +
- \$C** + scanează tastatura în timpul execuției programului, implicit +
- \$F** *șir* șirul trebuie să aibă 8 caractere (eventual blank-uri) și să fie separat de \$F cu un blank; comanda completează de pe bandă magnetică un fișier program-sursă salvat cu comanda W
- \$I** + verifică dacă există depășiri la compararea a două numere întregi (depășire există dacă între cele 2 numere este o diferență mai mare decât MAXINT (caz în care se poate genera rezultat incorect), implicit -
- \$L** + listează liniile program-sursă la compilare, implicit +
- \$O** + verifică depășirile pentru adunările și scăderile în numere întregi, implicit +
- \$P** comută listarea pe imprimantă/ecran, implicit ecran
- \$S** + verifică dacă există riscul depășirii stivei la apelul unei proceduri sau funcții, implicit +

4.7 Codurile erorilor

4.7.1 Erori la compilare

- 1 număr prea mare
- 2 lipsește ;
- 3 identificador nedeclarat
- 4 lipsește un identificador
- 5 se utilizează = nu :=
- 6 lipsește =
- 7 instrucțiunea nu poate începe cu acest identificador
- 8 lipsește :=
- 9 lipsește)
- 10 tip eronat
- 11 lipsește .
- 12 lipsește un factor
- 13 lipsește o constantă
- 14 identificadorul nu este o constantă
- 15 lipsește THEN
- 16 lipsește DO
- 17 lipsește TO/DOWNTO
- 18 lipsește (
- 19 tipul expresiei nu poate fi scris
- 20 lipsește OF
- 21 lipsește ,
- 22 lipsește :
- 23 lipsește PROGRAM
- 24 parametrul este de tip variabilă
- 25 lipsește BEGIN

- 26 parametrul în **READ** este variabilă
 27 nu pot fi comparate expresiile de tipul respectiv
 28 tipul trebuie sa fie **INTEGER** sau **REAL**
 29 acest tip de variabilă nu poate fi citit
 30 nu este identificator de tip
 31 lipsește exponentul numărului real
 32 lipsește o expresie scalară
 33 nu există șiruri nule (se utilizează **CHR(0)**)
 34 lipsește [
 35 lipsește]
 36 indexul tabloului trebuie să fie de tip scalar
 37 lipsește ..
 38 lipsește] sau ; în declarația tablourilor
 39 marginea inferioară este mai mare decât marginea superioară
 40 set prea mare (sint permise maximum 256 elemente)
 41 rezultatul funcției trebuie să fie identificator de tip
 42 lipsește] sau , în set
 43 lipsește) , , sau .. în set
 44 tipul parametrului trebuie să fie identificator de tip
 45 primul factor într-o instrucțiune, cu excepția asignării, nu poate fi un set nul
 46 lipsește un scalar, inclusiv real
 47 lipsește un scalar, exclusiv real
 48 seturi incompatibile
 49 seturile nu pot fi comparate cu < sau >
 50 lipsește **BEGIN**, **CONST**, **FORWARD**, **LABEL** sau **VAR**
 51 lipsește o cifră hexazecimală
 52 seturile nu pot fi utilizate în **POKE**
 53 tablou prea mare
 54 lipsește : sau **END** în **RECORD**
 55 lipsește un identificator de cîmp
 56 lipsește variabila după **WITH**
 57 variabila din **WITH** trebuie să fie de tip **RECORD**
 58 lipsește **WITH**
 59 lipsește întreg fără semn după **LABEL**
 60 lipsește întreg fără semn după **GOTO**
 61 etichetă utilizată la un nivel eronat
 62 etichetă nedeclarată
 63 parametrul lui **SIZE** trebuie să fie o variabilă
 64 pointerii pot fi comparați numai prin egalități
 67 format incorect de scriere pentru întregi
 68 șirurile nu pot conține caracterul sfîrșit-de-linie (**EOL**)
 69 parametrul lui **NEW**, **MARK** sau **RELEASE** trebuie să fie o variabilă de tip pointer
 70 parametrul lui **ADDR** trebuie să fie o variabilă

4.7.2 Erori la execuție

- 1 halt

- 2 depășire
- 3 depășirea memoriei RAM
- 4 împărțire cu 0
- 5 index prea mic
- 6 index prea mare
- 7 eroare matematică
- 8 număr prea mare
- 9 lipsește un număr
- 10 linie prea lungă
- 11 lipsește un exponent

prin compararea punctului de oprire unde a apărut eroarea cu adresele liniilor compilate se poate determina locul unde a apărut eroarea.

5.1 Instalarea*

C este un limbaj de programare de uz general, structurat care are un grad de portabilitate mai ridicat decît alte limbaje (cu excepția Forth) și al cărui compilator generează un cod mai eficient (rapid), datorită particularităților de limbaj. Deși compilatorul este orientat pe calculatoarele de 8 biți, care lucrează cu bandă magnetică, totuși C nu își poate arăta întreaga sa forță pe calculatoarele de tip Spectrum.

Încărcarea se face cu **LOAD ""** prin care (după încărcare) se lansează direct compilatorul.

Pentru a intra în editor se folosește Edit (CS1)

Pentru ca un program-sursă să fie compilat acesta trebuie să se termine cu EOF (end-of-file) (SSI).

Textul-sursă poate fi tastat direct în modul compilator, introdus din editor (memorie) cu comanda **# include** sau încărcat de pe bandă magnetică cu comanda **# include nume-fișier**.

Taste funcționale:

Enter **NEW LINE (LINE FEED)** cod 10

CS1 (Caps Shift & 1): Edit, afișat UDG 138, intrare în editor

SSI (Symbol Shift & I): EOF, cod -1, afișat UDG 134, sfîrșit de fișier

CS0 (Caps Shift & 0): Delete, șterge un caracter

Intrările și ieșirile, în C, se fac prin fișiere: bandă magnetică, memorie, ecran (fișier output), tastatură (fișier input).

5.2 Editorul

Prompt-ul > așteaptă o comandă de tipul:

comandă număr-1 , număr-2 , șir-1 , șir-2

unde *număr-1* , *număr-2* sînt numere întregi între 1 și 32767; *șir-1* , *șir-2* sînt șiruri de caractere de lungime între 0 și 20. Blank-ul nu este semnificativ decît în șiruri; , este separator. Comenzile pot fi introduse cu litere mari sau mici.

5.2.1 Generarea textului-sursă

I *număr-1* , *număr-2*

Introduce numărătoarea automată a liniilor.

Valori implicite: *număr-1* = 10, *număr-2* = 10.

O linie text-sursă este de forma:

număr-linie secvență-program

Lungimea *secvenței-program* este de cel mult 80 de caractere, numărul de linie servește doar pentru facilitarea editării.

Orice linie nou introdusă, cu comanda I sau direct după prompt-ul >, se suprapune peste textul anterior astfel încît liniile anterioare cu același număr de linie se șterg.

5.2.2 Listare

L *număr-1* , *număr-2*

Listează liniile textului-sursă avînd numerele între *număr-1* și *număr-2*; orice tastă efectuează scroll, cu excepția CS1.

Valori implicite: *număr-1* = 1, *număr-2* = 32767

K *număr*

Setează numărul de linii de listat înainte de scroll.

Valori implicite: *număr* = 16

W *număr-1* , *număr-2*

Listează la imprimantă liniile textului-sursă avînd numerele între *număr-1* și *număr-2*. Valori implicite: *număr-1* = 1, *număr-2* = 32767

5.2.3 Editare

D *număr-1* , *număr-2*

Sterge liniile textului-sursă, avînd numerele între *număr-1* și *număr-2*.

Valori implicite: fără

Dacă *număr-1* > *număr-2* sau lipsesc argumente comanda este ignorată.

M *număr-1* , *număr-2*

Introduce linia *număr-1* la linia *număr-2*; linia *număr-1* rămîne în text; linia *număr-2* anterioară se șterge.

Valori implicite: fără

N *număr-1* , *număr-2*

Renumerotează textul-sursă începînd cu *număr-1* și cu pasul *număr-2*

Valori implicite: fără

F *număr-1* , *număr-2* , *șir-1* , *șir-2*

șir-1 este căutat în liniile cu numerele între *număr-1* și *număr-2*; dacă este găsit atunci se intră în editorul de linie (comanda E) cu cursorul pe primul caracter al *șir-1*; *șir-2* substituie *șir-1*.

Valori implicite: cele aflate în buffer la o utilizare anterioară, altfel fără

E *număr*

Editor de linie.

Valori implicite: fără

Dacă *număr* lipsește, comanda este ignorată.

Sub-comenzi:

blank cursor dreapta

CS0 cursor stînga

CS8 avans tab

E terminarea editării, cu modificări

C supra-scriere, începînd de la poziția curentă a cursorului

F caută următorul *șir-1* din comanda F

I înserează caractere la poziția curentă a cursorului

K șterge un caracter la poziția cursorului

L listează restul liniei în curs de editare

Q terminarea editării, modificările sînt anulate

R anulează modificările

S înlocuiește *șir-1* din comanda F

X avansează cursorul la sfîrșitul liniei și introduce sub-comanda I

Z șterge toate caracterele de la cursor la sfîrșitul liniei

5.2.4 Comenzi de bandă magnetică

P *număr-1* , *număr-2* , *șir*

Salvează pe bandă magnetică liniile din textul-sursă între *număr-1* și *număr-2*, într-un fișier cu numele *șir*.

Valori implicite: cele aflate în buffer la o utilizare anterioară, altfel fără.

G , , *șir*

Încarcă de pe bandă magnetică fișierul (text-sursă) *șir* la sfîrșitul textului-sursă și liniile sînt renumerotate.

Valori implicite: dacă *șir* lipsește se încarcă primul fișier

5.2.5 Comenzi de uz general

B

Retur în BASIC Spectrum, reîntoarcere în C cu: **RANDOMIZE USR 25200**

C

Retur în compilator.

S, , d

Schimbă separatorul operanzilor din , în *d* ; *d* nu poate fi blank.

V

Afișează buffer-ul.

5.3 Structuri de limbaj

5.3.1 Introducere

C este un limbaj cu grad înalt de portabilitate. Standardul C este definit în "The C Programming Language", Appendix A, autori: Brian W. Kenirghan și Dennis M. Ritchie.

5.3.2 Convenții lexicale

Tipuri de simboluri:

cuvinte-cheie	separatori
identificatori	constante
operatori	șiruri

Blank-urile, tabulatorii, NEW LINE și comentariile (spațiile albe) sînt ignorate la compilare.

Comentarii

/ text */*

Identificatori

literă literă/cifră

(sublinirea) este considerată literă; sînt semnificative numai primele 8 caractere; identificatorii externi sînt utilizați doar pentru declararea înainte a tipului funcțiilor

Cuvinte-cheie

Cuvintele-cheie nu pot fi utilizate în alte scopuri decît cele definite și trebuie introduse în litere minuscule.

auto	extern	static
break	for	struct
case	goto	switch
cast	if	typedef
char	int	union
continue	long	unsigned
default	register	void
do	return	while
else	short	
entry	sizeof	

Constante (notație: const)

a) Întreg (int)

secvență-de-cifre

int este identic cu long

Dacă începe cu 0 se consideră număr octal, dacă începe cu 0x sau 0X se consideră număr hexazecimal.

b) Întreg lung (long int)

Dacă numărul se termină cu l sau L se consideră long (dar este identic cu int)

c) Caracter

'caracter'

Caractere speciale (pe un byte):

/n	New Line (Line Feed)
/t	Tab
/b	Back Space
/r	Carriage Return
/f	Form Feed
//	Back Slash
'	Single Quote
/număr	bit-pattern
/0	Null

unde număr este octal pe 1-3 cifre

d) Numere în virgulă flotantă: neimplementat

Șiruri

"șir"

Caracteristici hard și de implementare

microprocesor Z80

set de caractere și coduri ASCII

char: 1byte

int: 2bytes

short: 2bytes

float:

double:

range:

neimplementat

neimplementat

-

5.3.3 Interpretarea identificatorilor

Un identificator-variabilă are 2 caracteristici:

clasa de memorie

tipul

Clasele de memorie sînt:

- auto:** sînt valabile doar în interiorul blocului în care au fost definite și valorile nu sînt reținute între 2 apeluri succesive
- static:** sînt valabile doar în interiorul blocului în care au fost definite și valorile sînt reținute între 2 apeluri succesive
- extern:** variabilele sînt globale, putînd fi apelate de orice funcție
- register:** recunoscut dar tratat ca auto (din motive hard)

Tipurile de bază sînt:

- char:** tip caracter
- int:** tip întreg (pe 2 bytes) cu sub-tipurile:
short int, int, long int (manipulate toate la fel)
unsigned int: întreg fără semn

Cu tipurile de bază se pot construi:

- tablouri cu componente de același tip
- funcții cu rezultat de orice tip
- adrese ale datelor de orice tip
- structuri (**struct**) cu componente diferite
- uniuni (**union**) de componente diferite

În general metodele de construire a unor noi tipuri se pot folosi succesiv și recursiv.

5.3.4 Obiecte și s-valori (l-values)

Obiectele sînt regiuni de memorie manipulabile.

S-valorile (l-values) sînt expresii care se află în partea stîngă a unei atribuiri și sînt operanzi de tip referință, adică:

- identificator de variabilă simplă
- element de tablou (variabilă indexată)
- element de structură desemnat prin numele structurii
- element de structură desemnat prin adresa structurii
- pointer

5.3.5 Conversii

Caractere și întregi

Caracterele și întregii pot fi utilizați unul în locul celuilalt fără restricții. Caracterele apar ca întregi între 0 și 255 (fără extinderea semnului).

Adrese (pointeri și întregi)

Adresele sînt tratate ca întregi fără semn, dar rezultatul scăderii a două adrese este un întreg cu semn.

Întregi fără semn

Dacă într-o expresie există un întreg fără semn, atunci ceilalți întregi sînt convertiți în întregi fără semn, iar rezultatul este de tip întreg fără semn.

Conversii aritmetice

Operandii de tip caracter sînt transformați în tip `int`.

Dacă există operanzi de tip întreg fără semn, atunci toți ceilalți sînt convertiți la acest tip și rezultatul este de acest tip.

Altfel rezultatul este de tip întreg (tipurile `short` și `long` sînt la fel ca `int`).

5.3.6 Expresii (notație: `expr`)

Expresiile sînt formate din operanzi și operatori:

operandii sînt valori sau adrese de memorie

operatorii se aplică operanzilor

rezultatul evaluării unei expresii este o valoare sau o adresă

Ordinea de aplicare a operatorilor depinde de prioritatea asociată fiecărui operator și de prezența parantezelor utilizate pentru inversarea ordinii de prioritate/aplicare

În fiecare sub-paragraf operatorii au același grad de prioritate. Sub-paragrafele sînt ordonate în ordinea descreșterii gradului de prioritate. Altfel ordinea de evaluare a unei expresii nu este definită.

Depășirile întregi și împărțirile cu 0 sînt ignorate.

Expresii primare (notație: `expr-p`)

<code>id</code>	(a)
<code>const</code>	(b)
<code>șir</code>	(c)
<code>(expr)</code>	(d)
<code>expr-p [expr] ... [expr]</code>	(e)
<code>expr-p ()</code>	(f)
<code>expr-p (expr)</code>	(g)
<code>expr-p (expr ..., expr)</code>	(h)
<code>s-valoare . id</code>	(i)
<code>expr-p -> id</code>	(j)

unde:

- (a) `id` are tipul definit în declarații, un `id` declarat ca tablou este de tip pointer și are asociată adresa primului element din tablou
- (b) constanta tip `int`, caracterele au tipul `int`
- (c) are ca valoare adresa primului caracter din șir (șirul este asimilat cu un tablou de caractere)
- (d) utilizarea parantezelor pentru marcarea ordinii de evaluare, parantezele nu modifică interpretarea *s-valorilor*
- (e) utilizarea variabilelor indexate (element de tablou) unde `expr-p` este numele tabloului și `expr` este indexul;
- (f),(g),(h) `expr-p [expr]` este identică cu `*((expr-p) + (expr))`
- (f),(g),(h) apeluri de funcții ale căror parametri sînt `expr, ...`; parametrii funcțiilor sînt transmiși prin valoare pe stiva de parametri, `char` este convertit la `int`, tablourile sînt transmise prin pointeri; sînt permise apelurile recursive ale oricăror funcții, funcțiile returnează rezultatele conform tipului definit sau (implicit) `int`
- (i) utilizarea elementelor variabilelor de tip `struct` sau `union`
- (j) `expr-p` trebuie să fie un pointer către o variabilă `struct` sau `union` și

id trebuie să fie un identificator al unui membru al variabilei;
 rezultatul este s-valoarea pointer referitoare la membrul respectiv
expr-p este identică cu $(*expr-p) . id$

Operatorii din expresiile primare (), [], -> au prioritate maximă față de ceilalți operatori și se evaluează de la stânga la dreapta.

