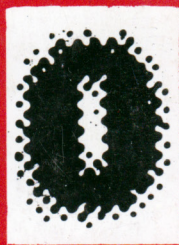


micro procesoare

Octavian PLETER
Cristian CONSTANTINESCU



ÎN
SPECTRUM

Octavian PLETER
Cristian CONSTANTINESCU
Seria
MICROPROCESOARE

Z80 în MICROCALCULATOARE COMPATIBILE SPECTRUM

Coordonatorul seriei: *Cristian Lupu*



EDITURA MILITARĂ - BUCUREȘTI 1995

Octavian PLETER
Cristian CONSTANTINESCU

Z80 ÎN MICROCALCULATOARE COMPATIBILE SPECTRUM

Coordonatorul seriei: *Cristian Lupu*



EDITURA MILITARĂ - BUCUREȘTI 1995

Octavian PLETER
Cristian CONSTANTINESCU

COMPATIBILE SPECTRUM MICROCALCULATOR ÎN 80

Coordonatorul seriei: Cristian Lăpuș

Redactor: DUMITRU NICOLESCU

Coperta: VICTOR ILIE

Tehnoredactor: D. ANDREI

Procesare computerizată: OCTAVIAN PLETER

VLAD COSMA

Bun de tipar 10.05.1995. Apărut 1995.

Coli de tipar 12 B 10495

Tiparul executat la tipografia ROMFEL



EDITURA MILITARĂ - BUCUREȘTI 1995

Cuvânt înainte	9
----------------------	---

1. PREZENTAREA GENERALĂ A ARHITECTURII ȘI A ORGANIZĂRII INTERNE A CALCULATOARELOR COMPATIBILE <i>SPECTRUM</i>	11
2. HARDWARE	18
2.1. Microprocesorul	19
2.1.1. Ceasul	19
2.1.2. Magistralele de date și de adrese	21
2.1.3. Semnalele de comandă și control	24
2.2. Memoria	27
2.2.1. Memoria ROM	27
2.2.2. Memoria RAM	33
2.3. Circuitul ULA	34
2.3.1. Generarea semnalului video	35
2.3.2. Citirea tastaturii	40
2.3.3. Generarea sunetului și interfața de casetofon	40
2.4. Sursa de alimentare	43
2.5. Metode de extindere a memoriei interne	43
2.5.1. <i>Spectrum</i> 80K	43
2.5.2. Amstrad <i>Spectrum</i> 128 +3	44
2.6. Interfața 1 de comunicație și de disc	44
2.6.1. Discul	48
2.6.2. Comunicația serială RS232	49
2.6.3. Rețeaua locală	50
2.7. Prezentarea generală a compatibilelor românești	50
3. PROGRAMAREA ÎN LIMBAJ DE ASAMBLARE PE COMPATIBILELE <i>SPECTRUM</i>	53
3.1. Instrucțiunile microprocesorului Z80	53
3.1.1. Tabelele matriceale ale instrucțiunilor	53
3.1.2. Tabelul instrucțiunilor în ordinea vitezei de execuție	54
3.2. Particularități de programare a microprocesorului Z80 la compatibilele <i>Spectrum</i>	54
3.2.1. Localizarea programelor	54
3.2.2. Registre rezervate	55
3.2.3. Regimul întreruperilor	55
3.3. Principalele subrutine din ROM	56
3.3.1. Subrutinele de casetă	56
3.3.1.1. Încărcarea de pe casetă (LD BYTES)	58
3.3.1.2. Salvarea pe casetă (SA BYTES)	58

3.3.1.3. Exemple de apelare	59
3.3.2. Generatorul de sunet	60
3.3.2.1. Apelarea subrutinei	60
3.3.2.2. Exemple de utilizare	61
3.3.3. Scrierea unui caracter	61
3.3.3.1. Apelarea subrutinei	61
3.3.3.2. Exemple de utilizare	62
3.3.4. Scrierea unui mesaj	63
3.3.4.1. Apelarea subrutinei	63
3.3.4.2. Exemple de utilizare	63
3.3.5. Subrutinele grafice	64
3.3.5.1. Ștergerea ecranului	64
3.3.5.2. Aprinderea unui pixel (PLOT)	65
3.3.5.3. Trasarea unei linii (DRAW)	66
3.3.5.4. Trasarea unui cerc	66
3.3.5.5. Poziționarea atributelor de culoare	66
3.3.5.6. Exemplu de utilizare	67
3.3.6. Calculatorul aritmetic	68
3.3.6.1. Exemplu de utilizare	71
3.3.7. Citirea tastaturii	74
3.3.8. Tratarea erorilor	75
3.3.8.1. Exemplu de interceptare a rutinei de eroare	76
3.3.9. Tratarea întreruperilor mascabile	77
3.3.9.1. Exemplu de interceptare a întreruperii mascabile	77
3.4. Tehnici de programare	79
3.4.1. Transferuri pe 16 biți	80
3.4.2. Cicluri	83
3.4.3. Utilizarea registrelor alternative	84
3.4.4. Subrutine cu parametri în linie (<i>in-line</i>)	86
3.4.4.1. Transmiterea parametrilor prin valoare	87
3.4.4.2. Transmiterea parametrilor prin adresă	87
3.4.4.3. Selecție după variabilă sau condiție	88
3.4.4.4. Tehnica <i>hook-code</i>	90
3.4.5. Înmulțirea rapidă a numerelor întregi	91
3.4.6. Selecții prin tabele de salturi	94
3.4.7. Tehnica decupării de biți	94
4. APLICAȚII ALE COMPATIBILITATEI <i>SPECTRUM</i>	96
4.1. Interfață de imprimantă	96
4.1.1. Definirea termenilor referitori la imprimantă	96
4.1.2. Programul interfeței	98
4.1.2.1. Încărcarea și activarea programului	99
4.1.2.2. <i>Formatter</i> -ul de text	100
4.1.2.3. Tipărirea fișierelor <i>Tasword</i>	105
4.1.2.4. <i>Formatter</i> -ul grafic	108
4.1.2.5. Harta memoriei	109
4.1.2.6. Compatibilitatea cu alte programe	109
4.1.2.7. Comenzile programului de interfață <i>Tuttiprint V4.7</i>	112
4.1.2.8. Listingul programului	114
4.1.3. Schema interfeței	126

4.2. Dezasamblor Z80	128
4.3. Grafice de funcții	130
4.4. Program de reprezentări grafice 3D	132
4.4.1. Termeni specifici pentru grafică	132
4.4.2. Instrucțiuni grafice	134
4.4.3. Aplicații ale programului	140
4.4.3.1. Aplicație în prezentarea datelor	140
4.4.3.2. Aplicație în arhitectură	141
4.4.3.3. Aplicație în inginerie	142
4.4.4. Listingul programului	142
4.5. Grafică de înaltă rezoluție la imprimanta matriceală	149
4.6. Derivator simbolic de funcții	152
4.6.1. Listingul programului	154
Bibliografie	157
Anexa A - Tabelul matriceal principal al instrucțiunilor Z80	160
Anexa B - Tabelul matriceal al instrucțiunilor extinse cu prefixul #CB	161
Anexa C - Tabelul matriceal al instrucțiunilor extinse cu prefixul #ED	162
Anexa D - Tabelul instrucțiunilor Z80 în ordinea vitezei de execuție	163
Anexa E - Tabelul caracterelor ASCII Spectrum	164
Anexa F - Tabelul variabilelor de sistem	173
Anexa G - Modulurile de lucru și funcțiile tastaturii	180
Anexa H - Funcțiile interpretorului Basic	185

Spectrum pe PC-uri, ce rulează cod mașină mult mai repede și încarcă programe de pe casetă. Un asemenea emulator am folosit și la scrierea acestei cărți. Credem că prin apariția emulațoarelor, patrimoniul software al Spectrum-ului devine accesibil și mai accesibil având șansa, dacă nu de a fi dezvoltat, cel puțin de a fi păstrat mai timp de acum înainte.

Spectrum-ul se poate caracteriza prin aceea că oferă o cantitate apreciabilă de inteligență tehnică la un preț deosebit de accesibil. Oricine îl studiază are foarte multe de învățat, de la principii până la tehnici de programare și soluții tehnologice, ceea ce explică adoptarea sa rapidă de către utilizatorii și programatorii pasionați, cărora le oferă reale satisfacții profesionale. Nu același lucru se poate spune de exemplu despre unele calculatoare de tipul PC, la care reacții ale tehnicienilor, de felul: "Fine, dar se putea găsi o soluție mai inteligentă!" nu sunt chiar rare. Este adevărat că utilizatori pasionați într-adevăr încât să intre în detalii tehnice sunt puțini, însă această minoritate a avut de jucat un rol decisiv în faza de pionierat a calculatoarelor personale. Era crucial în anii '80 ca programatorii pasionați și ingenioși să "placă" mașina pe care și pentru care lucrau. Degeaba s-au produs calculatoare mai performante decât Spectrum-ul, la același preț (de exemplu Oric Atmos sau Acorn BBC). Dacă modelul nu oferea suficiente satisfacții profesionale elitei, era compromis din cauza lipsei software-ului, pe care numai această elită era capabilă să-l producă și în mare măsură, chiar să-l folosească. Sir Clive Sinclair, inventatorul Spectrum-ului, era de părere că un om obișnuit al anilor '80 cumpăra un calculator fără să știe prea bine la ce

CUVÂNT ÎNAINTE

Calculatorul Spectrum reprezintă un fenomen aparte. În industria calculatoarelor, un model vândut în câteva mii de exemplare era în anii '70 un succes răsunător. În 1982 a apărut Spectrum-ul, care în primii ani s-a vândut în 2 milioane de exemplare. Acest "Ford T" al calculatoarelor a jucat un rol deosebit în configurarea industriei de hardware și software de astăzi.

În afara importanței lor istorice, calculatoarele Spectrum și compatibilele mai au încă o valoare funcțională, ele utilizându-se, de exemplu, destul de frecvent la noi în țară. Producția în Marea Britanie a încetat de câțiva ani, dar în Brazilia, CSI și România, asemenea calculatoare se mai fabrică, ceea ce reprezintă și un record de longevitate în această industrie.

În ultima vreme au apărut numeroase emulatoare fidele de Z80 și Spectrum pe PC-uri, ce rulează cod mașină mult mai repede și încarcă programe de pe casetă. Un asemenea emulator am folosit și la scrierea acestei cărți. Credem că prin apariția emulatoarelor, patrimoniul software al Spectrum-ului devine astfel și mai accesibil având șansa, dacă nu de a fi dezvoltat, cel puțin de a fi păstrat mult timp de acum înainte.

Spectrum-ul se poate caracteriza prin aceea că oferă o cantitate apreciabilă de inteligență tehnică la un preț deosebit de accesibil. Oricine îl studiază are foarte multe de învățat, de la principii până la tehnici de programare și soluții tehnologice, ceea ce explică adoptarea sa rapidă de către utilizatorii și programatorii pasionați, cărora le oferă reale satisfacții profesionale. Nu același lucru se poate spune de exemplu despre unele calculatoare de tipul PC, la care reacții ale tehnicienilor, de felul: "Bine, dar se putea găsi o soluție mai inteligentă!" nu sunt chiar rare. Este adevărat că utilizatori pasionați într-atât încât să intre în detalii tehnice sunt puțini, însă această minoritate a avut de jucat un rol decisiv în faza de pionierat a calculatoarelor personale. Era crucial în anii '80 ca programatorii pasionați și ingenioși să "placă" mașina pe care și pentru care lucrau. Degeaba s-au produs calculatoare mai performante decât Spectrum-ul, la același preț (de exemplu Oric Atmos sau Acorn BBC). Dacă modelul nu oferea suficiente satisfacții profesionale elitei, era compromis din cauza lipsei software-ului, pe care numai această elită era capabilă să-l producă și în mare măsură, chiar să-l folosească. Sir Clive Sinclair, inventatorul Spectrum-ului, era de părere că un om obișnuit al anilor '80 cumpăra un calculator fără să știe prea bine la ce

îi va folosi și de multe ori ajungea la concluzia că a aruncat banii pe fereastră. Pentru a folosi un calculator personal din acei ani la rezolvarea unei probleme practice mai importante, trebuia să fii un expert, un fel de "vrăjitor" al tastaturii, deși circula utopia "calculatoarelor care rezolvă toate problemele". Sinclair a dorit să demonteze această utopie, cu prețul pierderii numai a 100\$ (cât costa un ZX-80, ZX-81 sau un Spectrum) în loc de 1000\$ (cât costa un Commodore PET sau un Apple II). În plus, Sir Clive Sinclair era un inventator cu vocație. Realizase primul calculator de buzunar cu operații aritmetice și primul televizor de buzunar, cu tub cinescop extraplat, cu deflexie asimetrică. Era obișnuit cu ingeniozitatea tehnică, de care a făcut risipă în proiectarea Spectrum-ului.