Expresii unare (notație: *expr-u*)

* <i>expr</i>	(a)
& <i>s-valoare</i>	(b)
- <i>expr</i>	(c)
! <i>expr</i>	(d)
~ <i>expr</i>	(e)
++ <i>s-valoare</i>	(f)
-- <i>s-valoare</i>	(g)
<i>s-valoare</i> ++	(h)
<i>s-valoare</i> --	(i)
cast (<i>nume-tip</i>) <i>expresie</i>	(j)
size of <i>expr</i>	(k)
size of (<i>nume-tip</i>)	(l)

unde:

- (a) * este operator de indirectare; *expr* este un pointer, rezultatul este o *s-valoare*
- (b) rezultatul este un pointer către obiectul referit de *s-valoare*
- (c) - schimbă semnul operatorului pe care îl precede
- (d) ! este operator de negație logică:
 $expr = 0 \Rightarrow !expr = 1$
 $expr < 0 \Rightarrow !expr = 0$
- (e) ~ este operator de negare pe biți (complement față de 1)
- (f), (h) ++ este operator de incrementare a *s-valorii*
 (f) incrementează *s-valoarea* apoi produce valoarea incrementată
 ++ *s-valoare* este identică cu *s-valoare* + = 1
 (h) produce *s-valoarea* apoi o incrementează
 tipul rezultatului este dat de tipul *s-valorii*
- (g), (i) -- este operator de decrementare a *s-valorii*
 (g) decrementează *s-valoarea* apoi produce valoarea decrementată
 -- *s-valoare* este identică cu *s-valoare* - = 1
 (i) produce *s-valoarea* apoi o decrementează
 tipul rezultatului este dat de tipul *s-valorii*
- (j) forțează conversia rezultatului evaluării expresiei la *nume-tip*
 (predefinit sau definit anterior)
- (k) returnează dimensiunea operandului în bytes
- (l) returnează dimensiunea unui obiect de categoria *nume-tip*
 (predefinit sau definit anterior)

Operatori multiplicativi

expr * *expr*
expr / *expr*
expr % *expr*

- unde: * este operator de înmulțire
 / este operator de împărțire
 % este operator modulo (rezultatul este restul împărțirii primei *expr* la

a 2-a)

Operatorii se aplică de la stînga la dreapta.

Operatori aditivi

$expr + expr$

$expr - expr$

unde: $+$ $expr$ este operator de adunare

$-$ $expr$ este operator de scădere

Operatorii se aplică de la stînga la dreapta.

Adunarea/scăderea unui întreg la un pointer se face după înmulțirea întregului cu lungimea valorii adresate de pointer, iar rezultatul este un pointer.

Scăderea a 2 pointeri, asupra aceluiași tablou returnează ca valoare numărul de obiecte aflate între cei 2 pointeri (la evaluare se împarte rezultatul preliminar cu lungimea obiectului).

Operatori de shift

$expr-1 \ll expr-2$

$expr-1 \gg expr-2$

unde \ll și \gg realizează deplasarea la stînga (\ll)/dreapta (\gg) a $expr-1$ tratat ca o configurație binară cu un număr de poziții egal cu $expr-2$. Deplasarea se face cu introducerea de 0-uri printr-o extremitate și pierderea cifrelor binare care ies prin cealaltă extremitate.

\gg realizează shift aritmetic dacă $expr-1$ este de tip `int` (cu excepția `unsigned`).

Operatori relaționali

$expr-1 < expr-2$

$expr-1 > expr-2$

$expr-1 <= expr-2$

$expr-1 >= expr-2$

unde: $<$ mai mic $<=$ mai mic sau egal

$>$ mai mare $>=$ mai mare sau egal

Operatorii se aplică de la stînga la dreapta.

Rezultatul este de tip `int`: 0 pentru fals, 1 pentru adevărat.

2 pointeri pot fi comparați, rezultatul fiind semnificativ doar dacă sînt în cadrul aceluiași tablou.

Operatori de egalitate

$expr-1 == expr-2$

$expr-1 != expr-2$

unde: $==$ egal cu $!=$ nu este egal cu

Operatorii se aplică de la stînga la dreapta.

Rezultatul este de tip `int`: 0 pentru fals, 1 pentru adevărat.

2 pointeri pot fi comparați, rezultatul fiind semnificativ doar dacă sînt în cadrul aceluiași tablou. Un pointer egal cu 0 semnifică că acesta nu adresează nimic.

Operatorul binar și

$expr \& expr$

$\&$ realizează și logic pe biți; este operator asociativ.

Operatorul binar sau-exclusiv

$expr \wedge expr$

realizează sau-exclusiv logic pe biți; este operator asociativ.

Operatorul binar sau

$expr | expr$

$|$ realizează sau logic pe biți; este operator asociativ.

Operatorul logic și

expr && expr

&& realizează și logic; se aplică de la stînga la dreapta
 Rezultatul este de tip **int**: 0 dacă una din *expr* este nulă
 1 dacă ambele *expr* sînt nenule

Operatorii pot fi de tipuri diferite, dar predefinite sau pointeri.

Operatorul logic sau

expr || expr

realizează sau logic; se aplică de la stînga la dreapta
 Rezultatul este de tip **int**: 0 dacă ambele *expr* sînt nule
 1 dacă una din *expr* este nenulă

Operatorii pot fi de tipuri diferite, dar predefinite sau pointeri.

Operatorul condițional

expr-1 ? expr-2 : expr-3

Se aplică de la dreapta la stînga.

Dacă *expr-1* este nenulă, rezultatul este valoarea data de *expr-2* altfel rezultatul este *expr-3*.

Operatori de atribuire

s-valoare = expr

(a)

s-valoare op = expr

(b)

unde *op* este operator care poate fi: +, -, *, /, %, >>, <<, ^, !

Atribuirea se aplică de la dreapta la stînga.

(a) reprezintă operația de atribuire, *s-valoare* capătă valoarea *expr*

(b) realizează *s-valoare = s-valoare op (expr)*; pentru + sau - *s-valorile* pot fi pointeri

Rezultatul este de tipul *s-valorii*. Nu este permisă atribuirea între pointeri și întregi cu excepția 0 care înseamnă nici o adresă.

5.3.7 Declarații (notație: decl)

specificatori-decl ;

specificatori-decl listă-decl ;

unde *specificatori-decl* este:

specificator-tip

specificator-tip specificatori-decl

specificator-cm

specificator-cm specificatori-decl

specificatori-decl reprezintă o secvență de tipuri și clase de memorare.

listă-decl conține identicatorii de declarat.

Clase de memorare (notație: cm)

auto

static

extern

register

tipedef

extern este utilizat doar pentru funcțiile din bibliotecă, care nu sînt de tip **int**.

Specificatori de tip (notație: specificator-tip)

char
short
int
long
unsigned
specificator-union-sau-struct
specificator-typedef

int poate fi precedat de short, long sau unsigned.

Lista declaratorilor (notație: lista-decl)

declarator-*i*
declarator-*i* , ... declarator-*i*

unde declarator-*i* este:

declarator
declarator inițializator

inițializator realizează inițializarea declaratorilor și declarator este:

<i>id</i>	(a)
(declarator)	(b)
* declarator	(c)
declarator ()	(d)
declarator { }	(e)
declarator [expr-const]	(f)
declarator ...	(g)

Obiectele declarate sînt variabile valorice (a), pointeri (c), funcții (d) sau tablouri (e, f, g).

Funcțiile nu pot returna pointeri către acestea, nu există tablouri de funcții, dar pot exista tablouri de pointeri către funcții.

Structuri și uniuni (notație: specificator-union-sau-struct)

struct-sau-union {listă-decl-struct }
struct-sau-union *id* {listă-decl-struct }
struct-sau-union *id*

unde struct-sau-union este:

struct
union

și listă-decl-struct este:

declarație-struct
declarație-struct ... declarație-struct

unde declarație-struct este:

specificator-tip listă-declaratori-struct

unde listă-declaratori-struct este:

declarator
declarator ..., declarator

Structura este o variabilă care poate conține mai mulți membri de tipuri diferite, sînt permise cîmpuri vide (neocupate de membri), adresele obiectelor declarate sînt în ordine crescătoare conform citirii de la stînga la dreapta.

Uniunea este o variabilă care la un moment dat poate conține doar un membru al unei colecții, membrii putînd fi de tipuri diferite.

Structurile sau uniunile pot fi autoreferite numai prin pointeri. Identificatorii variabilelor, structurilor, uniunilor și membrilor trebuie să fie diferite.

Inițializarea (notație: inițializator)

= expr-const

(a)

= {*expr-const...* , *expr-const* } (b)

unde: (a) inițializează variabilele scalare

(b) inițializează variabilele tablou și **struct**

Variabilele **static** și **external** neinițializate explicit sînt inițializate cu 0. Variabilele **auto** nu pot fi inițializate.

Nume de tipuri (notație: nume-tip)

Un tip poate fi definit prin:

typedef tip id

unde: *tip* este un tip predefinit sau construcții **struct/union**

id este utilizat ca specificator de tip

5.3.8 Instrucțiuni

Instrucțiunile sînt executate secvențial cu excepția cazurilor menționate explicit

Expresii

expresie ;

În general expresia este de atribuire sau apel de funcție.

Instrucțiunea compusă/bloc (notație: bloc)

{*listă-decl listă-instrucțiuni* }

unde *listă-decl* este:

/ nimic */*

decl

decl ; ... decl ;

și *listă-instrucțiuni* este:

instrucțiune ;

instrucțiune ; ... instrucțiune ;

Instrucțiunea if

if (*expr*) *instrucțiune-1*

if (*expr*) *instrucțiune-1* **else** *instrucțiune-2*

Dacă *expr* este diferită de 0 (adeverată) atunci se execută *instrucțiune-1* altfel se execută *instrucțiune-2* (dacă există). În instrucțiunile **if** imbricate **else** aparține instrucțiunii **if** celei mai apropiate anterior definite.

Instrucțiunea while

while (*expr*) *instrucțiune*

Instrucțiunea este executată atîta timp cît expresie este diferită de 0 (adeverată), testul se efectuează înaintea execuției instrucțiunii.

Instrucțiunea do-while

do *instrucțiune* **while** (*expr*)

Instrucțiunea este executată pînă cînd *expr* devine 0 (fals), testul se efectuează după execuția instrucțiunii.

Instrucțiunea for

for (*expr-1 ; expr-2 ; expr-3*) *instrucțiune*

reprezintă bucla cu contor, unde:

expr-1 reprezintă inițializarea contorului

expr-2 reprezintă testul de sfîrșit al buclei

expr-3 reprezintă actualizarea contorului

Instrucțiunea este echivalentă cu

```
expr-1
```

```
while (expr-2) {instrucțiune expr-3 ; }
```

expr-1 / 2 / 3 sînt opționale, dar separatorii ; sînt obligatorii.

Valori implicite: *expr-2* = 1 (adevărat, bucla infinită)

Instrucțiunea switch

```
switch (expr )
```

```
{case expr-const : ... expr-const : instrucțiune
```

```
...
```

```
case expr-const : ... expr-const : instrucțiune
```

```
default : instrucțiune }
```

Se evaluează *expr* și se execută salt la prima instrucțiune etichetată **case** pentru care *expr* este egală cu *expr-const* sau la **default** în caz contrar, dacă există, și se execută în continuare toate instrucțiunile care urmează. Poate exista o singură etichetă, opțională, **default**; pot exista mai multe etichete **case** atașate unei instrucțiuni.

Instrucțiunea break

```
break ;
```

Forțează salt la sfîrșitul instrucțiunilor: **while**, **for** sau **switch** în care se găsește (imediat superioare).

Instrucțiunea continue

```
continue ;
```

Forțează salt la sfîrșitul corpului de instrucțiuni ale buclelor **while**, **do-while** sau **for** în care se găsește (imediat superioare) fără părăsirea acestora.

Instrucțiunea return

```
return ;
```

(a)

```
return expr ;
```

(b)

este opțională, se consideră implicit (a); specifică valoarea returnată de funcție unde:

(a) valoarea returnată este nedefinită (nespecificată)

(b) *expr* este valoarea returnată de funcție, care este convertită dacă este cazul la tipul funcției

Instrucțiunea goto

```
goto id-eticheta
```

generează salt necondiționat la instrucțiunea etichetată, *id* este valabil în funcția curentă.

Etichetarea instrucțiunilor

```
id-etichetă : instrucțiune
```

unde *id-etichetă* reprezintă eticheta instrucțiunii, utilizată de **goto**.

Instrucțiunea vidă

```
;
```

efectul instrucțiunii vide este nul, instrucțiunea vidă poate fi etichetată.

Instrucțiunea inline

```
inline (expr-const )
```

```
inline (expr-const ... , expr-const )
```

permite introducerea unui program în cod-mașină Z80

Dacă *expr-const* este între 0 și 255 atunci este evaluată la 1 byte altfel este evaluată la 2 bytes.

5.3.9 Definiții externe

Un program C este compus dintr-o serie de definiții externe printre care se află o funcție program-principal denumită **main** care este apelată la începutul rulării programului.

Definițiile externe au clasa de memorie **extern** implicit sau **static** și sînt valabile în tot fișierul-program.

Definițiile externe au același format ca toate celelalte definiții cu excepția faptului ca la acest nivel pot fi definite funcțiile.

Definițiile funcțiilor (notație: func)

specificator-decl declator-func corp-func

declator-func corp-func

unde *declator-func* este:

declator ()

sau

declator (id)

sau

declator (id ... , id)

și *corp-func* este:

listă-decl bloc

Specificatorii claselor de memorie permisi sînt **extern** și **static** cu semnificație și utilizare normală, **auto** plasat între *declator-func* și *corp-func* permite definirea și utilizarea funcțiilor cu număr variabil de parametri. Compilatorul plasează argumentele efective ale funcției (la apelul ei) după celelalte argumente ca o secvență de bytes. Această secvență este utilizată de funcție ca argument al ei. *lista-decl* conține declarații referitoare doar la parametrii funcției.

Definițiile datelor

decl

clasele de memorizare ale datelor pot fi doar **extern** (implicit) sau **static**.

5.3.10 Domeniile de valabilitate ale declarațiilor

Întregul program-sursă este compilat odată

Domeniul de valabilitate a argumentelor formale ale funcțiilor este în blocul de definiție al funcției respective

Domeniul de valabilitate lexical al unei variabile externe declarate în afara funcțiilor este programul-sursă

Funcția **main** este definită fără argumente

5.3.11 Preprocesorul

Compilatorul C conține un preprocesor care caută în programul-sursă și interpretează toate liniile care încep cu caracterul **#**.

Aceste linii de control ale compilării se numesc directive preprocesor

Directiva define**# define** *id* *șir*înlocuiește în programul-sursă *id* cu *șir***Directiva include**

```

# include (a)
# include "nume-fișier" (b)
# include <nume-fișier > (c)
# include nume-fișier (d)
# include ?nume-fișier ? (c)

```

unde:

- (a) compilează programul-sursă din editor
 (b), (c), (d) încarcă și compilează un text-sursă fișier de pe bandă magnetică cu numele *nume-fișier*
 (e) în textul-sursă, fișier de pe bandă magnetică cu *nume-fișier*, sînt căutate, încărcate și compilate doar funcțiile utilizate în program. În fișier funcțiile care apelează alte funcții trebuie puse la început pentru evitarea recitirii fișierului.

Directiva list**# list**

unde + activează / - dezactivează listarea liniilor următoare ale textului-sursă în decursul compilării.

Directiva direct**# direct**

unde + activează / - dezactivează execuția directă a instrucțiunilor (modul interpretor)

Directiva translate**# translate** *nume-fișier*salvează pe bandă magnetică codul compilat într-un fișier cu numele *nume-fișier*. Programul-obiect poate fi încărcat și executat cu secvența:**LOAD CODE "" : RANDOMIZE USR 25200****Directiva error****# error**

șterge mesajele de eroare, eliberînd memoria pentru programele mari

5.3.12 Expresii constante (notație: expr-const)

Expresiile evaluate la o constantă implică utilizarea constantelor **int** și **char**, expresii **sizeof**, operatorii unari: -, ~; operatorii binari: +, *, /, %, &, |, ^, <<, >>, ==, !=, <, >, <=, >=, operatorul ternar ?: și parantezele () pentru definirea ordinii de evaluare a operatorilor.