În țara noastră, există o adevărată "școală" a Spectrum-ului, la care s-au format mulți dintre experții de astăzi.

"Motorul" Spectrum-ului este microprocesorul Zilog Z80, practic cel mai performant microprocesor de 8 biți produs vreodată. Spectrum-ul și Z80-ul s-au conjugat perfect, cu mult dincolo de simpla lor însumare. Meritul acestei cuplări revine creatorului sistemului de operare, Steven Vickers, colaborator apropiat al lui Sir Clive Sinclair.

Motivale care ne-au determinat să scriem această carte pot fi ușor desprinse din cele prezentate mai sus. Cartea se adresează în principal tinerilor, cu precădere studenților. Totuși, nu este o carte pentru începători, sau mai exact nu pentru acei începători care au de gând să rămână așa. Sperăm să fie apreciată și de pasionații de calculatoare (personale) de toate vârstele, cărora de asemenea ne adresăm.

Capitolul 1 este o prezentare succintă a calculatorului Spectrum și a principalelor noțiuni curente în utilizarea sa.

Capitolul 2 intră în amănunte hardware, prezentând microprocesorul Z80 și schemele electronice ale calculatorului.

Capitolul 3 este dedicat programării în limbajul de asamblare Z80, cu un accent asupra tehnicilor de programare, care fac mai rar obiectul literaturii de specialitate.

Capitolul 4 prezintă câteva aplicații hardware și software originale, create de autori în decursul anilor 1982-1990. Ne-am străduit să oferim cititorilor cât mai multe exemple reprezentative. Listingurile incluse demonstrează practic cum se aplică tehnicile de programare în limbajul de asamblare și în Basic. Totodată, capitolul ajută la fixarea limitelor unui microprocesor Z80 în aplicații.

Anexele conțin tabele și planșe de referință, majoritatea fiind originale.

AUTORII

PREZENTAREA GENERALĂ A ARHITECTURII ȘI A ORGANIZĂRII INTERNE A CALCULATORILOR COMPATIBILE *SPECTRUM*

Calculatorul *Sinclair Spectrum* a fost construit în 1982 de firma Sinclair Research Ltd. din Cambridge, Marea Britanie, în ideea de a produce un calculator educațional extrem de ieftin, dar cu performanțe superioare predecesoarelor de succes: Sinclair ZX-80 și ZX-81. Acestea din urmă aveau următoarele dezavantaje:

- erau lente pentru că microprocesorul era încărcat cu generarea imaginii TV¹;
- aveau prea puțină memorie instalată (1 kB și respectiv 4 kB de RAM static; între timp RAM-ul dinamic devenise ieftin, la capacități mult superioare);
- aveau o grafică monocromă rudimentară;
- nu permiteau redefinirea caracterelor;
- nu aveau posibilități de sunet;
- tastatura cu membrană plată era nepopulară printre utilizatori;
- interpretorul Basic avea mai multe lacune.

Proiectarea *Spectrum*-ului a fost lansată în urma constatării acestor defecte, căutându-se soluții tehnice ieftine de a le elimina. Astfel, noul calculator era dotat cu:

- un circuit specializat de generare a imaginii TV;
- 16 kB de RAM dinamic expandabil intern la 48 kB;
- grafică de înaltă rezoluție (256×192) în 8 culori și 2 intensități;
- posibilitatea definirii caracterelor;
- generator de sunete prin software, printr-un port de 1 bit;
- tastatură cu membrană de cauciuc;
- un nou interpretor Basic în 16 kB de ROM.

Totodată, *Spectrum*-ul perpetua ideile excelente ale modelelor anterioare: pe de-o parte, aceea că în fiecare casă trebuie să pătrundă un calculator și pe de

¹ La ZX-81, microprocesorul execută instrucțiuni utile doar în timpul cursei de întoarcere a spotului, în rest se ocupă de generarea imaginii TV.

altă parte, că în fiecare casă există deja un televizor și un casetofon. În timp ce un IBM-PC se vindea împreună cu monitor și unitate de disc floppy, *Spectrum*-ul putea folosi un televizor color sau alb-negru obișnuit și cel mai ieftin casetofon.

Microprocesoarele de 16 biți (Intel 8086, Motorola 68000 și Zilog Z8000) erau încă prea scumpe și antrenau o gamă de circuite periferice de asemenea scumpe, iar echipa de la Sinclair Research căpătase deja o experiență valoroasă cu Z80, care merita a fi folosită. De altfel, chiar și IBM a oscilat la început între Intel 8088 și un microprocesor de 8 biți, pentru IBM-PC [35]. De abia în 1984, la modelul QL, Sinclair va recurge la un microprocesor de 32 biți, Motorola 68008. În treacăt fie spus, părăsirea microprocesorului Z80 va aduce dezastrul financiar al firmei Sinclair Research, care va fi vândută în 1986 firmei Amstrad pentru 12 milioane de lire sterline. Numai cu doi ani înainte, datorită succesului de piață al *Spectrum*-ului, aceeași firmă era evaluată la peste 200 milioane de lire [59]. O altă firmă celebră care a înregistrat pierderi financiare însemnate datorită grabei de a trece la microprocesoare de 16 sau 32 biți a fost Commodore, cu modelul Amiga, bazat pe Motorola 68000. Sir Clive Sinclair explica acest paradox astfel: "Nu am găsit o modalitate de a extrage din microprocesorul 68000 avantaje suplimentare pentru cumpărător... Eu am dorit un QL cu Z80, dar cei mai mulți dintre ingineri și Nigel¹ voiau să lucreze cu 68000. Mie nu mi se părea că puteam face cu acel circuit mai multe decât cu Z80 și în schimb costa o grămadă de bani. Adevărul este că tot ce poți face cu 68000 poți face și cu Z80. Desigur, pe hârtie are tot felul de avantaje, dar în practică multe din ele nu se confirmă." [59]. Firește că pledoaria pentru Z80 a lui Sir Clive Sinclair este valabilă pentru anii '80, între timp piața s-a maturizat suficient și pentru microprocesoare mult mai performante.

Deși firma Zilog oferea o dată cu Z80 CPU și o întreagă familie de circuite periferice (Z80 PIO, SIO, CTC, DME), *Spectrum*-ul nu putea beneficia de ele datorită faptului că erau mai scumpe decât microprocesorul însuși. Acest fapt era datorat unei politici abile de prețuri a firmei Zilog, de a vinde unitatea centrală cu profit minim pentru a-o face atrăgătoare, recuperând în schimb prin comercializarea celorlalte circuite. La Sinclair Research soluțiile tehnice trebuiau să fie în primul rând foarte ieftine.

Arhitectura calculatorului *Spectrum* pornește de la ideile de principiu arătate și impresionează prin simplitate, eficiență și ingeniozitate. În figura 1.1 se remarcă următoarele elemente:

1. Unitatea centrală (CPU) cu un microprocesor Z80A la 3.5 MHz. Frecvența de ceas are această valoare deoarece este divizată din frecvența mașinii video, care pentru afișarea unui pixel, adică a unui punct pe ecran, are nevoie de 7 MHz. Frecvența maximă după catalog a microprocesorului este de 4 MHz, dar o mașină video la 8 MHz ar fi condus la o imagine prea strâmtă pe

¹ Nigel Searle, partenerul de afaceri din Statele Unite al lui Sir Clive Sinclair. El avea misiunea de a comercializa variantele americane ale computerelor Sinclair, denumite Timex.

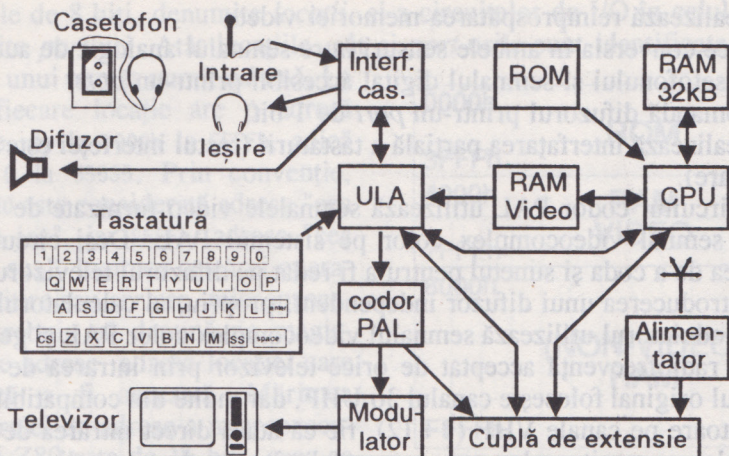


Fig 1.1. Schema-bloc funcțională a unui calculator *Spectrum*

ecran, la aceeași rezoluție. Unitatea centrală primește în mod normal o întrerupere la începutul fiecărui cadru TV, adică la fiecare 20 ms. Prin contorizarea acestor întreruperi, microprocesorul are o referință de ceas de timp real.

2. Memoria nevolatilă (ROM) are capacitatea de 16 kB, adică 16.384 octeți și conține funcțiile de bază ale sistemului de operare, sistemul de operare, interpretorul Basic, generatorul de caractere de ecran și *driver*-e pentru casetofon și imprimanta ZX Printer.

3. Memoria volatilă (RAM) are capacitatea de 48 kB, adică 49.152 octeți și conține harta ecranului, un *buffer* de imprimantă, variabilele de sistem, programul Basic cu variabilele sale, programe în cod mașină, stivele calculatorului și harta caracterelor definite de utilizator (UDG). Fizic, memoria volatilă este compusă din două bancuri de memorie: memoria video de 16 kB și memoria suplimentară de 32 kB. Împreună cu cei 16 kB de memorie nevolatilă, memoria volatilă ocupă întreg spațiul de adresare al microprocesorului Z80. De aceea, extinderile de memorie volatilă la *Spectrum* sunt problematice. În schimb, calculatorul poate funcționa cu mai puțină memorie volatilă, minimum 16 kB, ajustându-și automat parametrii la inițializare. Pe parcursul lucrării, vom considera totuși valoarea maximă a capacității.

4. *controller*-ul de periferie (ULA) integrează majoritatea funcțiilor din afara unității centrale și anume:

a. citește harta ecranului din memoria video și generează semnalul videocomplex alb-negru de televiziune și semnale separate de culoare;

b. arbitrează conflictele de acces la memoria video între el însuși și unitatea centrală;

c. generează câte o întrerupere mascabilă pentru unitatea centrală la începutul fiecărui cadru TV;

- d. realizează reîmprospătarea memoriei video;
- e. face conversia în ambele sensuri între semnalul analogic de audiofrecvență al casetofonului și semnalul digital accesibil printr-un *port*;
- f. comandă difuzorul printr-un *port* de 1 bit;
- g. realizează interfațarea parțială a tastaturii (restul interfeței este realizat prin software).

5. Circuitul codor PAL utilizează semnalele video furnizate de ULA și generează semnal videocomplex color pe sistemul PAL. Deși circuitul are posibilitatea de a coda și sunetul pentru a fi redat pe difuzorul televizorului, s-a preferat introducerea unui difuzor independent, propriu, al calculatorului.

6. Modulatorul utilizează semnalul videocomplex color PAL și generează semnal de radiofrecvență acceptat de orice televizor prin intrarea de antenă. *Spectrum*-ul original folosește canalul 36 UHF, dar multe din compatibile fie că au modulatoare pe canale VHF ($1 \div 12$), fie că atacă direct intrarea de semnal videocomplex a monitoarelor sau a unor televizoare. În acest din urmă caz, modulatorul nu mai este necesar.

7. Tastatura are 40 de taste dispuse matriceal 8×5 . În afară de cele 26 de litere ale alfabetului englez și cele 10 cifre, mai există 4 taste speciale: SPACE (spațiu), CAPS SHIFT (majuscule), SIMBOL SHIFT (simboluri) și ENTER (intrare). Tastatura este citită de unitatea centrală prin intermediul *controller*-ului de periferie, iar tasta apăsată este decodificată printr-un program activat la fiecare 20 ms de întreruperea de început de cadru TV.