5.4 Funcții predefinite

Funcțiile predefinite sînt compilate și introduse automat în program.

```
int fclose (fp) FILE *fp ;
```

Închide fișierul (deschis pentru scriere/citire de *fopen) indicat de pointerul *fp* și eliberează buffer-ul (pentru un alt fișier).

```
FILE *fopen (name , mode ) char *name , *mode ;
```

Deschide un fișier denumit de șirul *name* pentru operația definită de șirul *mode* care poate fi: *r* pentru citire, *w* pentru scriere; are ca rezultat un pointer către structura de descriere a fișierului utilizat de alte funcții, pointerul este 0 în caz de eroare.

```
void fprintf (fp , control , arg1 , arg2 ... ) FILE *fp ; char *control ;
```

Scrie în fișierul indicat de pointerul *fp* conform șirului *control* care poate conține caractere normale de scris (șiruri) (inclusiv caractere speciale) și /sau specificatori de format:

```
% d   întreg zecimal
% o   octal fără semn
% x   hexazecimal fără semn
% u   întreg fără semn
% c   caracter
% s   șir terminat cu caracterul \0
%%   scrie caracterul %
```

Între % și caracter pot fi modificatori care semnifică:

```
-      aliniere la stînga (implicit la dreapta)
0      completare cu 0
șir-de-cifre  specificator de lățime minimă cîmp
.șir-de-cifre specificator de lățime maximă cîmp
L      pentru long int, acceptat dar ignorat
```

(se pot utiliza mai mulți modificatori în același specificator)

```
int getc (fp) FILE *fp ;
```

Citește următorul caracter din fișierul indicat de pointerul *fp*, returnează -1 pentru EOF.

```
int getchar ( ) ;
```

Citește un caracter de la tastatură într-un buffer pînă la apăsarea Enter. Se poate utiliza CS0.

```
int isalpha (c) char(c) ;
```

Returnează 1 (adevărat) dacă *c* este literă, altfel 0 (fals).

```
int isdigit (c) char(c) ;
```

Returnează 1 (adevărat) dacă *c* este cifră, altfel 0 (fals).

```
int islower (c) char(c) ;
```

Returnează 1 (adevărat) dacă *c* este literă minusculă, altfel 0 (fals).

```
int isspace (c) char(c) ;
```

Returnează 1 (adevărat) dacă *c* este blank, new-line sau tab, altfel 0 (fals).

```
int isupper (c) char(c) ;
```

Returnează 1 (adevărat) dacă *c* este literă majusculă, altfel 0 (fals).

```
int keyhit ( ) ;
```

Returnează 1 (adevărat) dacă a fost apăsată o tastă, altfel 0 (fals); trebuie resetată înainte de o reutilizare prin resetarea bitului 5 al variabilei sistem FLAGS (utilizează rutina 028E).

```
void move (dest , source , length )
char *dest , *source ; unsigned lenght ;
```

Mută un șir de bytes, de lungime *length* și care începe la *source*, începînd de la *dest*. Copierea se face înainte de o eventuală suprascriere (fără pierderea informației).

```
int putc (c , fp ) int c ; FILE *fp ;
```

Scrie caracterul *c* în fișierul indicat de pointerul *fp* și returnează caracterul ca rezultat.

```
void printf (control , arg1 , arg2 ... ) char *control ;
```

Scrie, conform șirului *control*, pe ecran: *arg1*, *arg2* ... Șirul *control* are aceleași caracteristici cu cel utilizat de funcția *fprintf*.

```
int putchar (c ) int c ;
```

Scrie caracterul *c* pe ecran și returnează caracterul ca rezultat.

```
int rawin ( )
```

Citește un caracter de la tastatură fără prelucrarea codurilor. Bitul 5 al variabilei-sistem FLAGS este setat la apăsarea tastei, citește variabila-sistem LAST-K și re setează fanionul.

```
void sprintf (s , control , arg1 , arg2 ...) char *s , *control ;
```

Scrie, conform șirului *control*, în șirul *s*: *arg1*, *arg2*... Șirul *control* are aceleași caracteristici ca cel utilizat de funcția *fprintf*.

```
void swap (p , q , lenght ) char *p , *q ; unsigned lenght ;
```

Inversează 2 zone de memorie avînd numărul de bytes *length* fiecare și pointerii *p* și *q*.

```
char tolower (c ) char c ;
```

Returnează caracterul literă minusculă dacă *c* a fost literă majusculă, altfel *c* nu este modificat.

```
char toupper(c ) char c ;
```

Returnează caracterul literă majusculă dacă *c* a fost literă minusculă, altfel *c* nu este modificat.

```
int ungetc (c , fp ) int c , FILE *fp ;
```

Pune caracterul *c* înapoi în fișierul *fp*.

5.5 Codurile erorilor

- 0 lipsește caracter
- 1 numerele în virgulă flotantă nu sînt implementate
- 2 constantă caracter eronată
- 3 nu este comandă preprocesor
- 4 buffer-ul macro plin
- 5 doar identificatorii pot fi definiți ca macro
- 6 parametrii macro nu sînt implemeniați
- 7 fișierul nu poate fi deschis

- 8 # **include** imbricate nu sînt implementate
9 lipsește **while**
10 **break** poate fi utilizat doar în **do-loop**, **while**, **for** sau **switch**
11 **continue** poate fi utilizat doar în **do-loop**, **while** sau **for**
12 **case** sau **default** pot fi utilizate doar în **switch**
13 pot fi utilizate maxim 50 etichete **case** într-o instrucțiune **switch**
14 poate fi utilizată doar o etichetă **default** într-o instrucțiune **switch**
15 lipsește eticheta în **goto**
16 etichetă utilizată de mai multe ori (în **goto** sau ca etichetă)
17 **#direct** nu poate fi utilizat cu **#translate**
18 tabela de variabile utilizată la compilare este plină (prea multe variabile globale și/sau funcții cu prea multe variabile)
19 tabela de tipuri utilizată la compilare este plină
20 declarații multiple pentru același identificator, tip eronat
21 declarații multiple pentru același identificator, clasa de memorie eronată
22 tabela de variabile globale utilizate la compilare este plină
23 zona de date globale este plină (variabile globale prea multe și/sau cu dimensiuni prea mari)
24 declarație multiplă pentru același identificator
25 tabela de variabile locale utilizată la compilare este plină
26 parametrul efectiv nu este în lista parametrilor formali a funcției apelate
27 variabilă nedefinită
28 funcția nu poate returna acest tip
29 tablourile de funcții nu sînt permise
30 prea multe paranteze și/sau argumente în expresie
31 prea mulți operatori în expresie
32 tipurile variabilelor sînt incompatibile în expresie
33 tipul operandului sau operatorul eronat în expresie
34 lipsește o *s-valoare* (*l-value*)
35 nu este membru al unei structuri sau uniuni
36 lipsește o expresie primară
37 variabilă nedefinită
38 lipsește un nume de tip
39 lipsește o expresie constantă
40 expresia poate fi folosită doar ca apel de funcție
41 caracterul : utilizat incorect
42 lipsește o *s-valoare* (*l-value*) la operatorul de atribuire
43 lipsește : la operatorul ternar ?
44 lipsește un pointer
45 tipul parametrului funcției este incorect
46 numerele în virgulă flotantă nu sînt implementate
47 operatorul nu funcționează cu numere în virgulă flotantă
48 declarație eronată
49 clasa de memorie eronată în context
51 identificator utilizat multiplu în structuri sau uniuni
52 structura utilizată ca parametru al funcției trebuie declarată
53 o structura nu se poate contine pe sine
54 declarator eronat
55 lipsește) în declarația funcției (în declarație nu se dau parametrii formali)
56 listă eronată a parametrilor formali ai funcției
57 tipul trebuie să fie funcție
60 memorie plină

- 61 variabilele **auto** pot fi inițializate doar prin atribuire sau cu funcția **move**
 62 clasa de memorie nu permite inițializarea
 63 tipul nu permite inițializarea
 64 prea multe date de inițializare

5.6 Manipularea datelor în memorie

5.6.1 Dimensiunea tipurilor

char	1 byte
int	2 bytes, byte-ul semnificativ la adresa superioară (convenția Intel)
short	identic int
long	identic int
unsigned	identic int
pointer	2 bytes, byte-ul semnificativ la adresa superioară (convenția Intel)
tablouri	nr-elemente * nr-bytes-pe-element elementul inferior la adresa inferioară
struct	variabilele multiplu indexate sînt considerate tablouri de tablouri numărul de bytes necesari este egal cu suma numărului de bytes necesari fiecărui membru, primul membru la adresa inferioară
union	numărul de bytes este egal cu numărul de bytes ocupați de variabila cea mai extinsă, membrii sînt aliniați la adresa inferioară

5.6.2 Clase de memorie

constante	sînt incluse în programul compilat
extern,static	începînd de la adresa dată de variabila-sistem RAMTOP în jos, adresele sînt incluse în codul compilat
auto	sînt puse pe stivă și accesate prin regiștrii IX și SP

5.6.3 Funcții

Parametrii efectivi ai funcțiilor sînt evaluați de la stînga la dreapta și puși pe stivă în aceeași ordine. Pentru funcțiile cu număr variabil de argumente se trimit 2 bytes suplimentari care reprezintă numărul de bytes al argumentelor și sînt utilizați pentru accesarea argumentelor și resetarea stivei.

Funcțiile sînt apelate prin instrucțiunea CALL, se încarcă registrul SP în IX apoi se alocă spațiu în continuare pe stivă pentru variabilele proprii (locale).

5.6.4 Fișiere

Toate fișierele (inclusiv programul-sursă) sînt manipulate ca șiruri de caractere. Fișierele pe banda magnetica au un header de tip Spectrum și un șir de blocuri de 512 bytes fiecare din care primii 2 bytes reprezintă un contor de caractere; bitul cel mai semnificativ al contorului ultimului bloc este setat.

5.6.5 Regiștrii Z80

HL este utilizat pentru returnarea rezultatului funcțiilor și evaluarea expresiilor. A', F', B', C', D', E', H', L', IY, I și R nu sînt utilizați de compilator sau de codul compilat.

6

FORTH

6.1 Instalarea*

Alături de Pascal, Forth este cel mai eficient limbaj de programare pe calculatoarele compatibile Spectrum, fiind însă mai dificil de utilizat.

După încărcarea cu LOAD "" prompt-ul C așteaptă introducerea comenzilor (mod interpretor).

6.2 Structuri de limbaj

Forth poate lucra în mod interpretor, când așteaptă introducerea unor comenzi de la tastatură, sau mod compilator, când compilează programele.

Forth poate manipula 2 stive:

stiva de date este cea curentă; notație SD

stiva de retur notație SR

Notație pentru stive: (înainte-de-execuție — — după-execuție)
elementul din dreapta este în virful stivei

Programul constă în definirea succesivă a unor cuvinte (subrutine) pînă la definirea cuvîntului de nivel cel mai înalt care prin execuție generează rezultatul programului.

Un cuvînt poate fi definit cu ajutorul cuvîntului de definiție : :

: *cuvînt-de-definit*

cuvînt-1

*Fig-Forth este produs înregistrat al firmei ABERSOFT

...
cuvînt-n

unde *cuvînt-1 ...cuvînt-n* sînt cuvinte definite anterior (în același vocabular sau în vocabulare ierarhic superioare), ; este tot un cuvînt.

Un cuvînt poate fi definit și cu ajutorul cuvintelor de definiție (predefinite în Forth sau definite de utilizator)

cuvînt-de-definiție cuvînt-de-definit
cuvînt

Cuvintele sînt șiruri de caractere afișabile, separate de blank-uri, de orice lungime, dar din care doar primele 31 de caractere sînt semnificative.

Numerele sînt șiruri de cifre și pot fi exprimate în orice bază de la 0 la 36; pentru baze superioare lui 10 se utilizează literele ca cifre.

Distincția între cuvinte și numere se face astfel:

a) la introducerea unui șir de caractere (de la consolă sau RAM-DISC) se încearcă interpretarea primului șir ca un cuvînt prin căutarea acestuia în dicționar, începînd de la ultimul cuvînt, definit (în vocabularul de context);

b) dacă șirul nu este cuvînt din dicționar se încearcă interpretarea acestuia ca un număr în baza curentă; dacă încercarea este reușită numărul, transformat în corespondentul binar, este împins pe vîrfurile stivei SD; numerele în dublă lungime (pe 4 bytes) sînt semnalizate prin introducerea caracterului . în orice poziție

c) dacă șirul nu este nici număr atunci se generează mesaj de eroare
 Expresile se scriu în notație postfixată (operatorul la sfîrșit) astfel încît toți operatorii au aceeași prioritate și se evaluează de la stînga la dreapta (operatorii sînt tot cuvinte).

6.3 Structura memoriei

6.3.1 Harta memoriei

5E00	Variabile-utilizator
5E40	Interpretorul de adrese PUSHDE, PUSHL, NEXT
5E77	Dicționar FENCE, DP HERE PAD Stiva-date SO, TIB
CB40	Buffer-terminal Stiva-retur RO

CBE0 Buffer-intrare/ieşire cu RAM-DISC
 FIRST
 LIMIT
 D000 RAM-DISC
 FC00 UDG

Între 5E00 și 5E31 (inclusiv) sînt variabilele-utilizator predefinite (celelalte 10 locații sînt libere) care sînt utilizate la fel ca variabilele normale, dar sînt definite printr-un deplasament față de adresa de început a tabelii (5E00).

Forth simulează un RAM-DISC de 10 kbytes (10 ecrane) și care poate fi salvat sau încărcat pe bandă magnetică.

6.3.2 Structura cuvintelor

1) Cuvinte definite prin :

NFA byte-def
 byte-caracter-1
 ...
 byte-caracter-n
 LFA NFA-cuvînt-precedent
 CFA CFA-:
 PFA CFA-cuvînt
 ...
 CFA-;S

unde byte-def are structura:

7	6	5	4	3	2	1	0
1	P	S					lungime-nume (n)

P = bit de precedență, P = 1 pentru cuvinte imediate

S = bit de validare a definiției, S = 0 pentru definiție validă

2) Variabile

Antetul unui cuvînt-variabilă de la NFA la CFA este identic; urmează 2 bytes adresa de tratare a variabilei, apoi 2/4 bytes pentru memorarea valorii curente a variabilei.

3) Constante

Antetul unui cuvînt-constantă de la NFA la CFA este identic; urmează 2 bytes adresa de tratare a constantei, apoi 2/4 bytes pentru memorarea valorii constantei.

4) Vocabulare

Antetul unui cuvînt-constantă de la NFA la CFA este identic; urmează:

CFA adresa de tratare pentru cuvinte de definiție (DOES >)
 PFA adresa de tratare pentru cuvinte-vocabular
 PFA+2 A081, antet de cuvînt fictiv
 PFA+4 NFA al ultimei definiții făcută sub vocabularul respectiv
 PFA=6 legătura către vocabularul precedent în care a fost creat (FORTH fiind primul vocabular conține 0)

5) Cuvinte scrise în cod-mașină

Antetul unui cuvînt scris în cod maşină de la NFA la CFA este identic; în CFA este adresa CFA+2, urmează o secvenţă în cod-maşină care se termină cu un salt la adresa NEXT, PUSHHL sau PUSHDE.

6.4 Editorul

Editorul este scris în Forth (cuvintele editorului sînt definite în Forth), este construit ca vocabular separat cu numele **EDITOR**, şi accesul în el se face prin tastarea **EDITOR** (urmat de Enter).

Programul-sursă este organizat în ecrane avînd numerele între 0 şi 32767. Fiecare ecran are 16 linii numerotate de la 0 la 15 cu pînă la 64 de caractere pe fiecare linie.

Notăție:	n	număr cu semn pe 2 bytes
	adr	adresa pe 2 bytes
	c	caracter, cod ASCII, 1 byte
	text	şir de caractere afişabile
#LAG		(— —adr n)
	adr	adresa cursorului
	n	numărul caracterelor rămase pînă la sfîrşit de linie
#LEAD		(— —adr n)
	adr	adresa liniei
	n	deplasamentul cursorului în linie
#LOCATE		(— —n1 n2)
	n1	numărul de linie unde este cursorul
	n2	deplasamentul cursorului în linie
B		(— —)
		mişcă cursorul înapoi cu lungimea textului din PAD
C text		(— —)
		inserează <i>text</i> după poziţia cursorului
CLEAR		(n— —)
		şterge ecranul n
COPY		(n1 n2— —)
		copiază ecranul n1 în ecranul n2
D		(n— —)
		şterge linia n dar o reţine în PAD
DELETE		(n— —)
		şterge n caractere de la poziţia cursorului
E		(n— —)
		şterge linia n cu blank-uri
F text		(— —)
		introduce <i>text</i> în PAD şi îl caută în ecranul curent
FIND		(— —)
		caută textul din PAD în ecranul curent de la poziţia cursorului
H		(n— —)
		copiază conţinutul liniei n din ecranul curent în PAD
I		(n— —)

	introduce textul din PAD în linia n din ecranul curent
L	(— —)
	listează ecranul curent
1LINE	(— —n)
	caută în linia curentă de la poziția curentă a cursorului textul din PAD și returnează n=1 dacă îl găsește, altfel n=0
M	(n— —)
	avansează cursorul cu n caractere
MATCH	(adr1 n1 adr2 n2— —n3 n4)
	caută textul care începe la adr1 de lungime n1 în textul care începe la adr2 pe lungime n2; revine cu n4=1 dacă textul a fost găsit și n3 deplasamentul de unde începe textul
-MOVE	(adr n— —)
	copiază o linie de text care începe la adr în linia n a ecranului curent
N	(— —)
	caută în continuare textul din PAD
P text	(n— —)
	pune text în linia n
R	(n— —)
	înlocuiește linia n cu textul din PAD
S	(n— —)
	inserează o linie vidă n, următoarele linii se mută în jos, ultima linie se pierde
T	(n— —)
	afișează linia n din ecranul curent și o introduce în PAD
-TEXT	(adr1 n1 adr2— —n2)
	compară 2 texte de la adr1 și adr2 pe lungimea n1, revine cu n2=1 dacă sînt egale, altfel n2=0
TILL text	(— —)
	șterge caracterele de la poziția cursorului la text
TOP	(— —)
	poziționează cursorul la începutul ecranului curent
X text	(— —)
	caută un text de la cursorul curent și șterge textul

6.5 Cuvinte predefinite

Notații:

Pentru date:

adr	adresă de memorie, 2 bytes
b	byte
c	cod de caracter ASCII, 1 byte
d	număr cu semn în dublă lungime, 4 bytes
f	fanion logic, f=0 fals, f=1 adevărat, 2 bytes
n	număr cu semn, 2 bytes
u	număr fără semn, 2 bytes

ud număr fără semn în dublă lungime, 4 bytes
 text șir de caractere afișabile
 listă listă de cuvinte și/sau numere, separate de blank-uri

Pentru cuvinte:

C poate fi utilizat numai în mod compilare (definiții)
 E poate fi utilizat numai în mod execuție (interpretor)
 I este imediat (are bitul de precedență setat)
 U variabilă-utilizator, memorarea lor este diferită de cele normale dar sînt utilizate identic
 F scris în Forth
 m scris în cod-mașină

I	(n adr— —)	m
	memorează n la adr	
ICSP	(— —)	F
	memorează pointerul SD în variabila CSP	
#	(ud1— —ud2)	F
	generează din ud1 o cifră caracter ASCII în șirul de ieșire; se utilizează pentru formatarea numerelor, numai între <# și #>	
#>	(ud— —adr n)	F
	termină formatarea numerelor, revine cu adresa și lungimea șirului, parametrii utilizabili de TYPE	
#BUFF	(— —n)	
	constantă (utilizată de FLUSH)	
#S	(ud— —00)	F
	convertește toate cifrele din ud în șir de caractere ASCII, se utilizează numai între <# și #>	
'text	(— —PFA)	F
	returnează PFA al cuvîntului text	
((— —)	F, I, C
	început de comentariu care se termină cu caracterul)	
(.)	(— —)	F
	cuvînt compilat în definiții de către ."	
(;CODE)	(— —)	F
	cuvînt compilat în definiții de către ;CODE	
(+LOOP)	(— —)	m,C
	cuvînt compilat în definiții de către +LOOP	
(ABORT)	(— —)	F
	identic cu ABORT	
(DO)	(n1 n2— —)	m
	cuvînt compilat în definiții de către DO	
(FIND)	(adr NFA— —PFA b 1) sau (adr NFA— —0)	m
	caută în dicționar începînd de la NFA șirul care începe la adr, revine cu PFA, lungimea șirului și fanion adevărat dacă l-a găsit, altfel fanion fals	
(LINE)	(n1 n2— —adr n)	F,C
	n1 număr de linie, n2 număr de ecran adr adresă șirului de caractere, n lungimea șirului de caractere al liniei	
(LOOP)	(— —)	m,C
	cuvînt compilat în definiții de către LOOP	
(NUMBER)	(d1 adr— —d2 adr2)	F
	convertește textul de la adr1 + 1, conform BASE, într-un număr acumulat în	

	d1 și lăsat ca d2; adr2 este adresa primului caracter neconvertibil	
(TAPE)	(n— —) m	
	rutină folosită de LOADT, SAVET și VERIFY pentru manipularea RAM-DISC-ului pe bandă magnetică:	
	n=0 se realizează salvare (SAVET)	
	n=1 se realizează încărcare (LOADT)	
	n=2 se realizează verificare (VERIFY)	
	RAM-DISC-ul este salvat pe bandă magnetică într-un fișier cu numele DISC; header-ul este construit la adresa 75E6 (un alt header se află la adresa 75F7)	
*	(n1 n2— —n3)	F
	n3=n1*n2	
*/	(n1 n2 n3— —n4)	F
	d=n1*n2; n4=d/n3	
*/MOD	(n1 n2 n3— —n4 n5)	F
	d=n1*n2; n4=d/n3; n5=restul împărțirii d/n3	
+	(n1 n2— —n3)	F
	n3=n1+n2	
+I	(n adr— —)	m
	adaugă n la valoarea conținută la adr	
+ -	(n1 n2— —n3)	F
	aplică semnul lui n2 asupra lui n1 care devine n3	
+BUF	(adr— —f)	F
	cuvînt utilizat de BUFFER	
+LOOP	(n— —)	F
	marchează sfîrșitul buclei cu contor: DO <i>cuvînt</i> ... +LOOP	
	incrementează contorul cu n și iese din buclă dacă:	
	n>0 și contor >=valoarea-limită sau	
	n<0 și contor <= valoarea-limită	
+ORIGIN	(n— —adr)	F
	adaugă n la valoarea 5E40 (adresa de început a interpretorului de adrese)	
,	(n— —)	F
	introduce n în următoarea intrare liberă din dicționar	
-	(n1 n2— —n3)	F
	n3=n1-n2	
- ->	(— —)	F
	continuă interpretarea din ecranul următor	
-DUP	(n— —n n)	F
	sau (0— —0)	
	duplică elementul din virful stivei dacă este nenul	
-FIND <i>text</i>	(— —1 n PFA)	F,C
	sau (— —0)	
	caută cuvîntul <i>text</i> în dicționar, revine cu lungimea și PFA, altfel f=0	
-TRAILING	(adr1 n1— —adr2 n2)	F
	elimină blank-urile din șirul de la adr1 de lungime n1 și revine cu adresa și lungimea șirului format	
.	(n— —)	F
	afișează n conform BASE	
" <i>text</i>	(— —)	F, I
	afișează textul care se termină cu caracterul "	
.CPU	(— —)	F
	afișează mesajul 48k SPECTRUM	

.LINE	(n1 n2— —)	F
	afișează linia n1 din ecranul curent	
.R	(n1 n2— —)	F
	afișează numărul n1, conform BASE pe n2 spații, aliniat la dreapta	
/	(n1 n2— —n3)	F
	n3 = n1/n2	
/MOD	(n1 n2— —n3 n4)	F
	n4 = n1/n2, n3 = restul împărțirii n1/n2	
0	(— —0)	
	constantă	
1	(— —1)	
	constantă	
2	(— —2)	
	constantă	
3	(— —3)	
	constantă	
0<	(n— —f)	m
	f = 1 dacă n < 0, altfel f = 0	
0=	(n— —f)	m
	f = 1 dacă n = 0, altfel f = 0	
0BRANCH	(f— —)	m, C
	salt relativ conform numărului, cu semn, în complement față de 2, care urmează în definiția compilată a unui cuvânt dacă f < > 0; cuvânt compilat, în definiții, de IF, UNTIL, WHILE	
1+	(n1— —n2)	F
	n2 = n1 + 1	
2I	(d adr— —)	m
	memorează d la adr	
2@	(adr— —d)	m
	citește d de la adr	
2+	(n1— —n2)	F
	n2 = n1 + 2	
2CONSTANT text	(d— —)	F/m
	creează o constantă în dublă lungime, căreia i se atribuie valoarea d; execuția ulterioară a cuvântului text determină plasarea d pe stiva SD	
2DROP	(d— —)	F
	șterge d de pe SD	
2OVER	(d1 d2— —d1 d2 d3)	F
	duplică penultimul element pe SD	
2SWAP	(d1 d2— —d2 d1)	F
	interschimbă ultimele 2 elemente din vârful stivei	
2VARIABLE text	(d— —)	F/m
	creează o variabilă în dublă lungime căreia i se atribuie ca primă valoare d; execuția cuvântului text determină plasarea adresei variabilei pe stiva SD	
:	(— —)	F/m, I, E
	: <i>cuvânt-de-definit</i> cuvânt ... ; semnalizează începutul definiției unui cuvânt; Forth trece în mod compilare; vocabularul CONTEXT devine CURRENT	
;	(— —)	F, I, C
	indică sfârșitul definiției (:) unui cuvânt, Forth trece în mod interpretor, compilează ;S la sfârșitul cuvântului	
;CODE	(— —)	F, C, I

indică faptul că în definiția unui cuvînt urmează o secvență în cod-mașină. Forth utilizează regiștrii Z80 astfel:

A registru de uz general

BC conține IP, pointer către adresa CFA a următorului cuvînt de executat

DE registru de uz general

HL registru de uz general, prin intermediul acestuia se manipulează SR al cărui pointer se află la adresa 5E68.

IX conține 5E00, pointer către tabela variabilelor-utilizator

IY nu este folosit

SP manipulează SD

setul alternat de regiștri nu este folosit; secvența de cod-mașină trebuie să se termine cu un salt la una din adresele date de NEXT, PUSHHL, PUSHDE

;S	(— —)	m
	cuvînt folosit pentru terminarea compilării programului-sursă din ecrane, redînd controlul consolei; este compilat de : la sfîrșitul definițiilor	
<	(n1 n2— —f)	m
	f=1 dacă n1 < n2 altfel f=0	
<#	(d— —d)	F
	semnalizează începutul formatării numerelor	
<BUILDS	(— —)	F, C
	utilizat pentru definirea cuvintelor de definiție împreună cu DOES> ; semnalizează începutul secvenței executate la compilare de către cuvîntul de definiție	
	<i>: cuvînt-de-definiție</i>	
	<BUILDS <i>listă1</i>	
	DOES> <i>listă2</i> ;	
=	(n1 n2— —f)	F
	f=1 dacă n1 = n2, altfel f=0	
>	(n1 n2— —f)	F
	f=1 dacă n1 > n2, altfel f=0	
>R	SD {— —n} și SR (n— —)	m, C
	transferă virful SR pe SD, se utilizează în pereche cu R >	
?	(adr— —)	F
	afișează ca număr cu semn valoarea aflată la adr	
?COMP	(— —)	F
	testează dacă Forth se află în mod compilare, afișează mesaj de eroare dacă nu	
?CSP	(— —)	F
	afișează mesaj de eroare dacă pointerul stivei diferă de CSP	
?ERROR	(f n— —)	F
	afișează mesaj de eroare n dacă f=1	
?EXEC	(— —)	F
	afișează mesaj de eroare dacă Forth nu este în mod interpretor (execuție)	
?LOADING	(— —)	F
	testează dacă blocul de disc este încărcat în memorie, dacă BLK=0 emite mesaj de eroare	
?PAIRS	(n1 n2— —)	F
	emite mesaj de eroare dacă n1 < > n2 (utilizat pentru verificări sintactice)	
?STACK	(— —)	F

	verifică dacă stiva se află în limitele permise, emite mesaj de eroare în caz contrar	
?TERMINAL	(— —f)	F
	f = 1 dacă a fost apăsată tasta Break	
@	(adr — —n)	m
	citește n de la adr	
ABORT	(— —)	F
	resetează SD și SR, comută Forth în mod interpretor, repune BASE pe DECIMAL, vocabularul CONTEXT și CURRENT devine FORTH	
ABS	(n1 — —n2)	F
	returnează n2 modulul numărului n1	
AGAIN	(— —)	F
	marchează sfârșitul buclei infinite: BEGIN listă AGAIN	
ALLOT	(n — —)	F
	rezervă n bytes în dicționar, începînd cu prima intrare liberă	
AND	(n1 n2 — —n3)	m
	efectuează și-logic pe biți între n1 și n2 returnînd rezultatul n3	
AT	(n1 n2 — —)	F
	poziționează cursorul de afișare pe linia n1 (între 0 și 31) și coloana n2 (între 0 și 31)	
ATTR	(n1 n2 — —b)	F
	returnează atributul de culoare al unui caracter aflat în rîndul n1 (între 0 și 21) și coloana n2 (între 0 și 31)	
	biți 0-2 codul de culoare INK	
	biți 3-5 codul de culoare PAPER	
	bit 6 cod BRIGHT, 0 fără, 1 cu	
	bit 7 cod FLASH, 0 fără, 1 cu	
	cod de culoare	
	0 negru 4 verde	
	1 albastru 5 cyan	
	2 roșu 6 galben	
	3 magenta 7 alb	
B/BUF	(— —n)	
	n = 128 lungimea buffer-ului, în bytes, utilizat în operații cu memoria externă (RAM-DISC); constantă	
B/SCR	(— —n)	
	numărul de buffere de ecran; constantă	
BACK	(adr — —)	F
	calculează saltul relativ între HERE și adr și îl compilează (după BRANCH sau OBRANCH)	
BASE	(adr — —)	U
	variabilă-utilizator care conține baza curentă de numerație	
BEGIN	(— —adr)	F, C, I
	marchează începutul unei bucle:	
	BEGIN listă AGAIN	
	BEGIN listă UNTIL	
	BEGIN listă1 WHILE listă2 REPEAT	
	la compilare lasă pe stivă adresa HERE utilizată (la compilare) de cuvintele de sfârșit de buclă	
BL	(— —n)	
	n = 32, codul ASCII pentru blank; constantă	
BLANKS	(adr n — —)	F

depune în memorie începînd de la adr, pe n bytes, codul ASCII al blank-ului

BLEEP	(n1 n2— —)	m
emite un sunet la difuzor		
n1 = f*t unde f este frecvența sunetului (Hz) și t durata lui (s)		
n2 = 3.5*106/8/f-30.125		
do	261.63Hz	fa# 369.99Hz
do#	277.18Hz	sol 392Hz
re	293.66Hz	sol# 415.30Hz
re#	311.13Hz	la 444Hz
mi	329.63Hz	la# 466.16Hz
fa	349.23Hz	si 493.88Hz
la fiecare salt la octava superioară/inferioară frecvențele se dublează/ înjumătățesc		
BLK	(— —adr)	U
variabilă-utilizator care conține numărul de bloc care se interpretează; dacă este 0 interpretarea se face de la consolă		
BLOCK	(n— —adr)	F
returnează adresa blocului n		
BORDER	(n— —)	m
setează culoarea BORDER conform n:		
0	negru	4 verde
1	albastru	5 cyan
2	rosu	6 galben
3	magenta	7 alb
BRANCH	(— —)	m, C
salt relativ conform numărului (cu semn, în complement față de 2) care urmează în definiția compilată a unui cuvînt; cuvînt compilat (în definiții) de AGAIN, ELSE, REPEAT		
BRIGHT	(n— —)	F
strălucirea următoarelor caractere afișate este accentuată dacă n=1 sau revine la normal dacă n=0		
BUFFER	(n— —adr)	F
revine cu adresa următorului buffer disponibil și îl asignează numărului de bloc n		
CI	(b adr— —)	m
memorează b la adr		
C,	(b— —)	F
memorează b în prima locație liberă din dicționar		
C@	(adr— —b)	m
citește b de la adr		
C/L	(— —n)	
n=64, numărul de caractere pe linie; constantă		
CASE	(n— —)	F, C, I
marcător de început al unei ramificații:		
CASE n1 OF listă1 ENDOF		
...		
n2 OF listă2 ENDOF		
listă ENDCASE		

dacă n corespunde cu una din valorile n1, ... nk se execută secvența
corespunzătoare, dacă nu, se execută secvența de dinainte de ENDCASE
care este opțională; în toate cazurile execuția continuă cu cuvintele de