8. Circuitul analogic de casetofon și difuzor este interpus între *controller*-ul de periferie și dispozitivele respective. El realizează adaptarea în impedanță și ca nivel de semnal, precum și o funcție logică: difuzorul este activat cu un nivel sonor scăzut în timpul redării pe casetofon, dar este complet dezactivat în timpul înregistrării.

9. Circuitul de alimentare conține un stabilizator de tensiune la +5 V și generatoare de tensiune la -5 V și +12 V pentru alimentarea triplă cerută de memoria video și pentru generarea semnalului video. Circuitul este alimentat din exterior cu o tensiune continuă nestabilizată de $9 \div 14$ V.

Periferia externă standard a calculatorului, compusă din televizor pentru afișarea imaginii și casetofon pentru stocarea programelor și datelor, se conectează prin mufele specializate de pe panoul din spate al calculatorului. Tot în spate se găsește și mufa de alimentare cu tensiune continuă și cupla de extensie, care permite cuplarea unor interfețe sau a imprimantei ZX Printer.

Microprocesorul Z80 constituie unitatea centrală a calculatorului. Prin intermediul magistralelor de adrese și de date, acesta comunică cu două tipuri de circuite: memoriile și circuitele de intrare/ieșire (I/O). Comunicarea este în ambele sensuri și se numește fie *citire*, fie *scriere*, după sensul operației din punctul de vedere al unității centrale. Se spune despre un microprocesor că este de 4, 8, 16 sau 32 de biți, după mărimea magistralei sale de date. La Z80, magistrala de date este de 8 biți, ceea ce presupune că fiecare transfer de informație între unitatea centrală și memorii sau circuitele de I/O se face în cuante de 8 biți, denumite *octeți* sau *bytes*. Acest fapt determină organizarea memoriei

în celule de 8 biți, denumite *locații*, și a circuitelor de I/O în celule de 8 biți, denumite *port-uri*. Atât locațiile, cât și *port-urile* sunt identificate prin intermediul unui număr numit *adresă*. La Z80, fiecare locație are o adresă proprie, de la 0000h la FFFFh, adică de la 0 la 65535. Prin convenție, adresa 0 este considerată adresa "cea mai de jos", iar FFFFh, adresa "cea mai de sus". Înaintea oricărei operații de citire sau scriere, microprocesorul trebuie să depună pe magistrala de adrese, adresa locației care urmează a fi accesată. Mărimea magistralei de adrese a microprocesorului Z80 este de 16 biți, ceea ce corespunde numărului total de 65536 de locații care pot fi adresate. Practic vorbind, numărul total de *port-uri* este tot de 65536, dar din motive de compatibilitate cu predecesorul Intel 8080, la Z80 se consideră 256 de "*port-uri*", fiecare cu 256 de "*dispozitive*". De aceea, de multe ori, adresa unui *port* este dată numai pe 8 biți care corespund celor mai puțin semnificativi 8 biți ai adresei: de la 00h la FFh, sau de la 0 la 255.

0000h		0
3FFFh	ROM	16383
4000h	RAM	16384
7FFFh	VIDEO	32767
8000h		32768
	(NON-VIDEO) RAM	
FFFFh		65535

Fig. 1.2. Organizarea spațiului de memorie

Totalitatea locațiilor de memorie care pot fi adresate independent se numește *spațiu de memorie*. La *Spectrum*, întreg spațiul de memorie adresabil este utilizat, adică cele 65536 de locații există fizic. Organizarea spațiului de memorie este reprezentată în figura 1.2. Locațiile din primii 16 kB pot fi prin urmare numai citite, iar celelalte pot fi accesate în ambele sensuri.

În figura 1.3 sunt reprezentate principalele elemente ale imaginii produse de calculatorul *Spectrum* pe ecranul unui televizor. În primul rând, imaginea se

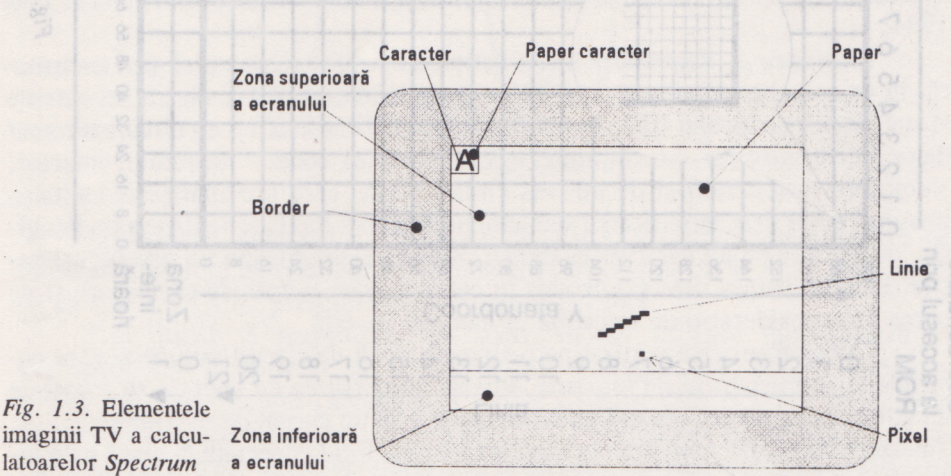


Fig. 1.3. Elementele imaginii TV a calculatoarelor *Spectrum*

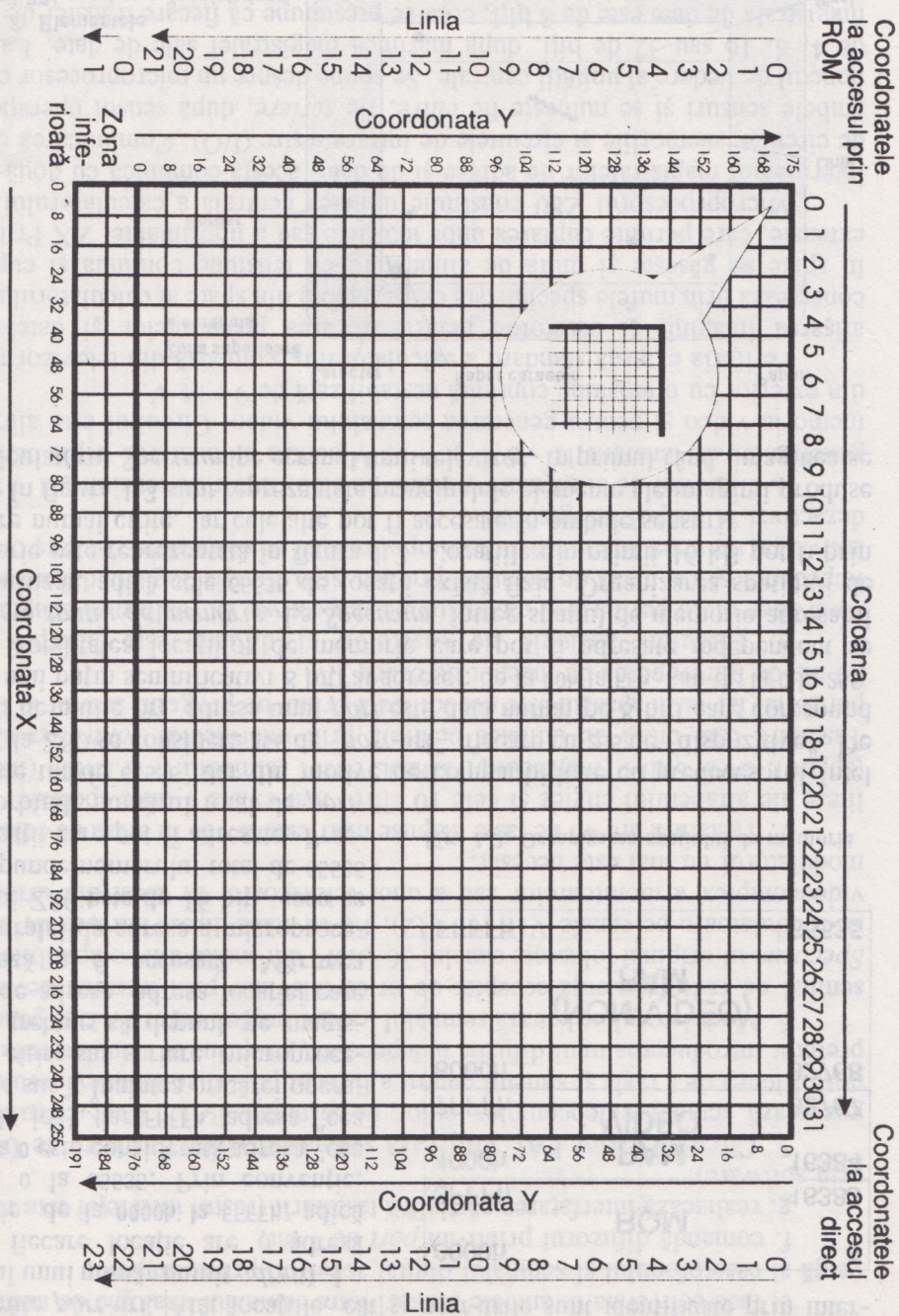


Fig. 1.4. Harta ecranului și sistemele de coordonate utilizate

împarte în două regiuni: în centru se găsește zona utilă, numită *paper*, iar pe margine *paper*-ul este înconjurat de o zonă inactivă numită *border*. Rostul *border*-ului este acela de a acoperi periferia ecranului, unde destul de multe televizoare sunt imprecise. Singurul element care poate fi schimbat la *border* este culoarea acestuia. *Paper*-ul este destinat afișării textului și graficii. La rândul lui, este împărțit în două părți inegale: zona superioară, principală, și zona inferioară, unde se afișează de obicei comenzile sau instrucțiunile în curs de editare. După cum se observă în figura 1.4, *paper*-ul este organizat ca o matrice de matrice. Elementele cele mai mici de imagine se numesc *pixeli*. O matrice de pixeli de 8×8 se numește *caracter*. *Paper*-ul conține 24×32 caractere, sau 256×192 pixeli, dintre care 21×32 caractere în zona superioară, respectiv 256×176 pixeli și 2×32 caractere în zona inferioară, respectiv 256×16 pixeli. Un caracter poate conține o literă, o cifră sau un simbol, reprezentat într-o matrice de 8×8 pixeli.

Precizarea poziției unui pixel sau a unui caracter pe ecran se face prin două coordonate, la intersecția cărora se găsește elementul respectiv. La caractere, coordonatele se numesc *linia* și *coloana*, iar la pixeli, X și Y.

În figura 1.4 se poate remarca faptul că există două tipuri de coordonate: cele de la accesul normal, prin ROM, unde de exemplu coordonata Y este definită numai în partea superioară a *paper*-ului și crește în sus, precum și cele de la accesul direct, din cod mașină, unde coordonata Y crește în jos și acoperă tot *paper*-ul.

HARDWARE

Deși arhitectura nu a fost modificată niciodată, iar funcțional toate calculatoarele *ZX Spectrum* produse de Sinclair Research Ltd. sunt identice, schemele electrice de detaliu și plasarea componentelor electronice pe placă au fost modificate de-a lungul timpului, fie pentru creșterea fiabilității, fie din rațiuni economice. Firma producătoare a grupat aceste modificări în trei versiuni majore (*issues*) care au dus la reproiectarea cablajului imprimat.

Versiunea 1. Este modelul inițial al calculatorului. Caracteristica principală constă în faptul că numai 16 kB sunt lipite pe placa de bază, restul memoriei montându-se opțional printr-o cuplă de extensie. Soluția a fost imediat abandonată datorită cererii de sisteme cu 48 kB de RAM. În această versiune s-au fabricat numai 26.000 de calculatoare;

Versiunea 2. A fost reproiectată placa de circuit imprimat pentru a putea realiza fie calculatoare de 16 kB, fie de 48 kB. S-au modificat sursa internă de alimentare, ULA, precum și codorul PAL. Acest model s-a fabricat până în anul 1984;

Versiunea 3. Remediază unele deficiențe constatate la versiunea anterioară, în principal privind supraîncălzirea componentelor active ale sursei interne și stabilitatea codorului PAL. Acesta din urmă cuprinde acum un circuit automat de reglare a culorilor, care elimină reglajul complex necesar modelului anterior; Versiunea beneficiază și de un nou circuit ULA, care îmbunătățește calitatea imaginii color prin modificarea semnalului de burst, dar care realizează o imagine "deplasată" cu un caracter la stânga (8 pixeli), producând un *border* asimetric. Este versiunea care s-a produs în cel mai mare număr de exemplare și de aceea am considerat-o de referință în lucrarea de față.

Calculatorul este realizat pe o singură placă de circuit imprimat care cuprinde și stabilizatorul de tensiune. Pentru alimentarea cu energie se folosește un alimentator extern capabil să furnizeze 9 V la un curent de maximum 1,5 A. Tastatura formează partea superioară a carcasei, fiind conectată la placă prin in-

intermediul a două cabluri plate și este realizată folosind o membrană de cauciuc și trasee metalice depuse pe material plastic. Toate conectoarele se află pe perețile din spate al carcasei.

2.1. MICROPROCESORUL

Z80 este unul dintre cele mai performante și mai răspândite microprocesoare de 8 biți. Explicarea funcționării sale în detaliu este bine acoperită de literatura de specialitate românească. În cele ce urmează se vor prezenta numai acele aspecte necesare înțelegerii funcționării calculatorului *Spectrum*. Pentru cei care abordează pentru prima oară subiectul, recomandăm să consulte lucrările [27], [29], [37] sau [2].

2.1.1. CEASUL

Pentru funcționarea sa normală, Z80 are nevoie de o referință care să asigure sincronizarea tuturor semnalelor. Acest semnal poartă denumirea de semnal de *ceas* (T) și este de fapt un semnal digital periodic cu factorul de umplere 50% (figura 2.1). Pentru a avea o bună stabilitate în timp, ceasul se obține plecând de la un oscilator bazat pe un rezonator cu cristal de cuarț.

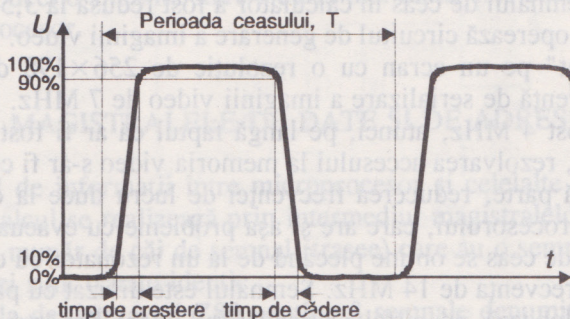


Fig. 2.1. Ceasul microprocesorului Z80

O perioadă completă a semnalului de ceas (T) poartă numele de *tact* sau *stare*. În literatura de specialitate, semnalul este notat fie cu Φ , după Intel 8080, fie cu CK sau CLK , prescurtări ale cuvântului *Clock*.

Este locul să reamintim, pe scurt, modul în care funcționează un microprocesor. El reunește într-un singur circuit integrat două dintre componentele de bază ale oricărui calculator și anume *Control Unit* (CU) – unitatea de control – și *Arithmetic and Logic Unit* (ALU) – unitatea aritmetică și logică. Calculatorul are nevoie de cel puțin încă două componente: memoria și sistemul

de intrări-ieșiri (periferia). Funcționarea microprocesorului are la bază un ciclu care constă din repetarea a patru pași:

- citește următoarea instrucțiune din memorie;
- decodifică această instrucțiune;
- execută instrucțiunea decodificată;
- salt la pasul 1.

Acești patru pași reprezintă de fapt execuția unei instrucțiuni și poartă numele de *ciclu-instrucțiune*. În funcție de instrucțiunea executată, un ciclu-instrucțiune se poate subdivide în mai multe *cicluri-mașină*, care corespund, în general, unei operații elementare, cum ar fi citirea din memorie, scrierea la un dispozitiv periferic etc. Primii doi pași corespund unui ciclu-mașină special, comun tuturor instrucțiunilor, care poartă denumirea de *ciclu de fetch*, de extragere, în care se aduce și se decodifică instrucțiunea. Fiecare ciclu-mașină poate dura un număr diferit de stări de ceas (T).

Toate transferurile de date între microprocesor și celelalte blocuri funcționale ale calculatorului, precum și operațiile interne se realizează la intervale de timp tranșate de semnalul de ceas, mai exact de schimbarea valorii acestuia, denumită *tranziție* sau *front*. Prin urmare, din punctul de vedere al vitezei de calcul, cu cât tranzițiile sunt mai numeroase în unitatea de timp, cu atât mai rapid va funcționa microprocesorul. Din păcate, frecvența maximă la care poate funcționa corect un circuit electronic este limitată tehnologic.

Microprocesorul Zilog Z80A cu care este dotat calculatorul *Spectrum* poate funcționa la o frecvență maximă de 4 MHz. Așa cum am spus, frecvența de tact pentru semnalul de ceas în calculator a fost redusă la 3,5 MHz, datorită modului în care operează circuitul de generare a imaginii video. Pentru a obține un punct "pătrat" pe un ecran cu o rezoluție de 256×198 de puncte, este necesară o frecvență de serializare a imaginii video de 7 MHz. Dacă frecvența ceasului ar fi fost 4 MHz, atunci, pe lângă faptul că ar fi fost necesare două cristale de cuarț, rezolvarea accesului la memoria video s-ar fi complicat foarte mult. Pe de altă parte, reducerea frecvenței de lucru duce la o încălzire mai redusă a microprocesorului, care are și așa probleme cu evacuarea căldurii.

Semnalul de ceas se obține plecând de la un rezonator cu cristal de cuarț, funcționând la frecvența de 14 MHz. Semnalul este divizat cu patru în circuitul ULA, care înglobează de altfel și oscilatorul. Mai departe, semnalul este sincronizat cu accesul generatorului video la memorie și apoi este aplicat pe intrarea CK a microprocesorului.

După cum am văzut, semnalul de ceas are un rol special și din acest motiv i se impun anumite cerințe. Tranzițiile sale trebuie să fie cât mai rapide, cu fronturi cât mai abrupte. De asemenea, nivelul 0 trebuie să fie cât mai aproape de 0 V, iar nivelul 1 cât mai aproape de 5 V. Totodată, încărcarea sa capacitivă este destul de importantă. Toate aceste cauze fac ca o ieșire standard TTL să nu corespundă necesităților.

În figura 2.2 este dată soluția adoptată în calculatorul *Spectrum*. Se folosește un tranzistor de comutație pe post de inversor. Dacă baza este menținută în 0, atunci tranzistorul este blocat și semnalul CLK este legat prin

rezistența de $180\ \Omega$ la $+5\text{ V}$. Dacă pe rezistența din bază se aplică 1, atunci tranzistorul intră în saturație, tensiunea colector-emitor (și deci pe CLK) fiind sub $0,1\text{ V}$. Pentru a accelera comutarea tranzistorului, în paralel cu rezistența din bază este prevăzută o diodă care are un dublu rol. Pe de o parte, atunci când este polarizată invers, se comportă ca un condensator de $3 \div 7\text{ pF}$, derivând semnalul și asigurând o ușoară supratensiune pe bază. Pe de altă parte, când este polarizată direct, scurtcircuitază rezistența de $3,3\text{ k}\Omega$, micșorând constanta de timp a circuitului format de aceasta cu capacitatea parazită bază-emitor și contribuind astfel la evacuarea sarcinii din baza tranzistorului.

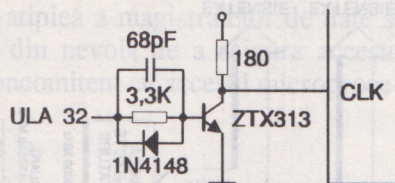


Fig. 2.2. Circuitul semnalului de ceas la calculatorul *Spectrum*

După cum se va vedea ceva mai târziu, semnalul de ceas care ajunge la microprocesor nu este riguros periodic. Acest lucru se datorează faptului că generarea semnalului video are prioritate maximă în privința accesului la memorie. Dacă ULA are nevoie de o dată din memorie, iar microprocesorul încearcă și el să acceseze memoria video, atunci ULA blochează semnalul de ceas în starea 0, oprind astfel orice activitate a microprocesorului în perioada accesului la memorie al ULA. Acest lucru este posibil deoarece microprocesorul nu își schimbă starea decât la apariția unei tranziții a ceasului. Faptul că ceasul a fost blocat, și deci o anumită stare a durat mai mult, este complet transparent pentru microprocesor.

2.1.2. MAGISTRALELE DE DATE ȘI DE ADRESE

Schimbul de informații între microprocesor și celelalte componente ale sistemului de calcul se realizează prin intermediul magistralelor. O magistrală constă dintr-un număr de căi de semnal (trasee) care au o semnificație comună și care de obicei sunt tratate identic.

Magistrala de date a lui Z80 conține 8 semnale denumite $D0 \div D7$. Din punctul de vedere al microprocesorului, magistrala este bidirecțională, sensul transferului de date fiind dictat de semnalele de control RD, WR, MREQ și TORQ.

Magistrala de adrese conține 16 semnale denumite $A0 \div A15$. Magistrala este unidirecțională, procesorul fiind singurul care are dreptul să genereze adrese în mod normal.

Ambele magistrale, precum și o parte din semnalele de control sunt realizate în tehnologie *tri-state*, adică au, în afară de stările 0 și 1, o a treia stare, denumită Z, de impedanță înaltă, în care microprocesorul cedează unui alt dispozitiv controlul semnalului respectiv.

În figura 2.3 este dată schema-bloc detaliată a calculatoarelor *Spectrum*, unde sunt reprezentate magistralele microsistemului, iar în figurile 2.5–2.9,

schemele electronice. Se observă conectarea atipică a magistrelor de date și de adrese la memorie. Acest lucru decurge din nevoia de a asigura accesul subsistemului video la o parte din memorie concomitent cu accesul microprocesorului la o altă parte din memorie.

Magistrala de date este conectată direct la:

- ieșirile D0÷D7 ale memoriei ROM;
- la conexiunile DIN și DOUT ale celor 8 circuite de memorie dinamică 4532, care formează blocul de memorie RAM, între adresele 32768 și 65535; vom numi această memorie *RAM non-video*;

- prin intermediul a 8 rezistențe de 470 Ω , la conexiunile DIN și DOUT ale celor 8 circuite de memorie dinamică 4116, care formează blocul de memorie RAM între adresele 16384 și 32767; vom numi această memorie *RAM video*;

- tot prin intermediul celor 8 rezistențe, la intrările D0÷D7 ale ULA.

Acest aranjament, evidențiat în figura 2.3, este necesar pentru a separa, atunci când este nevoie, cele două blocuri de memorie RAM. Accesul la memorie al microprocesorului decurge după următorul protocol:

a. Dacă procesorul solicită acces la memoria ROM sau la RAM-ul non-video, atunci acesta se poate desfășura în paralel cu accesul ULA la RAM-ul video, cele două magistrale de date fiind separate prin rezistențe de 470 Ω și deci putând conține valori diferite. Curentul maxim prin rezistențele de separare care apare în acest caz este:

$$I_{\max} = \frac{4 \text{ V}}{470 \Omega} \approx 8,5 \text{ mA}$$

b. Dacă procesorul cere acces la RAM-ul video sau execută o operație de I/O prin ULA, atunci prioritatea maximă o are ULA deoarece afișarea informației pe ecran nu poate fi oprită. Ceasul procesorului este blocat până când magistrala de date devine liberă. Când accesul este permis, acesta se face prin intermediul rezistențelor de separare. Pentru a nu avea erori de interpretare ale nivelurilor logice 0 și 1, căderea maximă de tensiune pe aceste rezistențe nu trebuie să depășească 0,35 V, deci curentul maxim este:

$$I_{\max} = \frac{0,35 \text{ V}}{470 \Omega} \approx 0,74 \text{ mA}$$

Fiecare bit al magistralei de date este conectat la +5 V printr-o rezistență de valoare relativ mare, de 8,2 k Ω sau de 10 k Ω . Acest lucru este necesar pentru a reduce timpul de creștere al semnalului respectiv, memoriile fiind realizate în tehnologie MOS, precum și pentru a obține un vector de întrerupere bine definit.

Biții superiori ai magistralei de adrese, A14 și A15, sunt folosiți pentru a selecta diversele blocuri de memorie și sunt conectați direct la ULA. Biții A0÷A13 sunt conectați direct la adresele corespunzătoare ale circuitului ROM.

Pentru a genera semnalele necesare adresării memoriilor RAM, se folosesc două blocuri multiplexoare, realizate fiecare cu câte două circuite multiplexoare 2:1 74LS157. Astfel, din semnalele A0 și A7 ale microprocesorului se obține A0 pentru DRAM, din A1 și A8 se obține A1 etc.