	după ENDCASE	
CFA	(PFA— —CFA)	F
	returnează CFA al cuvîntului al cărui PFA se află pe SD	
CLS	(— —)	m
	șterge ecranul	
CMOVE	(adr1 adr2 n— —)	m
	mută n bytes de la adr1 la adr2, transferul se execută începînd cu byte-ul de la adr1	
COLD	(— —)	F
	inițializare totală a sistemului Forth	
COMPILE text	(— —)	F, C
	compilează CFA al cuvîntului <i>text</i> în următoarea intrare liberă din dicționar	
CONSTANT text	(n— —)	F/m
	crează o constantă căreia i se atribuie valoarea n; execuția ulterioară a <i>text</i> determină plasarea n pe SD	
CONTEXT	(—.—adr)	U
	variabilă-utilizator care conține NFA al ultimului cuvînt definit în vocabularul CONTEXT , în care are loc căutarea cuvintelor în mod interpretor	
COUNT	(adr— —adr+1 n)	F
	la adr se află un șir de caractere, în primul byte se află lungimea șirului, în următorii, codurile ASCII; COUNT revine cu parametrii utilizabili de TYPE	
CR	(— —)	m
	efectuează un retur de car pe ecran	
CREATE text	(— —)	F
	crează o intrare în dicționar care conține cîmpurile NFA, LFA, CFA; în CFA este introdusă adresa CFA+2 adică PFA; cîmpurile PFA nu sînt completate	
CSP	(— —adr)	U
	variabilă-utilizator care conține pointerul SD	
CURRENT	(— —adr)	U
	variabilă-utilizator care conține NFA al ultimului cuvînt definit în vocabularul CURRENT la care se adaugă definițiile compliate	
D+	(d1 d2— —d3)	m
	$d3 = d1 + d2$	
D+-	(d1 d2— —d3)	F
	aplică semnul lui d2 asupra lui d1 care devine d3	
D.	(d— —)	F
	afișează d conform BASE	
D.R	(d n— —)	F
	afișează numărul d conform BASE pe n spații, aliniat la dreapta	
DABS	(d1— —d2)	F
	returnează d2 modulul numărului d1	
DECIMAL	(— —)	F
	setează BASE la 10	
DEFINITIONS	(— —)	F
	declară vocabularul CONTEXT ca vocabular CURRENT	
DIGIT	(c n1— —n2 1)	m
	sau (c n1— —0)	
	convertește c în n2 conform bazei n1 și lasă un fanion pentru semnalizarea validării operației de conversie	
DLITERAL	(d— —) mod compilare	F, I
	(— —d) mod execuție	

compilează ca literali numerele care apar în definiții; la execuția definițiilor numerele sint plasate pe SD

DMINUS	(d1 — — d2)	m
	returnează d2 complement față de 2 a lui d1 (înmulțire cu -1)	
DO	SD (n1 n2 — —)	m
	SR (— — n2 n1)	
	marcator de început al buclei cu contor: DO listă LOOP sau DO listă +LOOP n1 reprezintă valoarea inițială a contorului n2 reprezintă valoarea finală a contorului	
DOES >	(— —)	F/m
	marcator de început al secvenței realizate la execuție în definiția cuvintelor de definiție: : <i>cuvînt-de-definiție listă1 DOES > listă2 ;</i>	
DP	(— — adr)	U
	variabilă-utilizator care conține pointerul curent al dicționarului	
DPL	(— — adr)	U
	variabilă-utilizator care conține deplasamentul punctului la introducerea numerelor; DPL > = 0 pentru număr cu punct, DPL < 0 pentru număr fără punct, valoare implicită: -1	
DRO	(— —)	F
	resetează la 0 variabila-utilizator OFFSET	
DRAW	(n1 n2 — —)	F
	trasează o dreaptă din ultima poziție PLOT sau DRAW anterioară, implicit colțul stînga jos al ecranului pînă în punctul de coordonate absolute n1 (de la stînga la dreapta, între 0 și 255) și n2 (de jos în sus, între 0 și 183); depășirile marginilor sint ignorate; CLS resetează poziția la starea implicită	
DROP	(n — —)	m
	șterge n de pe SD	
DUP	(n — — n n)	m
	duplică n pe SD	
EDITOR	(— —)	
	vocabularul EDITOR devine vocabular CONTEXT	
ELSE	(— —)	F, C, I
	marchează începutul ramurii de condiție falsă, opțională, din structura IF	
EMIT	(c — —)	m
	afișează caracterul c pe ecran la poziția curentă a cursorului	
EMPTY-BUFFERS	(— —)	F
	inițializează zona buffer-elor de disc dintre FIRST și LIMIT	
ENCLOSE	(adr1 c — — adr2 n1 n2 n3)	m
	cuvînt utilizat de WORD pentru interpretarea unui șir de cuvinte de la adr, cu delimitatorul c	
END	(— —)	F, C, I
	identic cu UNTIL	
ENDCASE	(— —)	F, C, I
	marchează sfîrșitul structurii CASE	
ENDIF	(— —)	F, C, I
	identic cu THEN	
ENDOF	(— —)	F, C, I
	marchează sfîrșitul unei ramuri de execuție în structura CASE	
ERASE	(adr n — —)	F
	șterge cu 0, n bytes începînd de la adr	

ERROR	(n1 — — n2 n3)	F
cuvînt utilizat de ?ERROR		
EXECUTE	(CFA — —)	m
execută cuvîntul al cărui CFA se află pe SD		
EXPECT	(adr n — —)	F
așteaptă introducerea a n caractere de la tastatură, ale căror coduri ASCII le depune în memorie începînd de la adr; șirul poate fi mai scurt dacă se termină cu Enter		
EXIT	(— —)	F
forțează terminarea prematură a cuvîntului în care se află		
FENCE	(— —adr)	U
variabilă-utilizator care conține adresa limită pînă la care se pot șterge cu FORGET definițiile din dicționar		
FILL	(adr n b — —)	m
inițializează cu b, n bytes începînd de la adr		
FIRST	(— —adr)	
constantă, conține adresa primului buffer de disc		
FLASH	(n — —)	F
următoarele caractere afișate sînt cliptoare dacă n=1; cu n=0 se revine la normal		
FLD	(— —adr)	U
variabilă-utilizator care conține pointerul zonei de conversie a numerelor date la afișare		
FLUSH	(— —)	F
rescrie toate buffer-ele de disc care au setate indicatorul de punere la zi cu UPDATE anterior		
FORGET text	(— —)	F
șterge toate definițiile ulterioare, inclusiv text, din dicționar; vocabularul CURRENT trebuie să fie același cu CONTEXT		
FORTH	(— —)	
reprezintă numele vocabularului de bază, în care se intră la inițializarea sistemului ca vocabular CONTEXT și CURRENT		
FREE	(— —n)	F
returnează numărul n de bytes liberi din dicționar		
GOVER	(n — —)	F
n=1 permite suprascrierea în ecran peste caracterele afișate anterior, n=0 revine la normal		
HERE	(— —adr)	F
returnează adresa primei celule libere din dicționar		
HEX	(— —)	F
setează BASE la 16		
HLD	(— —adr)	U
variabilă-utilizator care conține pointerul ultimului caracter introdus în PAD		
HOLD	(c — —)	F
inserează caracterul c în șirul unui număr formatat, se utilizează numai între <# și #>		
HI	(— —n)	
constantă, n=FBFF, ultima adresă a RAM-DISC-ului		
I	(— —n)	m, C
copiază pe SD vîrfurile stivei SR		
I'	(— —n)	m, C
copiază pe SD penultimul element de pe SR		

ID.	(NFA— —)	F
	afișează numele cuvîntului al cărui NFA se află pe SD	
IF	(f— —)	F, C, I
	marchează începutul ramurii de condiție adevărată din structurile: IF <i>listă</i> THEN sau IF <i>listă1</i> ELSE <i>listă2</i> THEN; în ambele cazuri execuția continuă cu cuvintele aflate după THEN	
IMMEDIATE	(— —)	F
	marchează drept imediat ultimul cuvînt definit; cuvintele imediate sînt executate în decursul compilării dacă execuția lor nu a fost inhibată cu [COMPILE]	
IN	(— —adr)	U
	variabilă-utilizator, conține un offset care este o adresă relativă a următorului text de interpretat de către WORD	
INCX	(— —adr)	
	variabilă în dublă lungime utilizată de DRAW	
INCY	(— —adr)	
	variabilă în dublă lungime utilizată de DRAW	
INDEX	(n1 n2— —)	F
	afișează liniile 0 ale ecranelor cu numerele între n1 și n2	
INIT-DISC	(— —)	F
	inițializează (șterge) RAM-DISC-ul	
INK	(n— —)	F
	setează culoarea INK conform n: 0 negru 4 verde 1 albastru 5 cyan 2 roșu 6 galben 3 magenta 7 alb	
INTERPRET	(— —)	F
	lansează interpretarea șirului de intrare din blocul BLK (BLK=0: interpretarea se face din TIB)	
INKEY	(— —c)	m
	returnează codul ASCII al tastei apăsată (fără a aștepta apăsarea tastei); returnează 255 dacă nu a fost apăsată nici o tastă	
INP	(u1 — —u2)	m
	citește portul u1 și returnează valoarea u2 citită; se realizează instrucțiunile: LD BC, u1 ; IN A, (C); conținutul A este depus pe SD	
INVERSE	(n— —)	F
	următoarele caractere afișate sînt INVERSE dacă n=1; cu n=0 se revine la normal	
J	(— —n)	F, C
	copiază antepenultima valoare de pe SR pe SD	
KEY	(— —c)	m
	așteaptă apăsarea unei taste și returnează codul ASCII al tastei apăsată	
LATEST	(— —NFA)	F
	returnează NFA al ultimei definiții din vocabularul CURRENT	
LEAVE	(— —)	m, C
	forțează terminarea buclelor cu contor: DO ... LOOP/+LOOP la următoarea execuție a cuvîntului LOOP/+LOOP (prin egalarea valorii curente a contorului cu valoarea finală)	
LFA	(PFA— —LFA)	F
	returnează adresa LFA a cuvîntului a cărui adresă PFA este pe SD	
LINE	(n— —)	F

	returnează adresa liniei n din ecranul curent	
LINK	(n— —)	m
	n=1 cuplează imprimanta, n=0 decuplează imprimanta	
LIMIT	(— —n)	
	constantă, conține adresa ultimei locații de memorie + 1	
LIST	(n— —)	F
	listează ecranul n care devine ecran curent	
LIT	(— —)	m, C
	cuvânt compilat de LITERAL, în fața unui număr, în dicționar; în mod execuție LIT plasează pe SD numărul care urmează (după LIT) în definiția compilată	
LITERAL	(n— —) mod compilare (— —n) mod execuție	F, C
	compilează numerele care apar în definiții ca literali; la execuție aceste numere sint plasate pe SD	
LO	(— —n)	
	constantă, n=D000 prima adresă a RAM-DISC-ului	
LOAD	(n— —)	F
	lansează interpretarea din ecranul n, inițializează BLK, IN	
LOADT	(— —)	F
	încarcă de pe bandă magnetică un fișier DISC în RAM-DISC	
LOOP	(— —)	F
	marchează sfârșitul buclei cu contor: DO /listă LOOP ; incrementează contorul buclei cu 1 și iese din buclă dacă contorul curent este egal sau mai mare ca valoarea limită	
M*	(n1 n2— —d)	F
	d=n1*n2	
M/	(d n1— —n2 n3)	F
	n3=d/n1, n2=restul împărțirii d/n1	
M/MOD	(ud1 u1— —u2 ud2)	F
	ud2=ud1/u1, u2=restul împărțirii ud1/u1	
MAX	(n1 n2— —n3)	F
	n3 maxim dintre n1 și n2	
MESSAGE	(n— —)	F
	utilizat de ?ERROR, afișează locul erorii și mesajul MSG# n, n fiind codul erorii	
MIN	(n1 n2— —n3)	F
	n3=minimum dintre n1 și n2	
MINUS	(n1— —n2)	m
	returnează n2 complement față de 2 a lui n1 (înmulțire cu -1)	
MOD	(n1 n2— —n3)	F
	n3=restul împărțirii n1/n2	
MON	(— —)	m
	reîntoarcere în BASIC Spectrum (cu mesajul de eroare 9: STOP statement 2:1); relansarea Forth se face GO TO 2 (COLD) sau GO TO 3 (WARM)	
NEXT	(— —n)	
	constantă, n=5E6C adresa de reîntoarcere dintr-o secvență în cod-mașină (adresa interpretorului de adrese)	
NFA	(PFA— —NFA)	F
	revine cu adresa NFA a cuvântului a cărui adresa PFA se află pe SD	
NOOP	(— —)	F
	instrucțiune vidă	

NOT	(f1 — —f2)	F
	inversează fanionul logic f1, returnînd f2; identic cu 0=	
NUMBER	(adr — —d)	F
	convertește un șir de caractere, terminate cu blank, de la adr, în număr pe dublă lungime conform BASE; emite un mesaj de eroare dacă operația nu este posibilă	
OF	(— —)	F, C, I
	cuvînt utilizat în structura CASE, marchează începutul unei ramuri de prelucrare	
OFFSET	(— —adr)	U
	variabilă-utilizator care conține un offset pentru calculul adresei fizice a unui bloc	
OR	(n1 n2 — —n3)	m
	efectuează sau-logic pe biți între n1 și n2 returnînd rezultatul n3	
OUT	(— —adr)	U
	variabilă-utilizator utilizată de EMIT	
OUTP	(u1 u2 — —)	m
	trimite la portul u2 byte-ul u1; se realizează instrucțiunile: LD BC, u2; LD A, u1; OUT (C), A	
OVER	(n1 n2 — —n1 n2 n1)	m
	copiază în vârful SD penultimul element	
PAD	(— —adr)	F
	returnează adresa primei locații PAD (HERE + 70)	
PAPER	(n — —)	F
	setează culoarea PAPER conform n:	
	0 negru 4 verde	
	1 albastru 5 cyan	
	2 roșu 6 galben	
	3 magenta 7 alb	
PLOT	(n1 n2 — —)	m
	afișează un punct de coordonate:	
	n1, între 0 și 255, de la stînga la dreapta	
	n2, între 0 și 183, de jos în sus	
	depășirile marginilor sînt ignorate	
PFA	(NFA — —PFA)	F
	revine cu adresa PFA a cuvîntului a cărui adresa NFA se află pe SD	
POINT	(n1 n2 — —f)	m
	returnează f = 1 pentru pixel aprins (de culoare INK) unde n1, n2 coordonatele pixel-ului:	
	n1, între 0 și 255, de la stînga la dreapta	
	n2, între 0 și 183, de jos în sus	
	depășirile marginilor sînt ignorate	
PREV	(— —adr)	U
	variabilă-utilizator utilizată de +BUF și UPDATE	
PUSHDE	(— —n)	
	constantă, n = 5E6A, adresa de reîntoarcere dintr-o secvență în cod-mașină (adresa interpretorului de adrese); regiștrii DE și HL sînt salvați pe SD (HL pe vârful stivei SD)	
PUSHHL	(— —n)	
	constantă, n = 5E6B, adresa de reîntoarcere dintr-o secvență în cod-mașină (adresa interpretorului de adrese); registrul HL este salvat pe SD	
QUERY	(— —)	F

QUIT	așteaptă maxim 80 de caractere și le depune în TIB, resetează IN (— —) F
R	bucla sistemului de operare, inițializează SD și SR și trece în mod execuție (interpretor) (— —n) F, C
R#	identic cu I (— —adr) U
R/W	variabilă-utilizator utilizată de cuvintele din EDITOR (adr n f— —) F
R >	procedură de scriere/citire pe RAM-DISC: adr adresa buffer-ului de bloc n numărul blocului f=0/1 pentru scris/citit SD (— —n) m, C SR (n— —)
R0	transferă vârful SR pe SD; se utilizează în pereche cu R (— —adr) U
REPEAT	variabilă-utilizator care conține adresa bazei SR (— —) F, C, I
ROT	marcător de sfârșit al buclei: BEGIN listă1 WHILE listă2 REPEAT (n1 n2 n3— —n2 n3 n1) m
RPI	permută circular ultimele 3 elemente din SD (— —) m
RP@	inițializează pointerul SR cu R0 (resetează SR) (— —adr) m
S->D	returnează pointerul curent al SR (n— —d) m
S0	transformă un număr în echivalentul său în dublă lungime (— —adr) U
SAVET	variabilă-utilizator care conține adresa bazei SD (— —) F
SCR	salvează pe bandă magnetică RAM-DISC-ul într-un fișier cu numele DISC (— —adr) U
SCREEN	variabilă-utilizator care conține numărul ecranului curent, actualizată de LIST (n1 n2— —c) m
SIGN	returnează codul ASCII al caracterului afișat în poziția: n1, linia, între 0 și 21, de sus în jos n2, coloană, între 0 și 31, de la stânga la dreapta (n d1— —d2) F
SIZE	inserează semnul -, dacă este cazul, în șirul unui număr formatat, se utilizează numai între <# și #> (— —n) F
SDMUDGE	returnează dimensiunea dicționarului (împreună cu interpretorul de adrese) (— —) F
SPI	complementează bitul de validare al ultimei definiții create (— —) m
SP@	inițializează pointerul SD cu S0 (resetează SD) (— —adr) m
SPACE	returnează pointerul curent al SD (de dinaintea execuției cuvântului) (— —) F

	afișează un blank pe ecran (în poziția curentă)	
SPACES	(n— —)	F
	afișează n blank-uri pe ecran (din poziția curentă)	
STATE	(— —adr)	U
	variabilă-utilizator care conține starea Forth: n=0/C0 mod interpretor/compiler	
SWAP	(n1 n2 — —n2 n1)	m
	permută ultimele 2 valori din virful SD	
TEXT	(c— —)	F
	transferă un text, delimitat de c, din TIB în PAD	
THEN	(— —)	F, C, I
	marcător de sfârșit al structurilor IF ... THEN sau IF ... ELSE ... THEN	
TIB	(— —adr)	U
	variabilă-utilizator, conține adresa buffer-ului de intrare de la tastatură	
TOGGLE	(adr n— —)	m
	realizează sau-exclusiv între byte-ul de la adr și byte-ul mai puțin semnificativ din n, rezultatul este depus la adr	
TRAVERSE	(adr1 n— —adr2)	F
	caută capătul opus al numelui unui cuvânt Forth din dicționar: n=1/-1 căutarea se face către adrese superioare/inferioare adr1 adresa unui capăt (NFA sau ultima litera din nume) adr2 adresa celui alt capăt	
TRIAD	(n— —)	F
	listează 3 ecrane: k, k+1, k+2 unde k este un număr divizibil cu 3 și n este egal cu k, k+1 sau k+2	
TYPE	(adr n— —)	F
	afișează un șir de n caractere aflat în memorie la adr	
U*	(u1 u2— —u3)	m
	u3=u1*u2	
U<	(u1 u2— —f)	F
	f=1 dacă u1 < u2, altfel f=0	
U/MOD	(ud u1— —u2 u3)	m
	u2=ud/u1, u3=restul împărțirii ud/u1	
U.	(u— —)	F
	afișează u conform BASE	
U.R	(u n— —)	F
	afișează u conform BASE pe n spații	
UDG	(— —adr)	F
	returnează adr zonei UDG	
UNTIL	(f— —)	F, C, I
	marcător de sfârșit de buclă: BEGIN /listă UNTIL ; reia bucla de la BEGIN dacă f < > 0 altfel continuă execuția după UNTIL	
UPDATE	(— —)	F
	setează Indicatorul de punere la zi al buffer-elor de disc care au fost modificate	
USE	(— —adr)	
	variabilă, adresa primului buffer de disc disponibil	
USER text	(n— —)	F/m
	crează o variabilă-utilizator cu numele text, n este adresa variabilei, offset față de adresa de început a tabelii (5E00); execuția text plasează pe SD adresa (absolută) a variabilei	
VARIABLE text	(n— —)	F/m

		crează o variabilă cu numele <i>text</i> căreia îi atribuie valoarea inițială <i>n</i> ; execuția <i>text</i> plasează pe SD adresa (PFA) a variabilei	
VERIFY	(— —)	verifică corectitudinea salvării pe bandă magnetică a fișierului DISC care conține RAM-DISC-ul	F
VOC-LINK	(— —adr)	variabilă-utilizator care conține adresa PFA+6 a ultimului vocabular creat (utilizată pentru înlănțuirea vocabularelor)	U
VOCABULARY	<i>text</i> (—)	crează un nou vocabular cu numele <i>text</i> , vocabularele trebuie declarate ca fiind de tip imediat; vocabularul nou este creat în vocabularul CURRENT; execuția <i>text</i> determină ca acesta să devină vocabular CONTEXT; există 2 vocabulare predefinite: FORTH (de bază) și EDITOR	F
VLIST	(—)	listează toate cuvintele definite (începînd cu ultimul) din vocabularul CONTEXT	F
WARM	(—)	inițializarea sistemului Forth: șterge buffer-ele de disc, resetează SD și SR; vocabularul CONTEXT devine FORTH și intră în QUIT	F
WARNING	(—adr)	variabilă-utilizator care conține indicatorul modulului de scriere a erorii	U
WHERE	(n1 n2—)	afișează locul erorii la încărcarea dintr-un ecran, parametrii sînt cei lăsați de ERROR	F
WHILE	(f—)	utilizat în bucla: BEGIN //stă1 WHILE //stă2 REPEAT dacă f=1 continuă execuția buclei dacă f=0 se lese imediat din buclă (execuția continuă cu primul cuvînt de după REPEAT)	F, C, I
WIDTH	(—adr)	variabilă-utilizator, conține lungimea maximă a unui cuvînt Forth (31 caractere)	U
WORD	(c—)	separă următorul șir de caractere delimitat de c din BLK și îl introduce în HERE	F
X1	(—adr)	variabilă în dublă lungime, utilizată de DRAW	
XOR	(n1 n2—n3)	efectuează sau-exclusiv pe biți între n1 și n2, returnează rezultatul n3	F
Y1	(—adr)	variabilă în dublă lungime, utilizată de DRAW	
[(—)	trece temporar în mod execuție, cuvintele dintre [și] vor fi executate, nu compilate, se utilizează în pereche cu]	F,C,I
[COMPILE]	<i>text</i> (—)	compilează cuvîntul imediat <i>text</i> în dicționar, nu îl execută	F, C, I
]	(—)	trece înapoi în mod compilare, se utilizează în pereche cu [F,C

6.6 Codurile erorilor

- 0 şirul nu poate fi identificat ca cuvînt (cuvînt inexistent în vocabularul **CONTEXT** sau ierarhic superioare)
- 1 stivă vidă
- 4 definiție dublă
- 6 număr de bloc eronat
- 7 debordare stivă și/sau dictionar prea mare (ocupă aceeași zonă de memorie)
- 9 ecranul 0 este considerat ca fiind atribuit consolei
- 17 cuvînt utilizat doar în mod compilare
- 18 cuvînt utilizat doar în mod execuție
- 19 structuri de control **BEGIN, CASE, DO, IF** sau **REPEAT** incorect definite
- 20 definiție terminată incorect
- 21 definiție protejată la ștergere
- 22 --> utilizat numai în timpul încărcării **LOAD**
- 23 numărul de linii dintr-un ecran este maxim 15
- 24 vocabularul **CONTEXT** este diferit de vocabularul **CURRENT**

7.1 Instalarea*

Prolog diferă în mod esențial de toate celelalte limbaje de programare, cu excepția Lisp, fiind un limbaj declarativ (nu imperativ). După încărcarea cu **LOAD** "" programul intră în execuție în SUPERVIZOR, care este bucla sistemului de operare.

SUPERVIZOR-ul așteaptă comenzi pe care le execută direct sau "date" de adăugat la baza de date. Toate comenzile necesită un argument (care pentru unele primitive poate fi orice) și numai unul (prin folosirea anumitor primitive se poate forța citirea în continuare a buffer-ului de intrare și defini comenzi cu mai multe argumente).

Taste funcționale:

Enter	redă controlul SUPERVIZOR-ului pentru interpretarea buffer-ului de intrare	
Break	(Symbol Shift & Space)	întrerupe execuția programelor
Cursor ->	(Caps Shift & 8)	folosite pentru editarea buffer-ului de intrare
Cursor <-	(Caps Shift & 5)	
SS&A	(Symbol Shift & A)	întrerupe temporar execuția unei comenzi
Delete	(Caps Shift & 0)	șterge caracterul din stînga cursorului

Se pot utiliza cursoarele L, C, G și E pentru accesarea literelor mari, mici, cifrelor și simbolurilor grafice diverse.

Prompt-ul sistemului este compus din 2 părți:

prompt-ul pus de SUPERVIZOR

- & pentru spațiul de lucru
- *numele-modulului* pentru module
- *cifră* care reprezintă numărul de paranteze rotunde rămase deschise din buffer-ul de intrare introdus anterior

prompt-ul . pus de rutina de citire de la tastatură

*Micro-PROLOG este produs înregistrat al firmei Logic Programming Associates Ltd.

este reluat de fiecare dată cînd o comandă sau dată (de adăugat la baza de date) nu a fost terminată după Enter.

Buffer-ul de intrare nu este golit după citirea termenilor astfel încît pot fi introduse mai multe comenzi pe linie.

Pentru editarea programelor se utilizează primitivele ADDCL, CL, DELCL și RFILL (sau editoare construite pe baza acestor primitive)

7.2 Structuri de limbaj

7.2.1 Termeni

Constante

Sînt șiruri de caractere de lungime maximă 60 caractere. Se face distincția între literele mari și mici. Sublinierea – și minus - sînt considerate litere. Cu excepția literelor și cifrelor celelalte caractere sînt considerate separatori. Dacă o constantă conține separatori, atunci aceasta se introduce între ghilimele: "constantă "; utilizarea unei constante care poate fi confundată cu un număr impune utilizarea ghilimelelor. Ghilimelele introduse într-o constantă sînt precedate de @. Caracterele neafișabile (de control) sînt formate din @ urmat de un caracter al cărui cod ASCII este mai mare cu 64 decît codul caracterului neafișabil

Variabile

Literele X, Y, Z, x, y, z urmate (imediat) sau nu de un număr-index sînt variabile. În general variabilele pot înlocui orice: termeni, liste (întregi sau fragmente), nume de predicate, atomi, clauze (cu unele restricții la unele predicate care acceptă ca variabile asignate termeni de un tip specificat). Numai ultimele 2 cifre ale numărului-index sînt semnificative. Într-o clauză pot exista maxim 64 de variabile. Variabilele sînt locale și numele lor pot fi modificate de Prolog dar cu 2 restricții: variabilele cu nume diferite rămîn cu nume diferite; numele este modificat la fel în toate pozițiile în care apare variabila

Numere

Întreg pozitiv: o secvență de cifre zecimale fără semnul +

Întreg negativ: o secvență de cifre zecimale precedată de semnul -

Număr pozitiv în virgulă flotantă: o secvență de cifre zecimale fără semnul + care conține un punct zecimal . ; opțional poate fi urmat de un exponent care este un număr întreg precedat de E sau e. Punctul zecimal trebuie să fie precedat de cel puțin o cifră.

Număr negativ în virgulă flotantă: are același format ca numărul pozitiv în virgulă flotantă, fiind precedat de -

Numerele în virgulă flotantă pot fi introduse în orice formă, dar sînt afișate într-o formă standard (mantisa între -10 și 10):

cifră . cifră ... cifră E cifră cifră

dacă exponentul este 0 atunci acesta nu se mai afișează.

Numerele întregi sînt în domeniul -32767, 32767.
Exponenții trebuie să fie în domeniul -127, 127.

7.2.2 Liste

Lista reprezintă un șir de termeni încadrați de paranteze rotunde:
(*termen ... termen*)

Listele pot fi imbricate una în alta nerestrictiv.

Are sens lista vidă: ()

Caracterul | separă capul listei de o lungime și/sau structură determinată de coada listei care poate avea o lungime/structură variabilă. Capul listei trebuie să conțină cel puțin un termen (eventual lista vidă).

7.2.3 Atomi

Atomul este o listă de termeni din care primul este numele unui predicat logic, iar următorii sînt argumentele sale:

(*nume-predicat termen-1 ... termen-n*)

Predicatele reprezintă o relație logică între termenii săi.

Rezultatul evaluării unui atom este o valoare logică: adevărat sau fals.

Predicatele predefinite în Prolog se numesc primitive.

Pot exista predicate care nu au nici un argument sau cu număr variabil de argumente.

7.2.4 Clauze

Clauzele reprezintă liste de atomi cu ajutorul cărora se definesc predicatele. Primul atom din clauză cuprinde predicatul de definit. Un predicat poate fi definit prin mai multe clauze, ca alternative.

((*nume-predicat-de-definit termen-1 ... termen-n*)

(*predicat termen ... termen*)

...
(*predicat termen ... termen*))

La evaluare primul atom returnează adevărat dacă toți ceilalți returnează adevărat la evaluarea în ordine.

Dacă o clauză (la evaluare) returnează fals se încearcă evaluarea clauzelor următoare (dacă nu este ultima).

În baza de date nu contează ordinea clauzelor referitoare la predicate diferite, dar pentru un același predicat de definit clauzele sînt ordonate conform introducerii lor de la tastatură (ordinea contează pentru clauze definind același predicat).

Predicatele pot fi definite (în clauze) în mod recursiv nerrestrictiv.

O clauză poate conține numai atomul de definit:

((*predicat-de-definit termen-1 ... termen-n*))

caz în care acesta este întotdeauna adevărat.

7.2.5 Comenzi

Introducerea unei linii de tipul:

nume-de-predicat-unar argument

produce tentativa de validare (prin backtraking) a atomului:

(*nume-de-predicat-unar argument*)

unde *argumentul* poate fi un termen sau o listă.

Dacă atomul returnează adevărat se afișează prompt-ul (care semnalizează așteptarea introducerii unei noi comenzi sau clauze).

Dacă atomul returnează fals se afișează ? apoi prompt-ul.

7.2.6 Module

Modulele sînt colecții închise de definiții (clauze) care nu pot fi modificate din exterior (spațiul de lucru sau alte module) și a căror utilizare (cu excepția celor specificate explicit) este transparentă în exterior (modulele sînt analogul subrutinelor din alte limbaje de programare) constantele fiind locale modulului (cu excepția celor specificate explicit).

Modulul este caracterizat de:

- numele modulului: constantă diferită de numele predicatelor utilizate și/sau numele fișierelor utilizate
- lista-export: cuprinde numele predicatelor definite în modul; care pot fi utilizate în alte module și/sau spațiul de lucru
- lista-import: cuprinde numele predicatelor definite în alte module și/sau spațiul de lucru și utilizate de clauzele din modul

Toate constantele nespecificate în cele 2 liste sînt locale și pot fi utilizate cu semnificație diferită în alte module și/sau spațiul de lucru (fiecare modul posedă un dicționar propriu).

Primitivele Prolog sînt definite într-un modul distinct, inaccesibil utilizatorului (definițiile lor nu pot fi modificate) și numele lor sînt exportate implicit în toate modulele și în spațiul de lucru.

7.3 Modul de lucru

Pentru validarea unui atom se încearcă toate variantele care ar putea fi soluții, prin metoda backtraking (căutare înapoi prin reevaluare).

Atomul de validat este comparat cu clauzele de definire ale predicatului corespunzător, din baza de date, în ordine.

Variabilelor nu li se atribuie valori decât dacă este strict necesar, o variabilă putînd înlocui orice.

O variabilă neasignată se potrivește cu alta variabilă neasignată, caz în care devin legate și asignarea unei variabile implică asignarea tuturor variabilelor legate de ea.

În cazul în care pentru atomul de validat este găsită o clauză pentru care se potrivește cu primul atom (din clauză) Prolog memorează un pointer către următoarea clauză (pentru predicatul respectiv) într-o stivă și încearcă validarea atomilor care compun clauza. Dacă validarea este reușită (succesiv, după aceeași metodă) Prolog semnalizează acest lucru, terminînd programul (apare prompt-ul curent). Dacă validarea este nereușită atunci se descarcă pointer-ul din stivă și este încercată clauza următoare. Dacă validarea eșuează (atomul returnează fals) Prolog semnalizează acest lucru afișînd ? (urmat de prompt-ul curent).

Stiva nu se încarcă în 3 situații:

- clauza predicatului definit este ultima din baza de date
- clauza specifică în mod explicit că este ultima/singura care poate fi încercată pentru validarea atomului
- atomul specifică în mod explicit (sau implicit pentru unele primitive) că admite soluție unică

7.4 Primitive Prolog

ABORT

ABORT

(atom)

ABORT *termen*

(comandă)

Produce abandonarea tuturor evaluărilor (resetează stiva); *termen* poate fi orice.

ADDCL

(ADDCL *clauză nr*)

(a)

(atom)

(ADDCL *clauză*)

(b)

(atom)

ADDCL *clauză*

(c)

(comandă)

unde:

- (a) adaugă clauza în baza de date după clauza cu numărul de ordine *nr*
- (b),(c) adaugă clauza la sfîrșitul bazei de date; returnează adevărat; *clauza*, *nr* pot fi înlocuite cu variabile care trebuie să fie asignate în

momentul evaluării

ALL

ALL

Nume pentru baza de date din spațiul de lucru/modulul curent, utilizat de LIST și KILL.