Primul bloc multiplexor folosește A0 ÷ A13 și generează semnalele necesare pentru accesul la RAM-ul video. Deoarece acesta este adresat direct de către ULA, ieșirea din multiplexoare este conectată prin intermediul unor rezistențe de separare. Pentru ca adresa pe circuitele DRAM să fie cea generată de microprocesor, este necesar ca DRAM0 ÷ DRAM7 să fie în starea de impedanță înaltă, în caz contrar având prioritate adresa ULA. Semnalele de comandă necesare sunt asigurate în ambele cazuri de către ULA.

Celălalt bloc multiplexor folosește A0 ÷ A14 pentru accesul la RAM-ul non-video. Semnalele de control trebuie în acest caz generate de către un circuit logic separat.

Bitul A0 este folosit pentru validarea operațiilor de I/O prin ULA. Circuitul format din tranzistorul și rezistența de 680 Ω implementează funcția $TORQULA = TORQ + A0$, ceea ce face ca ULA să nu răspundă decât la adrese de *port* cu bitul A0 = 0. Adresa folosită de sistemul de operare este #FE = 254, însă schema permite adresarea cu orice adresă pară, de exemplu: 0, 2, 4 ș.a.m.d.

2.1.3. SEMNALELE DE COMANDĂ ȘI CONTROL

Schimbul de informații între microprocesor și memorie sau *port*-uri se face sub controlul unei set de semnale specializate. Toate aceste semnale sunt active în starea 0, fapt evidențiat și de notația cu bară deasupra. Nu toate semnalele oferite de Z80 sunt folosite în schema *Spectrum*-ului. Vom trece pe scurt în revistă unele semnale. Pentru informații suplimentare privind diagramele exacte de timp, recomandăm consultarea lucrărilor [27], [29], [37].

RD - Read, citire. Semnal generat de Z80. Indică direcția transferului pe magistrala de date. În ciclul-mașină curent, microprocesorul va citi date de pe magistrală, acestea fiind furnizate fie de memorie, dacă semnalul apare concomitent cu $\overline{MREQ}$, fie de un dispozitiv I/O, dacă apare concomitent cu $TORQ$. Semnalul este activ aproximativ 1,5 T. În momentul activării, magistrala de adrese se află într-o stare stabilă, care se menține cel puțin până când semnalul devine inactiv. Datele sunt citite pe frontul crescător al semnalului, ceea ce oferă maximum de rezervă de timp pentru stabilizarea lor. Semnalul RD este conectat direct la ULA și la memoria ROM și este folosit de către logica de adresare a memoriei RAM non-video.

$\overline{WR}$ - Write, scriere. Semnal generat de Z80. Indică faptul că în ciclul-mașină curent, microprocesorul va forța date pe magistrala de date pentru a scrie în memorie sau la un *port*. Semnalul este activ aproximativ 1 T. Atât datele cât și adresele sunt stabile pe magistrală cu aproape un tact înainte de activarea lui $\overline{WR}$ și se mențin neschimbate pe durata cât semnalul este activ.

$\overline{MREQ}$ sau $\overline{IORQ}$ sunt de asemenea active cu un tact mai devreme, dând astfel posibilitatea logicii de selecție să fie "gata" la activarea lui $\overline{WR}$. Semnalul este conectat direct la ULA și este folosit de logica de adresare a memoriei RAM non-video.

$\overline{MREQ}$ - *Memory Request*, cerere de acces la memorie. Semnal generat de Z80. Indică faptul că în ciclul-mașină curent, microprocesorul va accesa memoria. Însotăște în mod uzual $\overline{RD}$ sau $\overline{WR}$, durând în ambele cazuri 1,5 T. Magistrala de adrese trebuie să fie stabilă în momentul activării lui $\overline{MREQ}$. Semnalul devine inactiv o dată cu $\overline{RD}$ sau cu $\overline{WR}$. $\overline{MREQ}$ este de asemenea folosit împreună cu $\overline{RFSH}$ pentru a semnala ciclul de reîmprospătare (*refresh*). Semnalul este conectat direct la ULA și la memoria ROM și este folosit de logica de adresare a memoriei RAM non-video.

$\overline{IORQ}$ - *Input/Output Request*, cerere de intrare/ieșire. Semnal generat de Z80. Indică faptul că transferul de date se face cu un dispozitiv periferic și nu cu memoria. Joacă același rol ca $\overline{MREQ}$, pentru operațiile I/O. Durata sa normală este de aproximativ 2,5 T, ciclurile I/O având un tact suplimentar. Activat împreună cu $\overline{MI}$ semnalează faptul că ciclul-mașină curent este un răspuns la cererea de întrerupere, moment în care vectorul de întrerupere poate fi plasat pe magistrala de date. Semnalul este legat la ULA prin intermediul unei rezistențe de 680 Ω și, împreună cu A0, generează $\overline{IORQGE}$ pentru ULA. Operațiile I/O prin ULA pot fi astfel blocate prin legarea lui $\overline{IORQGE}$ la +5 V, semnalul fiind prezent la cupla de extensie (figura 2.4).

$\overline{RFSH}$ - *Refresh*, reîmprospătare. Semnal generat de Z80. Indică, prin activarea împreună cu $\overline{MREQ}$, inițierea unui ciclu de reîmprospătare pentru memoriile RAM dinamice. Ultimele două stări ale ciclului de extragere a codului instrucțiunii sunt folosite de microprocesor pentru decodificarea și eventual execuția instrucțiunii curente. În acest interval nu se mai efectuează operații de citire sau de scriere și prin urmare magistrala de adrese poate fi folosită pentru a asigura reîmprospătarea memoriilor dinamice. În intervalul în care $\overline{RFSH}$ este activ, cei mai puțin semnificativi 7 biți¹ ai magistralei de adrese reflectă conținutul unui registru intern al microprocesorului, care este incrementat la fiecare instrucțiune. De remarcat că bitul cel mai semnificativ al acestui registru, corespunzător lui A7, poate fi poziționat prin program pe 0 sau 1, numărătorul fiind modulo 128. Acest lucru este folosit de unele compatibile *Spectrum* pentru comanda modului de lucru. La acestea, semnalul este folosit doar pentru reîmprospătarea memoriei RAM video pe durata impulsului de sincronizare verticală. Pentru RAM-ul non-video se folosește $\overline{MREQ}$, logica de comandă fiind astfel proiectată încât să realizeze reîmprospătarea prin $\overline{MREQ}$, fără $\overline{WR}$ sau $\overline{RD}$.

$\overline{MI}$ - *Machine Cycle One*, primul ciclu-mașină. Semnal generat de Z80. Indică faptul că microprocesorul începe execuția unei noi instrucțiuni. Este activ

¹Z80 a fost proiectat în 1974, când cele mai mari memorii dinamice erau de 16 Kb, cu 7 biți de adresă.

Partea inferioară (cablaj)	A14	1	A15	Partea superioară (piese)
	A12	2	A13	
	+5V	3	D7	
	+9V	4	-	
		5		
	A	GND	6	D0
		GND	7	D1
		CLK	8	D2
		A0	9	D6
		A1	10	D5
		A2	11	D3
		A3	12	D4
		IORQGE	13	INT
		GND	14	NMI
		VIDEO	15	HALT
		Y	16	MREQ
		U	17	IORQ
		V	18	RD
		BUSRQ	19	WR
		RESET	20	-5V
		A7	21	WAIT
		A6	22	+12V
		A5	23	~12V
		A4	24	M1
		ROMCS	25	RFSH
		BUSACK	26	A8
		A9	27	A10
		A11	28	-

Fig. 2.4. Semnalele la cupla de extensie

variante de ULA, să aibă o altă valoare.

NMI - Non-Maskable Interrupt, întrerupere nemascabilă. Semnal de intrare prin care se solicită necondiționat serviciile microprocesorului. Nu poate fi dezactivat prin program și are prioritate față de **INT**. Spre deosebire de **INT**, care este activ pe nivelul 0, întreruperea nemascabilă este activă pe frontul căzător și este întotdeauna achitată. Semnalul nu este folosit la *Spectrum*, fiind menținut în 1 printr-o rezistență de 10 kΩ. Generarea unei astfel de întreruperi, posibilă prin conectarea semnalului de pe cupla de extensie la 0, inițializează calculatorul datorită unei greșeli voluntare în rutina de tratare.

RESET - inițializare. Semnal de intrare prin care se inițializează microprocesorul. Pentru a asigura o inițializare corectă a microprocesorului,

pe primele două stări ale acestui ciclu. Activat împreună cu **IORQ** semnalează un ciclu de achitare a cererii de întrerupere. Nu este folosit în *Spectrum*.

INT - Interrupt Request, cerere de întrerupere. Semnal de intrare prin care un dispozitiv periferic solicită serviciile microprocesorului. Poate fi dezactivat prin program. Dacă întreruperile sunt activate, atunci microprocesorul va răspunde la sfârșitul instrucțiunii curente printr-un ciclu special de achitare a întreruperii. *Spectrum*-ul folosește modul 1 de întreruperi, care nu presupune un vector de întrerupere. Semnalul **INT** este generat de ULA la începutul impulsului de stingere pe verticală a spotului, având deci o frecvență de 50 Hz¹ și este aplicat microprocesorului printr-o rezistență de 680 Ω. Durata sa este de aproximativ 8 ÷ 10 T. Folosirea rezistenței permite blocarea semnalului de întrerupere prin conectarea sa la +5 V prin cupla de extensie. Pentru aplicațiile care trec procesorul în modul 2 de întreruperi, adresa furnizată ca vector de întrerupere nu este garantată. Ea este în mod uzual #FF la majoritatea calculatoarelor *Spectrum*, însă este posibil ca periodic, pentru anumite

¹ Compatibilele *Spectrum* livrate în Statele Unite produc imagine *NTSC* și au 60 de întreruperi pe secundă.

semnalul trebuie să dureze cel puțin 4 T. Un circuit RC asigură generarea automată la punerea sub tensiune a unui $\overline{\text{RESET}}$. Inițializarea se mai poate obține prin conectarea la masă, pentru scurt timp, a semnalului corespunzător din conectorul de extensie (figura 2.4).

$\overline{\text{HALT}}$ - *Halt State*, oprirea procesorului. Semnal generat de Z80 prin care indică faptul că a executat o instrucțiune $\overline{\text{HALT}}$ și stă în așteptarea unei întreruperi mascabile sau nemascabile. Nu este folosit în *Spectrum*.

$\overline{\text{WAIT}}$ - așteptare. Semnal de intrare prin care microprocesorul se poate sincroniza cu memoriile sau perifericele mai lente. Are ca efect prelungirea operațiilor de citire și de scriere. Nu este folosit de *Spectrum*, fiind menținut la 1 printr-o rezistență de 1,5 k Ω .

$\overline{\text{BUSRQ}}$ - *Bus Request*, cerere de magistrală. Semnal de intrare prin care se cere microprocesorului să cedeze magistralele de date, de adrese și o parte din semnalele de control. Nu este folosit de *Spectrum*, fiind menținut la 1 printr-o rezistență de 1 k Ω .

$\overline{\text{BUSAK}}$ - *Bus Acknowledge*, confirmare a cedării de magistrală. Semnal generat de Z80 ca răspuns la $\overline{\text{BUSRQ}}$. Nu este folosit de *Spectrum*.

2.2. MEMORIA

2.2.1. MEMORIA ROM

Memoria nevolatilă a calculatorului *Spectrum* constă dintr-un singur circuit ROM cu capacitatea de 16 kB, care conține interpretorul Basic. Circuitul are 28 de terminale a căror semnificație este dată în figura 2.5 (IC5). S-au folosit două tipuri de memorii ROM: NEC și Hitachi. Cele două circuite sunt echivalente cu excepția a două semnale care sunt inversate. La circuitele ROM NEC, conexiunea 20 este $\overline{\text{CE}}$ și conexiunea 27 este $\overline{\text{CS}}$, iar la cele de fabricație Hitachi, conexiunea 20 este $\overline{\text{CS}}$, iar conexiunea 27 este $\overline{\text{CE}}$. Pentru a adresa 16.384 locații de memorie de 8 biți, se folosesc 14 conexiuni de adresă ($2^{14}=16.384$) și 8 conexiuni de date, care sunt legate direct la conexiunile corespunzătoare ale microprocesorului. Din punctul de vedere al adresării, ROM-ul ocupă zona de memorie de la 0 la 16383 (#3FFF) inclusiv, deci selecția sa se face cu $\text{A14}=0$ și $\text{A15}=0$. Semnalul de selecție, $\overline{\text{ROMCS}}$ este generat de ULA din cei doi biți superiori de adresă, A14 și A15, și $\overline{\text{RDD}}$. El este aplicat circuitului prin intermediul rezistenței R33, de 680 Ω . Pentru a evita activarea memoriei la operațiile de intrare-ieșire, intrarea $\overline{\text{OE}}$ de validare a datelor este conectată la $\overline{\text{MREQ}}$. Intrarea de validare $\overline{\text{CE}}$ este conectată la $\overline{\text{RD}}$. Prezența rezistenței de separare face posibilă invalidarea circuitului ROM din calculator, prin legarea la +5 V a semnalului corespunzător de pe conectorul de extensie și folosirea unei memorii nevolatile exterioare.