BORDER

(**BORDER nr**)

(atom)

BORDER nr

(comandă)

Setează culoarea BORDER conform codului culorii nr:

0 negru	4 verde
1 albastru	5 cyan
2 roșu	6 galben
3 magenta	7 alb

Returnează adevărat, numărul este prelucrat modulo 8 și poate fi înlocuit cu o variabilă care trebuie să fie asignată în momentul evaluării.

BP

(**BP nr1 nr2**)

(atom)

Emite un semnal sonor la difuzor de frecvență f (în Hz) și durată t (în s) unde:

$nr1 = f * t$; $nr2 = 3.5E6 / f / 8 - 30.125$

do	261.63Hz	fa#	369.99Hz
do#	277.18Hz	sol	392Hz
re	293.66Hz	sol#	415.30Hz
re#	311.13Hz	la	444Hz
mi	329.63Hz	la#	466.16Hz
fa	349.23Hz	si	493.88Hz

La fiecare salt la octava superioară/inferioară frecvențele se dublează/înjumătățesc.

Returnează adevărat, numerele pot fi înlocuite cu variabile care trebuie să fie asignate în momentul evaluării.

CHAROF

(**CHAROF caracter nr**)

(atom)

Returnează adevărat dacă caracter are codul ASCII nr, termenii pot fi înlocuiți cu variabile din care cel puțin una, oricare, trebuie să fie asignată în momentul evaluării, cealaltă, dacă este neasignată, capătă valoarea corespunzătoare.

CL

(**CL clauză**)

(a)

(atom)

(**CL clauză nr1 nr2**)

(b)

(atom)

CL clauză

(c)

(comandă)

Returnează adevărat dacă:

- (a), (c) clauza există în baza de date
 (b) clauza există în baza de date și are numărul de ordine nr2, $nr2 > = nr1$, nr1 numărul de ordine de unde începe căutarea

Predicatul care se definește în clauză și nr1 trebuie să fie cunoscute în momentul evaluării (pot fi variabile dar asignate în momentul evaluării).

CLMOD

CLMOD

(atom)

CLMOD termen

(comandă)

Închide modulul curent revenind în spațiul de lucru, termen poate fi orice; returnează adevărat; spațiul de lucru nu poate fi închis.

CLOSE

(CLOSE nume-fișier)

(atom)

CLOSE nume-fișier

(comandă)

Închide fișierul *nume-fișier* deschis cu **OPEN** sau **CREATE**; se utilizează **CLOSE** "" dacă s-a utilizat **OPEN** "". Returnează adevărat dacă *nume-fișier* coincide cu numele fișierului deschis și poate fi închis.

CLS**(CLS nr)**

(atom)

CLS nr

(comandă)

unde *nr* este număr prelucrat modulo 32 și semnifică:

*cod-culoare-PAPER + 8*cod-BRIGHT + 16*cod-FLASH*

unde:

*cod-culoare-PAPER**cod-BRIGHT**cod-FLASH*

0 negru

4 verde

0 normal

0 normal

1 albastru

5 cyan

1 strălucitor

1 strălucitor

2 roșu

6 galben

3 magenta

7 alb

CLS șterge ecranul și setează culorile conform *nr*. Returnează adevărat, numărul poate fi înlocuit cu o variabilă care trebuie să fie asignată în momentul evaluării.

CMOD**(CMOD X)**

(atom)

CMOD X

(comandă)

Returnează adevărat dacă *X* este variabilă neasignată și simplă (nestructurată). Variabila este asignată cu numele modulului curent (& pentru spațiul de lucru).

CON**(CON termen)**

(atom)

CON termen

(comandă)

Returnează adevărat dacă termen este o constantă sau o variabilă asignată cu o constantă în momentul evaluării.

"CON:"**"CON:"**

Numele fișierului consolă deschis permanent pentru citit (tastatură) și scris (ecran). Tentativele **CREATE**, **OPEN** și **CLOSE** pentru acest fișier nu generează fals sau eroare dar nu au nici o acțiune.

CREATE**(CREATE nume-fișier)**

(atom)

CREATE nume-fișier

(comandă)

Creează un fișier pe bandă magnetică deschis pentru scris. Returnează adevărat dacă nu există alte fișiere deschise la momentul evaluării, din cele care pot fi închise. Numele fișierului este o constantă din 1-8 caractere. Scrierea efectivă (salvarea pe bandă) are loc în momentul închiderii fișierului.

CRMOD**(CRMOD nume-modul listă-export listă-import)**

(atom)

Creează un modul numit nume-modul unde:

nume-modul trebuie să fie o constantă (variabilă asignată) diferit de numele altor module, predicate definite și constante utilizate în modulul respectiv sau în spațiul de lucru

listă-export cuprinde numele predicatelor exportate de modul, acestea vor fi definite ulterior în cadrul modulului

listă-import cuprinde numele predicatelor importate în modul din spațiul de lucru sau alte module; predicatele importate pot fi listate dar nu pot fi șterse

decît în modulul/spațiul de lucru unde au fost definite; predicatelor importate trebuie să fie definite; primitivile Prolog sînt implicit exportate în toate modulele și în spațiul de lucru

Returnează adevărat dacă condițiile de creare a modulului sînt respectate. Utilizatorul este lăsat în modulul respectiv după crearea acestuia dacă nu mai urmează nici un atom/clauză de evaluat.

DELCL

(DELCL *clauză*)

(a)(atom)

DELCL *clauză*

(b)(comandă)

(DELCL *predicat nr*)

(c)(atom)

Șterge clauza respectivă din baza de date:

(a),(b) șterge *clauza* , aceasta poate conține variabile neasignate dar trebuie să fie în mod univoc determinată

(c) șterge clauza cu numărul de ordine *nr* referitoare la *predicat* (*predicat* și *nr* pot fi variabile dar asignate)

Returnează adevărat dacă condițiile de ștergere a clauzei sînt îndeplinite.

DICT

(DICT *nume-modul listă-export listă-import listă-constante*)

(atom)

Reprezintă atomul de stare al spațiului de lucru/modulului curenț

Pentru module:

nume-modul

numele modulului

listă-export

lista predicatelor exportate de modul

listă-import

lista predicatelor importate de modul

listă-constante

lista predicatelor definite în modul

Pentru spațiul de lucru:

nume-modul

&

listă-export

(), spațiul de lucru nu exportă nimic

listă-import

lista predicatelor exportate de toate celelalte module

listă-constante

lista predicatelor definite în spațiul de lucru și predicatul

“?ERROR?”de mascare a erorilor

EQ

(EQ *termen1 termen2*)

(atom)

Returnează adevărat dacă *termen1* este identic cu *termen2* (listele sau variabilele au aceeași structură). Variabilele devin asignate sau legate.

FAIL

FAIL

(atom)

FAIL *orice*

(comanda)

Returnează întotdeauna fals.

FORALL

(FORALL *termen1 termen2*)

(atom)

unde *termen1* și *termen2* sînt liste de atomi.

Returnează adevărat dacă pentru toate variabilele locale pentru care *termen1* este adevărat atunci și *termen2* este adevărat. Variabilele globale din *termen1* trebuie să fie asignate în momentul evaluării altfel se pot genera răspunsuri greșite (Prolog nu verifică îndeplinirea acestei condiții).

HYBRID

HYBRID

(atom)

HYBRID *orice*

(comandă)

Comută zona de dialog a ecranului pe ultimile 4 linii și șterge ecranul conform ultimei specificații CLS (eliberează zona superioară a ecranului); avînd acțiune inversă lui NORMAL. Returnează adevărat.

IF
 (IF termen1 termen2 termen3) (atom)
 unde termen1/ 2/ 3 sînt liste de atomi. Returnează adevărat dacă termen1 și termen2 sînt adevărați sau termen1 fals și termen3 adevărat.

INKEY
 (INKEY X) (atom)
 INKEY X (comanda)

Asignează X cu codul tastei apăsate, comută cursorul în L înainte de citire, caracterele nelterare sînt încadrate în ghilimele, cele neafișabile sînt precedate de @ urmat de un caracter al cărui cod ASCII este cu 64 mai mare decît al caracterului neafișabil. Nu așteaptă apăsarea unei taste, în cazul în care nu a fost apăsată nici o tastă X este asignat cu un șir vid "". X trebuie să fie variabilă neasignată în momentul evaluării. Returnează adevărat.

INT
 (INT nr) (atom)
 (INT nr X) (atom)
 INT nr (comanda)

unde:
 (a), (c) returnează adevărat dacă nr este un număr întreg în intervalul -32767, +32767 (nr poate fi o variabilă, asignată în momentul evaluării)
 (b) returnează adevărat dacă nr este în intervalul -32767, +32767 și variabilei X, neasignate în momentul evaluării, i se atribuie partea întreagă a nr

INTOK
 (INTOK nume-fișier X) (atom)

Citește un termen din fișierul nume-fișier care trebuie să fie deschis pentru citit și îl atribuie variabilei neasignate X. Termenul trebuie să fie o constantă sau un număr, inserează ghilimele pentru constantele care conțin separatori sau pot fi interpretate ca variabile. Returnează adevărat.

ISALL
 (ISALL termen1 termen2 termen3) (atom)

Returnează adevărat dacă termen1 este lista tuturor soluțiilor posibile de tipul termen2 care reprezintă o variabilă locală sau o listă de variabile locale și care verifică termen3 care este o listă de atomi. Variabilele globale din termeni trebuie să fie asignate în momentul evaluării altfel se pot genera răspunsuri greșite (Prolog nu verifică îndeplinirea acestei condiții). Termen1 reprezintă lista soluțiilor în ordinea inversă găsirii lor.

KILL
 (KILL nume) (atom)
 KILL nume (comandă)

Dacă nume este un nume de predicat atunci șterge toate clauzele de definiție din toate modulele sau spațiul de lucru referitoare la predicat. Dacă nume este nume de modul atunci șterge modulul. Dacă nume este ALL șterge toate clauzele din spațiul de lucru/modulul curent. Un modul nu poate fi șters din interiorul său. Nume poate fi o variabilă asignată în momentul evaluării. Returnează adevărat.

LESS
 (LESS termen1 termen2) (atom)

Returnează adevărat dacă *termen1* este strict mai mic decât *termen2* conform ordinii lexicografice pentru constante (numerele sînt mai mici decât literele majuscule care sînt mai mici decât literele minuscule) sau ordinii normale pentru numere (constantele care pot fi confundate cu numere trebuie încadrate în ghilimele. Variabilele trebuie asignate la momentul evaluării.

LIST

(LIST *nume*) (atom)
LIST *nume* (comandă)

Scrie la consolă toate clauzele de definiție (eventual nici una) referitoare la predicatul *nume*, definite în spațiul de lucru/modulul curent sau importate de acesta. Dacă *nume* este ALL listează toate clauzele existente în spațiul de lucru/modulul curent. *Nume* poate fi înlocuit cu o variabilă asignată în momentul evaluării. Returnează adevărat cu excepția cazului în care *nume* este numele unei primitive Prolog care nu poate fi listată.

LISTP

(LISTP *nume-fișier* *nume*) (atom)
LISTP *nume* (comandă)

Scrie în fișierul *nume-fișier* toate clauzele de definiție (eventual nici una) referitoare la predicatul *nume*, definite în spațiul de lucru/modulul curent sau importate de acesta. Dacă *nume* lipsește (este opțional) se scriu toate clauzele existente în spațiul de lucru/modulul curent. *Nume* poate fi înlocuit cu o variabilă asignată în momentul evaluării. Returnează adevărat cu excepția cazului în care *nume* este numele unei primitive Prolog care nu poate fi listată.

LNE

(LNE *nr1 nr2 nr3 nr4 nr5 nr6*) (atom)

Trasează o linie între punctele de coordonate (*nr1*, *nr2*) și (*nr3*, *nr4*) unde:

nr1, *nr3* între -128 și +127, de la stînga la dreapta
nr2, *nr4* între -88 și +87, de jos în sus
nr5 este opțional și are valoarea (modulo 256):
cod-culoare-INK +8*cod-culoare-PAPER
+64*cod-BRIGHT +128*cod-FLASH
valoare implicită: cea anterioară
nr6 este opțional și are valoarea (modulo 2):
0 suprascriere (OVER); 1 normal
valoare implicită: 0

cod-culoare:	cod-BRIGHT	cod-FLASH
0 negru	4 verde	0 normal
1 albastru	5 cyan	1 strălucitor
2 roșu	6 galben	
3 magenta	7 alb	

Returnează adevărat dacă *nr5*, *nr6* sînt întregi între -32767 și 32767 (*nr1/2/3/4* sînt rotunjiți la partea întregă); dacă *nr1* = *nr3* și *nr2* = *nr4* nu se trasează nimic).

LOAD

(LOAD *nume*) (atom)
LOAD *nume* (comandă)

Citește un fișier *nume* de pe bandă magnetică, dacă *nume* este "" se citește primul fișier întîlnit. Fișierele sînt formate dintr-un număr de blocuri de lungime 260 bytes fiecare, dacă unul din blocuri s-a încărcat cu eroare se reia citirea de la acest bloc. Returnează adevărat.

LST

- (LST termen) (atom)
LST termen (comandă)
- Returnează adevărat dacă *termen* este de tip listă.
- "LST:"
- "LST:"
- nume de fișier-imprimantă deschis pentru scris.
- NEW**
- NEW** (atom)
NEW orice (comandă)
- Reinițializează total sistemul Prolog dar nu afectează culorile CLS și modul NORMAL/HYBRID anterioare.
- NORMAL**
- NORMAL** (atom)
NORMAL orice (comandă)
- Comută zona de dialog a ecranului pe primele 22 de linii și șterge ecranul conform ultimei specificații **CLS**, avînd acțiune inversă lui **HYBRYD**. Returnează adevărat (inițial **NORMAL**).
- NOT**
- (**NOT predicat termen-1 ... termen-n**) (atom)
NOT comandă (comandă)
- Dacă atomul (*predicat termen-1 ... termen-n*) este adevărat atunci atomul (**NOT predicat termen-1 ... termen-n**) returnează fals și reciproc. În regim de comandă **NOT** anulează comanda și returnează adevărat.
- NOT** poate fi utilizat doar pentru verificare, variabilele globale trebuie să fie asigurate în momentul evaluării. Înaintea evaluării **NOT**, este evaluat atomul care conține predicatul astfel încît dacă această evaluare implică o acțiune, ea va fi executată.
- NUM**
- (**NUM termen**) (atom)
NUM termen (comandă)
- Returnează adevărat dacă *termen* este un număr (sau o variabilă asigurată cu un număr).
- OPEN**
- (**OPEN nume-fișier**) (atom)
OPEN nume-fișier (comandă)
- Deschide un fișier pe bandă magnetică pentru citit. **OPEN ""** va citi primul fișier întîlnit. Returnează adevărat.
- OPMOD**
- (**OPMOD nume-modul**) (atom)
OPMOD nume-modul (comandă)
- Deschide modulul *nume-modul*. Returnează adevărat dacă modulul există și poate fi deschis (un modul nu poate fi deschis din el însuși). În mod comandă utilizatorul este lăsat în modulul deschis. În mod evaluare de clauză execuția continuă cu evaluarea atomului următor.
- OR**
- (**OR listă-atomi-1 listă-atomi-2**) (atom)
- Returnează adevărat dacă *lista-atomi-1* sau *lista-atomi-2* (sau amîndouă) sînt adevărate.
- P**

(P termen-1 ... termen-n)

(atom)

P termen

; comandă)

Scrie seria de termeni *termen-1 ... termen-n* la consolă, nu efectuează retur de car după scriere. Pentru variabilele asignate scrie valorile corespunzătoare, numele variabilelor neasignate pot fi modificate. Returnează adevărat.

PIO

(PIO nr termen)

(atom)

Citește sau scrie la portul *nr* (poate fi variabilă asignată), între -32767 și +32767 (întreg). *Nr* este convertit intern în număr fără semn, între 0 și 65535 (complement față de 2). Dacă *termen* este număr cunoscut (variabilă asignată) între -32767 și +32767 acesta este convertit intern în număr fără semn, între 0 și 65535 (complement față de 2) și byte-ul mai puțin semnificativ este trimis la portul specificat (se execută instrucțiunile LD BC, nr ; LD A, termen ; OUT (C), A). Dacă *termen* este variabilă neasignată citește de la port o valoare și atribuie variabilei numărul citit, între 0 și 255 (se execută instrucțiunile LD BC, nr ; IN (C), A; LD termen , A). Returnează adevărat dacă valorile date sînt în domeniul specificat și întregi.