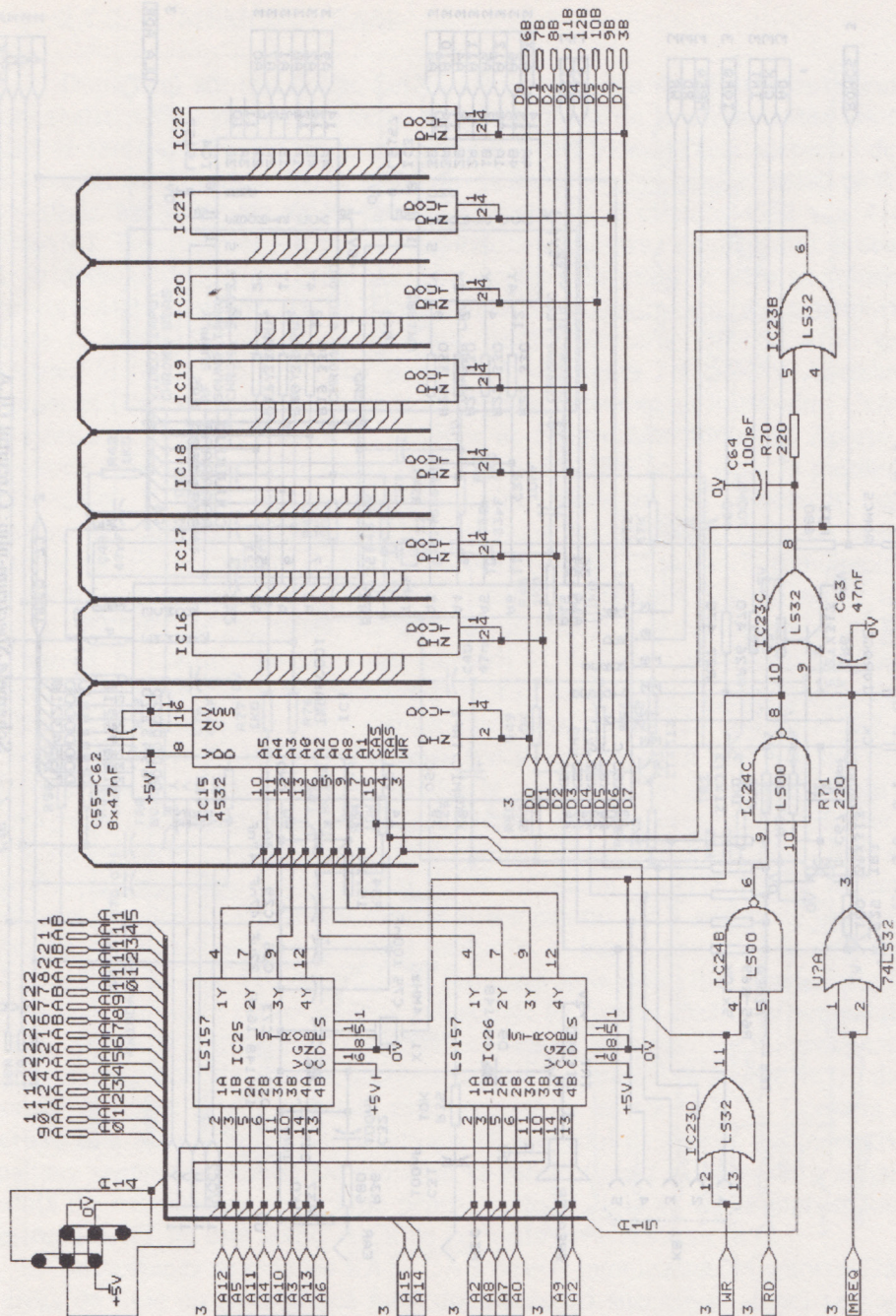


Fig. 2.6. Schemele Spectrum-ului: Memoria RAM

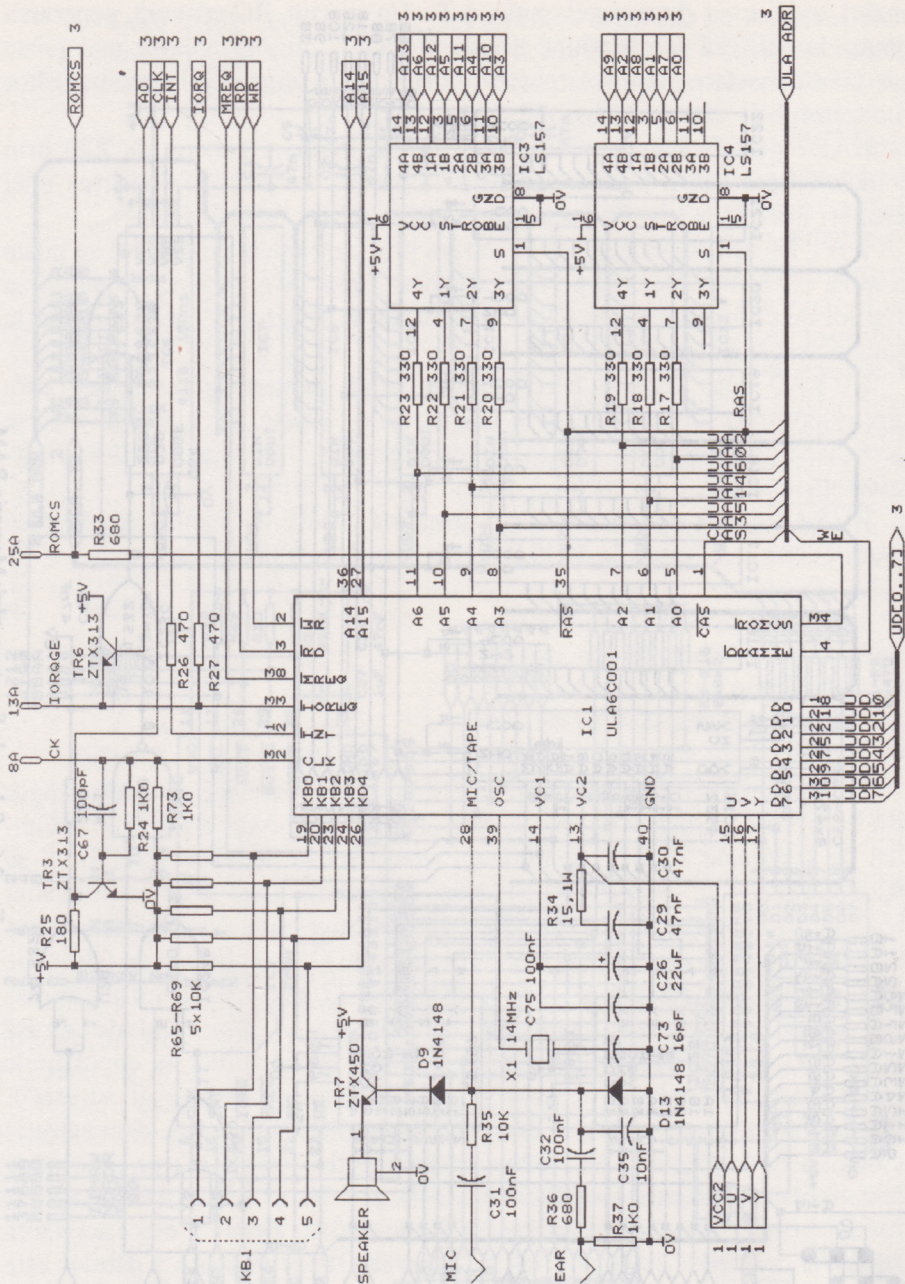


Fig. 2.7. Schemele *Spectrum*-ului: Circuitul ULA

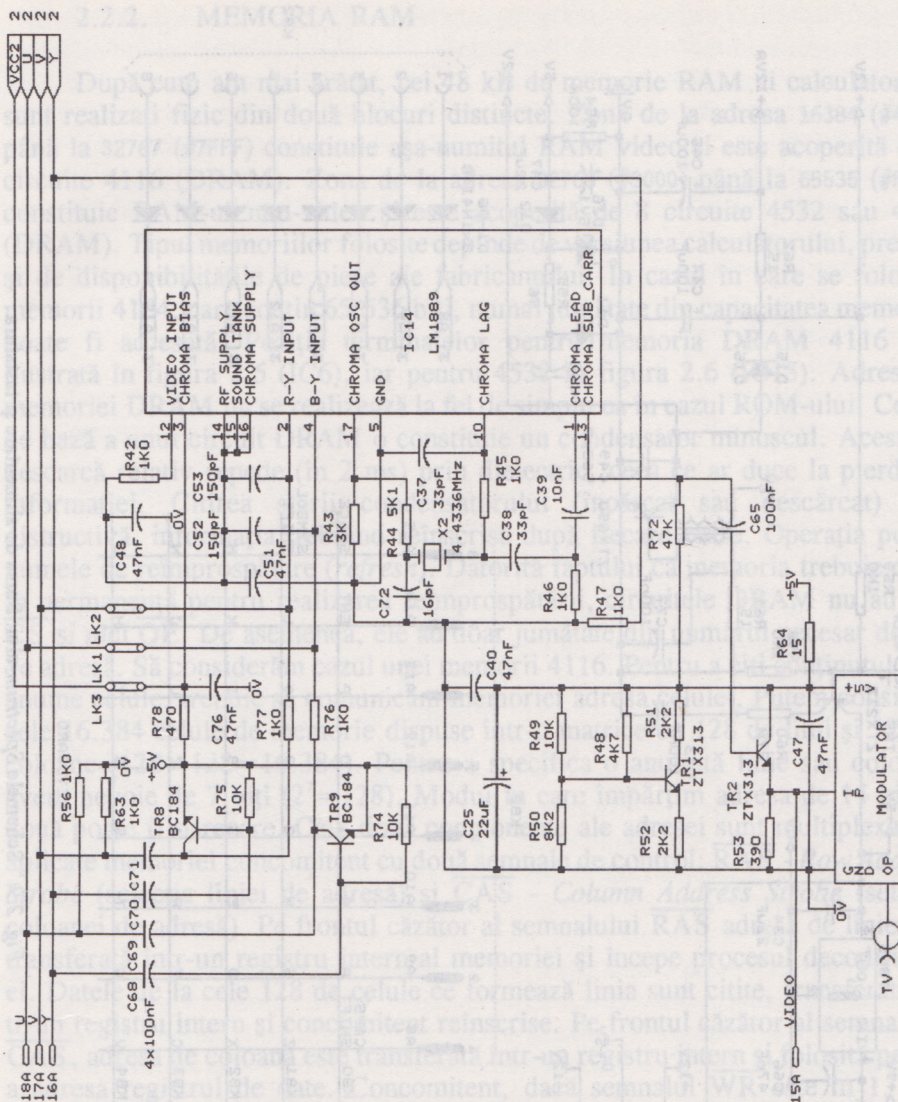


Fig. 2.8. Schemele Spectrum-ului: Codorul PAL

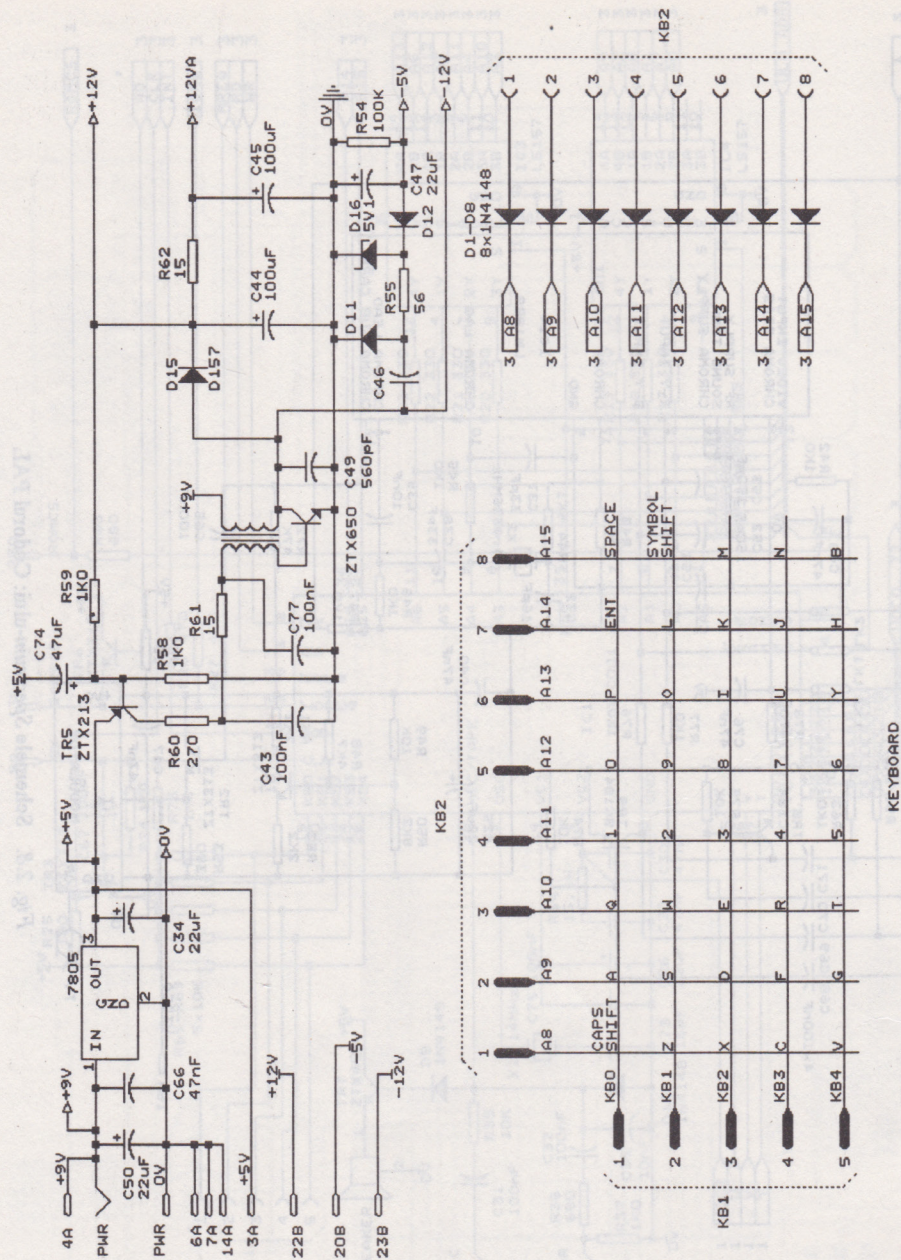


Fig. 2.9. Schemele Spectrum-ului: Sursa de alimentare și tastatura

2.2.2. MEMORIA RAM

După cum am mai arătat, cei 48 kB de memorie RAM ai calculatorului sunt realizați fizic din două blocuri distincte. Zona de la adresa 16384 (#4000) până la 32767 (#7FFF) constituie așa-numitul RAM video și este acoperită de 8 circuite 4116 (DRAM). Zona de la adresa 32768 (#8000) până la 65535 (FFFF) constituie RAM-ul non-video și este acoperită de 8 circuite 4532 sau 4164 (DRAM). Tipul memoriilor folosite depinde de versiunea calculatorului, precum și de disponibilitățile de piese ale fabricantului. În cazul în care se folosesc memorii 4164, care conțin 65.536 biți, numai jumătate din capacitatea memoriei poate fi accesată. Poziția terminalelor pentru memoria DRAM 4116 este ilustrată în figura 2.5 (IC6), iar pentru 4532 în figura 2.6 (IC15). Adresarea memoriei DRAM nu se realizează la fel de simplu ca în cazul ROM-ului. Celula de bază a unui circuit DRAM o constituie un condensator minuscul. Acesta se descarcă relativ repede (în 2 ms) prin dielectric, ceea ce ar duce la pierderea informației. Citirea stării condensatorului (încărcat sau descărcat) este distructivă, informația trebuind reînscrisă după fiecare citire. Operația poartă numele de reîmprospătare (*refresh*). Datorită faptului că memoria trebuie citită în permanență pentru realizarea reîmprospătării, circuitele DRAM nu au nici $\overline{CS}$ și nici $\overline{OE}$. De asemenea, ele au doar jumătate din numărul necesar de biți de adresă. Să considerăm cazul unei memorii 4116. Pentru a citi conținutul unei anume celule, trebuie să comunicăm memoriei adresa celulei. Putem considera cele 16.384 celule de memorie dispuse într-o matrice cu 128 de linii și 128 de coloane ($128 \times 128 = 16.384$). Pentru a specifica o anumită linie sau coloană, avem nevoie de 7 biți ($2^7 = 128$). Modul în care împărțim adresa de 14 biți în două poate fi oarecare. Cele două componente ale adresei sunt multiplexate și aplicate memoriei concomitent cu două semnale de control: $\overline{RAS}$ - *Row Address Strobe* (selecția liniei de adresă) și $\overline{CAS}$ - *Column Address Strobe* (selecția coloanei de adresă). Pe frontul căzător al semnalului $\overline{RAS}$ adresa de linie este transferată într-un registru intern al memoriei și începe procesul decodificării ei. Datele de la cele 128 de celule ce formează linia sunt citite, transferate într-un registru intern și concomitent reînscrise. Pe frontul căzător al semnalului $\overline{CAS}$, adresa de coloană este transferată într-un registru intern și folosită pentru a adresa registrul de date. Concomitent, dacă semnalul $\overline{WR}$ este în 1, data corespunzătoare celulei selectate este prezentă la ieșirea DOUT. Operația de scriere este asemănătoare, cu observația că semnalul de scriere trebuie activat înaintea semnalului $\overline{CAS}$. De remarcat că este suficient să accesăm o singură celulă de memorie de pe o linie pentru ca toată linia să fie reîmprospătată. Ieșirea DOUT nu este activă decât pe perioada cât $\overline{CAS}$ este în 0.

Sucesiunea de semnale $\overline{RAS}$, $\overline{CAS}$ nu este întotdeauna obligatorie. Dacă adresa de linie nu se modifică între două accesări succesive, atunci se poate economisi timpul necesar pentru multiplexarea celei de-a doua adrese de linie prin succesiunea $\overline{RAS}$, $\overline{CAS}_1$, $\overline{CAS}_2$. Acest mod de adresare este folosit în calculatorul *Spectrum* de ULA pentru generarea imaginii, vezi § 2.3.1.

Multiplexarea adresei se face separat pentru fiecare din cele două blocuri, folosindu-se circuite 74LS157 (4 multiplexoare 2:1). Generarea întârzierilor necesare între semnalele MUX, RAS și CAS și se realizează de ULA pentru RAM-ul video și cu ajutorul unor circuite RC pentru RAM-ul non-video. Semnalele DOUT și DIN ale RAM-ului video sunt conectate direct la ULA și, prin intermediul rezistențelor de 470 Ω , la magistrala de date a procesorului. Cei mai puțin semnificativi 14 biți ai magistralei de adrese ($A0 \div A13$) sunt multiplexați folosind IC3 și IC4 (figura 2.7) și sunt conectați la blocul de memorie video prin rezistențele $R17 \div R23$ de 330 Ω . Pentru ca procesorul să schimbe date cu memoria video sunt necesare 3 condiții:

- adresele $DRAM0 \div DRAM6$ ale ULA să fie în starea de impedanță înaltă, altfel adresa ULA este prioritară;
- ieșirile $D0 \div D7$ ale ROM-ului să fie în starea de impedanță înaltă;
- ieșirile DOUT ale RAM-urilor non-video să fie în starea de impedanță înaltă.

După cum am arătat în § 2.1, datorită separării prin rezistențe, este posibil accesul paralel și simultan la cele două blocuri de memorie.

Semnalele DOUT și DIN ale RAM-urilor non-video sunt conectate direct la magistrala de date a procesorului. Adresele $A0 \div A14$ sunt multiplexate folosind IC25 și IC26 (figura 2.6) și apoi sunt aplicate direct blocului de memorie. De menționat că, având o capacitate de 32.768 biți, memoriile 4532 pot fi organizate fie cu 8 linii și 7 coloane, fie cu 7 linii și 8 coloane. Pentru a putea folosi memoria provenind de la fabricanți diferiți (cele mai multe calculatoare sunt echipate cu circuite Texas Instruments RAM TMS 4532 sau cu OKI RAM MSM3732), pe placă este prevăzută o selecție care permite conectarea lui $A14$ fie ca adresă de linie, fie ca adresă de coloană. De remarcat că memoriile de 32 kb sunt de fapt circuite de 64 kb care au o jumătate defectă. Pentru a selecta jumătatea funcțională, adresa superioară trebuie multiplexată fie cu 0, fie cu 1, în funcție de tipul circuitului folosit (-3/-4 pentru TI sau -H/-L pentru OKI).

Asigurarea reîmprospătării se face diferit pentru cele două blocuri. Pentru memoria video, accesul ULA la informația necesară pentru afișarea imaginii este suficient atât timp cât durează zona activă a ecranului. Durata *paper*-ului nu reprezintă însă decât 2/3 din durata unui semicadru. Pentru restul intervalului, la unele compatibile *Spectrum* se folosește semnalul $\overline{RFSH}$ al microprocesorului.

Pentru RAM-ul non-video, reîmprospătarea informației este asigurată de microprocesor prin intermediul unui mecanism propriu, nefolosind semnalul $\overline{RFSH}$, ci $\overline{MREQ}$ neînsoțit de $\overline{RD}$ sau $\overline{WR}$.

2.3. CIRCUITUL ULA

Circuitul Ferranti ULA a fost produs în mai multe variante. Versiunea 1 era echipată cu Ferranti ULA 5C102. Acesta avea un defect de proiectare,

remediat cu ajutorul unui 74LS00 montat pe un mic cablaj suplimentar. Varianta următoare a circuitului ULA, 5C112, nu mai necesită acest montaj, însă are o problemă cu generarea corectă a semnalului video color, motiv pentru care imaginile obținute pe anumite tipuri de receptoare TV nu sunt de cea mai bună calitate. Validarea lui TORQ de la procesor se face folosind un circuit SAU cu diode. Încă de la versiunea 2 s-a introdus o a treia variantă de circuit ULA, și anume Ferranti 6C001 (figura 2.7). Acesta rezolvă problemele menționate, dar introduce o descentrare a poziției *paper*-ului pe ecran, având ca efect obținerea unui *border* asimetric. Validarea lui TORQ se face folosind un circuit SAU cu un tranzistor (TR6 din figura 2.7).

ULA asigură toate operațiile de intrare/ieșire ale sistemului. Funcțiile sale sunt:

- generator de semnal video;
- *port* pentru citirea tastaturii;
- *port* pentru citirea și scrierea pe casetă;
- *port* pentru generarea sunetului.

2.3.1. GENERAREA SEMNALULUI VIDEO

Afișarea imaginii pe ecranul TV se realizează prin baleierea orizontală a ecranului cu un fascicul de electroni, începând din colțul din stânga-sus. Imaginea este compusă prin urmare din "fâșii" foarte subțiri, denumite *linii*. *Semnalul video complex* conține două tipuri de informație: informația de imagine efectivă și informația de control care organizează această informație pe ecran: *semnalul de sincronizare*. Acesta din urmă este format la rândul său din două componente: *semnalul de sincronizare cadre (sincro-vertical)* care specifică momentul în care fasciculul trebuie să înceapă baleierea și *semnalul de sincronizare linii (sincro-orizantal)* care marchează începutul fiecărei linii. În afară de acestea, semnalul video color mai cuprinde și o a treia componentă necesară sincronizării informației de culoare, denumită *burst*.