PNT

(PNT nr1 nr2 nr3 nr4)

(atom)

Desenează un punct de coordonate (*nr1, nr2*) unde:

nr1 între -128 și +127, de la stînga la dreapta

nr2 între -88 și +87, de jos în sus

nr3 este opțional și are valoarea (modulo 256):

cod-culoare-INK + 8**cod-culoare-PAPER*
+ 64**cod-BRIGHT* + 128**cod-FLASH*

valoare implicită: cea anterioară

nr4 este opțional și are valoarea (modulo 2):

0 suprascriere (OVER); 1 normal
valoare implicită:0

cod-culoare:

cod-BRIGHT

cod-FLASH

0 negru

4 verde

0 normal

0 normal

1 albastru

5 cyan

1 strălucitor

1 strălucitor

2 roșu

6 galben

3 magenta

7 alb

Returnează adevărat dacă *nr3, nr4* sînt întregi între -32767 și 32767 (*nr1/2* sînt rotunjiți la partea întreagă).

PP

(PP termen-1 ... termen-n)

(atom)

PP termen

; comandă)

Scrie seria de termeni *termen-1 ... termen-n* la consolă, efectuează retur de car după scriere. Pentru variabilele asignate scrie valorile corespunzătoare, numele variabilelor neasignate pot fi modificate. Returnează adevărat.

R

(R X)

(atom)

R X

; comandă)

Citește termenul următor din buffer-ul de intrare (consolă) dacă acesta nu a fost citit complet sau de la tastatură și îl atribuie variabilei neasignate X în momentul evaluării (termenii pot fi orice). Returnează adevărat.

READ

(READ nume-fișier X)

(atom)

READ nume-fișier X

; comandă)

Citește termenul următor din fișierul *nume-fișier* și îi atribuie variabilei neasignate *X* în momentul evaluării (termenii pot fi orice). Returnează adevărat.

RFILL

(RFILL (*termen-1 ... termen-n*) *X*) (atom)

Introduce în editare (pe linie, cu ajutorul cursorului) lista de termeni. Variabila *X* neasignată la momentul evaluării capătă valoarea listei de termeni (modificată sau nu de utilizator) după apăsarea tastei Enter. Returnează adevărat.

RND

(RND *X nr*) (atom)

Numărul întreg *nr* între -32767 și 32767 (variabila asignată) este convertit intern într-un număr întreg fără semn, între 0 și 65535 (complement față de 2) și apoi se calculează un număr aleator între 0 și *nr* -1 cu care este asignată variabila *X* (la afișare acesta va fi considerat ca număr cu semn, în complement față de 2, între -32767 și +3276). Returnează adevărat dacă *X* este variabilă neasignată la momentul evaluării.

SAVE

(SAVE *nume-fișier nume-predicat*) (a)(atom)
 (SAVE *nume-fișier*) (b)(atom)
 SAVE *nume-fișier* (c)(comandă)

Scrie într-un fișier pe bandă cu numele *nume-fișier* :

- (a) toate clauzele referitoare la *nume-predicat*
 (b),(c) toate clauzele

Returnează adevărat.

SIGN

(SIGN *nr X*) (atom)

Atribuie variabilei neasignate *X* valoarea 1 dacă *nr* este număr pozitiv sau valoarea -1 dacă *nr* este număr negativ (*nr* poate fi variabilă asignată). Returnează adevărat.

SPACE

(SPACE *X*) (atom)

Atribue variabilei neasignate *X* numărul de kbytes care mai sînt disponibili în memorie pentru program (inițial 19). Returnează adevărat.

STRINGOF

(STRINGOF *listă constantă*) (atom)

Returnează adevărat dacă *listă* este lista tuturor caracterelor, în aceeași ordine, din care este formată *constantă*; *lista* și *constantă* pot fi variabile dintre care cel puțin una (oricare) trebuie să fie asignată în momentul evaluării.

SUM

(SUM *nr1 nr2 nr3*) (atom)

Returnează adevărat dacă numerele *nr1/2/3* sînt în relația: $nr1 + nr2 = nr3$. *nr1/2/3* pot fi variabile din care cel puțin 2 (oricare) trebuie să fie asignate în momentul evaluării.

SYS

(SYS *constantă*) (atom)
 SYS *constantă* (comandă)

Returnează adevărat dacă *constantă* (variabilă asignată) este numele unei primitive Prolog.

TIMES

(TIMES *nr1 nr2 nr3*) (atom)

Returnează adevărat dacă numerele $nr1/2/3$ sînt în relația: $nr1 * nr2 = nr3$. $nr1/2/3$ pot fi variabile din care cel puțin 2 (oricare) trebuie să fie asigurate în momentul evaluării.

VAR

(VAR X)

(atom)

VAR X

(comandă)

Returnează adevărat dacă X este o variabilă neasignată în momentul evaluării.

W

(W nume-fișier listă-termeni)

(atom)

Scrie în fișierul *nume-fișier*, deschis pentru scris, termenii din listă, nu inserează retur de car. Returnează adevărat (cu excepția cazului în care *listă-termeni* nu este listă).

WRITE

(WRITE nume-fișier listă-termeni)

(atom)

Scrie în fișierul *nume-fișier*, deschis pentru scris, termenii din listă, inserează retur de car. Returnează adevărat (cu excepția cazului în care *listă-termeni* nu este listă).

?

(? listă-atom)

(atom)

? listă-atom

(comandă)

Evaluează atomii din listă și returnează adevărat dacă toți atomii returnează adevărat (cu excepția cazului în care *listă-atom* nu este listă de atomi)

|

(| nume-predicat termen-1 ... termen-n)

(atom)

Atenționează Prolog că la evaluare atomul:

(nume-predicat termen-1 ...termen-n)

are soluție unică astfel încît stiva nu se mai încarcă pentru backtraking. Nu se poate utiliza în atomul de definit.

/

/

(atom)

Atenționează Prolog că clauza în care apare atomul / este ultima care poate fi folosită pentru găsirea soluțiilor, astfel încît stiva nu se mai încarcă pentru backtraking. Returnează adevărat.

/*

(/* comentariu)

(atom)

/* comentariu

(comandă)

Permite introducerea comentariilor în programe. Returnează adevărat.

"<>"

("<>" termen)

(atom)

"<>" termen

(comandă)

Reprezintă primitiva Prolog de interpretare a textului introdus la tastatură. Dacă *termen* este o comandă cu un singur parametru citește parametrul ei și o execută; dacă *termen* este o clauză o adaugă la baza de date. Returnează adevărat dacă termen este o comandă (executată corect) sau o clauză.

"<SUP>"

("<SUP>")

(atom)

Reprezintă programul SUPERVIZOR (bucla sistemului de operare).

"?ERROR?"

(("?ERROR?" nr X) atom ... atom)

(clauză)

"?ERROR?" permite mascarea erorilor de tipul *nr* (dacă există clauza de mascare), asignează variabila *X* cu atomul care a generat eroarea și continuă evaluarea cu lista de atomi din clauza de mascare a erorilor. Dacă toți aceștia returnează adevărat, evaluarea continuă cu atomul de după cel care a generat eroarea.

7.5 Codurile erorilor

- 0 subdepășire aritmetică (număr prea mic)
- 1 supradepășire aritmetică (număr prea mare)
- 2 predicat nedefinit
- 3 prea multe sau prea puține variabile
- 4 definiție protejată
- 5 eroare de fișier
- 6 fișier rămas neînchis
- 9 eroare de scriere
- 11 Break
- 12 utilizare incorectă a modulelor
- 13 linie sau punct în afara ecranului
- 15 Break în timpul scrierii/citirii pe bandă magnetică
- 22 culoare incorectă

Sumar

1 RUTINELE ROM	3
1.1 Rutinele restart	3
1.2 Rutinele de tastatură	4
1.3 Rutine difuzor	5
1.4 Rutine casetofon	6
1.5 Rutine ecran	7
1.6 Rutine de executie	10
1.7 Interpretorul BASIC	14
1.8 Evaluatorul de expresii	18
1.9 Rutine aritmetice	21
1.10 Calculatorul în virgulă flotantă	22
1.11 Variabilele de sistem	28
1.12 Harta memoriei totale	31
1.13 Structuri BASIC	32
1.14 Setul de caractere	33
2 ASAMBLORUL/DEZASAMBLORUL	34
2.1 Microprocesorul Z80	34
2.2 Încărcare pe 8 biți	35
2.3 Încărcare pe 16 biți	36
2.4 Schimb, transfer de blocuri și căutare	38
2.5 Aritmetică și logică pe 8 biți	39
2.6 Control	41
2.7 Aritmetică pe 16 biți	42
2.8 Rotație și shift	43
2.9 Set, reset și test pe biți	45
2.10 Transfer	46
2.11 Subrutine	47
2.12 Input și output	48
2.13 Tabelele de dezasamblare	50
2.13.1 Tabela 1 (principală), fără prefix	50
2.13.2 Tabela 2; prefix: CB	52
2.13.3 Tabela 3; prefix ED	54
2.13.4 Tabela 4; prefix DD/FD pentru registrul IX/IY	55
2.13.5 Tabele de conversie: binar/zecimal/hexazecimal	56
2.14 Asamblorul GENS	57
2.14.1 Instalarea	57
2.14.2 Editorul	57
2.14.3 Directive de asamblare	61
2.14.4 Codul erorilor generate	63
2.15 Dezasamblorul MONS	63
2.15.1 Instalarea	63
2.15.2 Comenzi	64
3 BASIC	68
3.1 BASIC Spectrum	68

3.1.1	Constante, variabile, expresii	68
3.1.2	Cuvinte cheie BASIC Spectrum	69
3.1.3	Structura programului BASIC	83
3.1.4	Mesaje de eroare	83
3.2	Compilerul HIBASIC	85
3.2.1	Instalarea	85
3.2.2	Directive	85
3.3	Fifth	86
3.3.1	Instalarea	86
3.3.2	Erori	86
3.3.3	Comenzi	87
3.3.4	Funcții	92
3.4	Beta BASIC	93
3.4.1	Instalarea	93
3.4.2	Comenzi	94
3.4.3	Funcții	101
3.4.4	Mesaje de eroare	104
4	PASCAL	106
4.1	Instalarea	106
4.2	Editorul	107
4.2.1	Generarea textului sursă	107
4.2.2	Listare	107
4.2.3	Editare	107
4.2.4	Comenzi de bandă magnetică	108
4.2.5	Comenzi de uz general	109
4.2.6	Compilarea și rularea programelor	109
4.3	Structuri de limbaj	109
4.4	Manipularea datelor în memorie	115
4.5	Funcții și proceduri predefinite	116
4.5.1	Funcții	116
4.5.2	Proceduri	118
4.6	Directive de compilare	119
4.7	Codurile erorilor	120
4.7.1	Erori la compilare	121
4.7.2	Erori la execuție	121
5 C		123
5.1	Instalarea	123
5.2	Editorul	124
5.2.1	Generarea textului sursă	124
5.2.2	Listare	124
5.2.3	Editare	124
5.2.4	Comenzi de bandă magnetică	125
5.2.5	Comenzi de uz general	126
5.3	Structuri de limbaj	126
5.3.1	Introducere	126
5.3.2	Convenții lexicale	126
5.3.3	Interpretarea identificatorilor	127
5.3.4	Obiecte și s-valori (l-values)	128
5.3.5	Conversii	128
5.3.6	Expresii	129

5.3.7 Declarații	132
5.3.8 Instrucțiuni	134
5.3.9 Definiții externe	136
5.3.10 Domeniile de valabilitate ale declarațiilor	136
5.3.11 Preprocesorul	136
5.3.12 Expresii constante	137
5.4 Funcții predefinite	138
5.5 Codurile erorilor	139
5.6 Manipularea datelor în memorie	141
5.6.1 Dimensiunea tipurilor	141
5.6.2 Clase de memorie	141
5.6.3 Funcții	141
5.6.4 Fișiere	142
5.6.5 Regiștrii Z80	142
6 FORTH	143
6.1 Instalarea	143
6.2 Structuri de limbaj	143
6.3 Structura memoriei	144
6.3.1 Harta memoriei	144
6.3.2 Structura cuvintelor	145
6.4 Editorul	146
6.5 Cuvinte predefinite	147
6.6 Codurile erorilor	163
7 PROLOG	164
7.1 Instalarea	164
7.2 Structuri de limbaj	164
7.2.1 Termeni	165
7.2.2 Liste	165
7.2.3 Atomi	166
7.2.4 Clauze	166
7.2.5 Comenzi	167
7.2.6 Module	167
7.3 Modul de lucru	168
7.4 Primitive Prolog.	168
7.5 Codurile erorilor	178

BIBLIOGRAFIE

- 1.*** — **ZX Spectrum + — User Guide**
Sinclair Research Ltd. — London 1984
- 2.**Dr. Ian Logan, Dr. Frank O'Hara**
The Complete Spectrum ROM Dissassembly
Melbourne House Publishers — Nashville 1983
- 3.**Jean-Francois Sehan**
Clefs pour le ZX Spectrum et Timex 2000
Editions du P.S.I. — Torcy Marne-la-Valle 1983
- 4.**M. Patrubany**
Totul despre microprocesorul Z80
Editura Tehnică — București 1989
- 5.*** — **HISOFT DEVPAC 3T**
HISOFT — 1984
- 6.**Richard Taylor**
Fifth — Users Manual
Computer Rentals Ltd.
- 7.*** — **Beta Basic**
Betasoft — 1984
- 8.*** — **HISOFT — Pascal 4T**
HISOFT — 1982
- 9.**Brian W. Kenirghan, Dennis M. Ritchie**
The C Programming Language
Prentice - Hall Inc. — New Jersey 1978
- 10.*** — **HISOFT C for ZX Spectrum**
HISOFT — 1984
- 11.**Radu Berindeanu, Agota Matecovits**
Forth, concept informatic și limbaj de programare
Editura Facla — Timișoara 1991
12. **K. L. Clark, J. R. Ennals, F. G. McCabe**
A Micro PROLOG Primer
L. P. A. — London 1981

În scopul de a răspunde cât mai bine dorințelor dumneavoastră vă rugăm să completați acest chestionar și să îl trimiteți pe adresa editurii: Editura APH, str. Cap. Preda nr. 12, sector 5, 76437 București 69.

De unde ați fost informat de această carte ?

- Librărie (magazin). Care?
- Mi-a spus un prieten
- Am citit într-un ziar, revistă. Care?
- Altele. Vă rugăm specificați.

Ce vîrstă aveți?

- 10 - 15
- 16 - 19
- 20 - 24
- peste 24

Ce părere aveți despre această carte ?

- Bună
- Potrivită
- Proastă
- Scumpă
- Preț mediu
- Ieftină

Vă rugăm să specificați ce alte teme ați dori să trateze Seria Informatică.

.
.
.
.

Ce alte cărți v-ar interesa ?

1) Tehnico - Științifice

.
.
.
.

2) Beletristică

.
.
.
.

Ocupația

Localitatea

Județul

Vă mulțumim.



Lucrarea contine informatii deosebit de utile tuturor utilizatorilor si în special programatorilor de calculatoare ZX SPECTRUM, multe dintre acestea fiind pentru prima oara publicate în România.

Sînt prezentate informatii cu privire la rutinele ROM (adresele lor si modul de utilizare), modul de utilizare al compilatoarelor PASCAL, C, FORTH, PROLOG, BASIC, BETABASIC precum si limbajul de asamblare al microprocesorului Z80 cu modul de utilizare al asamblorului/dezasamblorului GENS/MONS.

Cartea este conceputa ca un ghid în care informatiile necesare sa poata fi usor gasite si utilizate.

În curs de aparitie:

M. M. POPOVICI

– **BASIC pentru calculatoarele ZX SPECTRUM, HC, TIM-S, COBRA, CIP, JET ...**

•• **COLECTIE DE PROGRAME**

Lucrarea contine programe tehnico-stiintifice, de matematica, de interes general, programe de divertisment (jocuri), precum si prezentarea programelor BETA BASIC si HISOFT BASIC însoțite de aplicatii.

– **LIMBAJUL MASINA al calculatoarelor ZX SPECTRUM, HC, TIM-S, COBRA, CIP, JET ...**

Este prima lucrare care trateaza în mod unitar folosirea limbajului de asamblare Z80 si este ilustrata cu peste 150 de rutine care realizeaza spectaculoase efecte vizuale, sonore, de scriere, de animatie, etc.

ISBN 973-95175-6-0

LEI: ~~450~~

580