Imaginea este afișată cu o frecvență de 25 de cadre/s, dar pentru a evita fenomenul de pâlpâire, un cadru este format din două semicadre întrețesute. Frecvența semnalului de sincro-vertical este deci 50 Hz. Fiecare cadru este format din 625 de linii, câte 312,5 pentru fiecare semicadru. Rezultă durata nominală a unei linii care este:

$$\frac{1}{625 \cdot 25 \text{ Hz}} = 64 \mu\text{s}$$

(frecvența de linii este 15.625 Hz). Nu toată durata unei linii este folosită pentru afișarea pe ecran, deoarece o parte este necesară întoarcerii fasciculului din partea dreaptă în partea stângă. Evident că, în acest interval, fasciculul trebuie "stins", motiv pentru care impulsul respectiv poartă numele de *stingere* orizontală. Durata sa este de 12 μs , deci durata activă a unei linii este de 52 μs . De

asemenea, nu toate liniile sunt active, datorită faptului că este necesară întoarcerea spotului pe verticală. Pentru aceasta se folosesc 25 de linii pe fiecare semicadru, perioadă care poartă numele de *stingere verticală*. Structura semnalului de stingere verticală este: 2,5 linii impuls de egalizare + 2,5 linii sincro-semicadre + 20 linii stingere. Trebuie menționat că *Spectrum*-ul nu asigură o baleiere întrefesută a imaginii și nici o structură foarte corectă a semnalelor de sincronizare. Cu toate acestea, marea majoritate a receptoarelor TV nu au probleme la afișarea imaginii.

Semnalul video complex color este reprezentat în figura 2.10. Nivelurile de tensiune folosite pentru diversele componente au fost reprezentate în procente. După cum se remarcă, nivelul de tensiune corespunzător pentru negru este mai mic decât cel pentru alb (semnalul este video-pozitiv). Considerând nivelul de sincronizare 0% și cel de alb 100%, avem situația din tabelul 2.1.

TABELUL 2.1. Nivelurile procentuale ale componentelor semnalului video-complex

Nivelul	Valoarea în procente
nivel de alb	100
nivel de negru	$30 \div 35$
nivel de stingere	30
nivel de sincronizare	0

Generarea imaginii color se face folosind trei fascicule de electroni ce baleiază simultan ecranul. Fiecare fascicul reprezintă una dintre culorile fundamentale, roșu (R), verde (G) și albastru (B), și este destinat tipului respectiv de luminofor ce se află depus pe ecran. Prin însumarea culorilor fundamentale se produce în final imaginea color. Pentru generarea acesteia avem deci nevoie de cele trei componente R, G și B. Din considerente legate de compatibilitatea cu semnalul de televiziune alb-negru și lărgimea de bandă a semnalului video, cele trei componente nu se transmit în această formă. Prin însumarea lor ponderată se obțin alte trei semnale¹:

– luminanța: $Y = 0,3 R + 0,59 G + 0,11 B$ (ponderea diferită a culorilor în semnalul de luminanță este datorată faptului că ochiul nu este sensibil în mod egal la componentele spectrului);

– două semnale de diferență de culoare: $U = 0,493 (B - Y)$ și $V = 0,877 (R - Y)$ (reducerea amplitudinii semnalelor de diferență de culoare se face pentru a evita supramodularea subpurătoarei de culoare).

Semnalele de diferență de culoare modulează în amplitudine două subpurătoare obținute din același oscilator de 4,43361875 MHz, defazate între

¹Explicațiile din text sunt valabile pentru sistemul PAL.

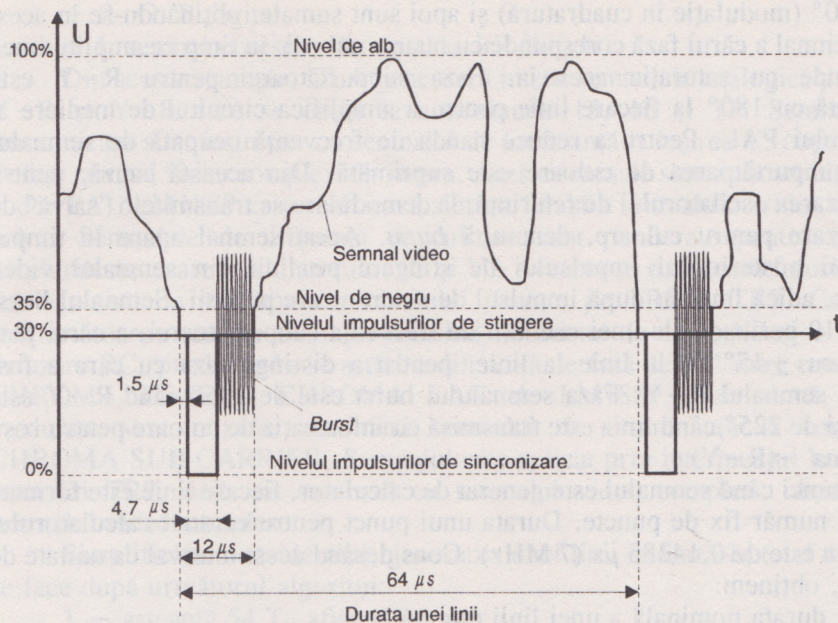


Fig. 2.10. Semnalul video-complex color

Bit					Bit						
D0	D1	D2	D3	D4	D4	D3	D2	D1	D0		
Port 63486	1	2	3	4	5	6	7	8	9	0	Port 61438
Port 64510	Q	W	E	R	T	Y	U	I	O	P	Port 57342
Port 65022	A	S	D	F	G	H	J	K	L	ENTER	Port 49150
Port 65278	CS	Z	X	C	V	B	N	M	SS	SPACE	Port 32766
-1 -2 -4 -8 -16					-16 -8 -4 -2 -1						

Valorile de mai sus trebuie scăzute din 255 pentru testarea apăsării acelei taste.

Exemplu: pentru testarea apăsării simultane a tastelor 1 și 4 se face IN 63486,255-1-8.

Fig. 2.11. Harta port-urilor de citire directă a tastaturii

ele cu 90° (modulație în cuadratură) și apoi sunt sumate, obținându-se în acest fel un semnal a cărui fază corespunde cu nuanța culorii, în timp ce amplitudinea corespunde cu saturația acesteia. Faza subpurtătoarei pentru R-Y este schimbată cu 180° la fiecare linie pentru a simplifica circuitul de mediere al decodorului PAL. Pentru a reduce banda de frecvență ocupată de semnalul video, subpurtătoarea de culoare este suprimată. Din această cauză, pentru sincronizarea oscilatorului de referință, la demodulare se transmite o "salvă" de sincronizare pentru culoare, denumită *burst*. Acest semnal apare în timpul palierului posterior al impulsului de stingere pe linii din semnalul video complex, adică imediat după impulsul de sincronizare pe linii. Semnalul burst conține 10 perioade ale unei oscilații de frecvență subpurtătoarei, a cărei fază variază cu $\pm 45^\circ$ de la linie la linie, pentru a distinge faza cu care a fost modulată semnalul R-Y. Faza semnalului burst este de 135° când R-Y este pozitiv și de 225° când linia este transmisă cu informația de culoare pentru roșu sub forma $-(R-Y)$.

Atunci când semnalul este generat de calculator, fiecare linie este formată dintr-un număr fix de puncte. Durata unui punct pentru ecranul calculatorului *Spectrum* este de $0,14285 \mu s$ (7 MHz). Considerând acest interval ca unitate de timp T_v , obținem:

- durata nominală a unei linii este:

$$64 \mu s \cdot 7 \text{ MHz} = 448 T_v$$

- din care durata cursei de întoarcere este:

$$12 \mu s \cdot 7 \text{ MHz} = 84 T_v$$

Un semicadru este format din 312 linii nominale, din care 25 sunt folosite pentru stingerea verticală.

Din cele $364 T_v$ disponibile, numai $256 T_v$ sunt folosite efectiv pentru afișarea informației, restul de $108 T_v$ constituind *border*-ul. Pentru ca acesta să fie dispus simetric, este necesar să lăsăm jumătate în stânga *paper*-ului și jumătate în dreapta sa, deci câte $54 T_v$. Fiecare din cei 256 de pixeli ai *paper*-ului corespunde stării unui bit din memoria RAM video. Pentru a afișa un pixel, trebuie să știm dacă el este "aprins" sau "stins", precum și culorile corespunzătoare acestor două stări.

Calculatorul *Spectrum* este capabil să afișeze 16 culori¹ (cele 8 culori disponibile pot avea două nuanțe, în funcție de bitul de strălucire). Pentru a sintetiza semnalul video complex, fiecare din componentele sale este obținută digital, iar apoi se realizează o sumă ponderată. Semnalele digitale necesare sunt:

- sincro-orizontal;
- sincro-vertical;

¹De fapt numai 15 culori, deoarece negrul nu este influențat de strălucire.

– R, G, B digital;

– I, semnal corespunzător strălucirii (*bright*).

Din aceste semnale, ULA generează intern semnalele analogice pentru Y, Y–R și Y–B care se obțin la conexiunile 15, 16 și 17. Semnalele de sincronizare sunt însumate cu semnalul de luminanță la ieșirea Y, obținându-se de fapt semnal video complex alb-negru. Aceste semnale sunt aplicate circuitului codor PAL LM1889 care realizează sinteza finală a semnalului video complex color. Semnalele de diferență de culoare sunt ponderate pentru a obține nivelurile necesare modulatorului (U și V). Aceasta se realizează folosind două rețele active de schimbare a nivelului, bazate pe TR8 și TR9. Cele două subpurtătoare necesare se obțin folosind un cristal de cuarț X2 și o rețea de defazare RC, producându-se astfel o diferență de fază de 90° între conexiunile CHROMA LEAD și CHROMA LAG ale LM1889. Cele două semnale modulate sunt mixate, rezultând semnalul de cromaticitate la conexiunea CHROMA SUB-CARRIER. Semnalul este aplicat prin intermediul lui C65 pe baza lui TR2, unde se sumează cu semnalul video complex, obținându-se semnalul video complex.

Simplificând considerabil, generarea unei linii durează în total $448 T_v$ și se face după următorul algoritm:

1 – așteaptă $54 T_v$ afișând *border*;

2 – citește din memorie 2 octeți reprezentând 8 pixeli și attributele corespunzătoare ($0 T_v$);

3 – afișează cei 8 pixeli, unul câte unul (serializare); fiecare pixel durează $1 T_v$ ($32 \times 8 T_v$);

4 – dacă nu s-au terminat cei 256 pixeli (32 de cicluri), se reia de la pasul 2 ($0 T_v$);

5 – așteaptă $54 T_v$ afișând *border*;

6 – activează semnalul sincro-orizantal (semnalul este activ în 0) și așteaptă $84 T_v$;

7 – dezactivează sincro-orizantal și reia de la pasul 1 ($0 T_v$).

Datorită faptului că pasul 2 practic nu se execută instantaneu, algoritmul nu este implementabil direct. Pentru a avea datele pregătite atunci când este nevoie de ele, se impune suprapunerea pasului 2 fie peste pasul 1 pentru primul octet serializat dintr-o linie, fie peste pasul 3 pentru ceilalți octeți, deci citirea datelor din memorie se face înainte de terminarea serializării octetului curent.

Culoarea *border*-ului este controlată de biții D0, D1 și D2 ai *port*-ului 254 (#FE), interpretat de ULA. O valoare 111₂ produce un *border* alb, iar o valoare 000₂, unul negru. Se observă că nu putem avea decât $2^3 = 8$ culori diferite.

Pe verticală, din cele 287 de linii disponibile, numai 192 sunt folosite pentru *paper*, deci rămân 95 de linii pentru *border*, repartizate sus și jos în mod egal.

Modul în care sunt generate aceste semnale de control și adresele de încărcare este relativ simplu. Se pleacă de la frecvența oscilatorului de 14 MHz, care este divizată cu 2. Se obține astfel ceasul de serializare video de 7 MHz, cu factorul de umplere 1/2. Prin divizare cu 448 se obține semnalul de

sincro-orizantal și apoi, prin divizare cu 312, cel de sincro-vertical. Prin decodificarea ieșirilor numărătoarelor se obțin adresele de încărcare și forma necesară a semnalelor de sincronizare. Datorită implementării LSI, mecanismul intern al circuitului ULA este ceva mai greu de identificat cu precizie.

2.3.2. CITIREA TASTATURII

La fiecare întrerupere, deci o dată la 20 ms, sistemul verifică dacă o tastă este apăsată sau nu. Tastatura constă dintr-o matrice de 5 linii și 8 coloane. La intersecția fiecărei linii cu o coloană se află un contact normal deschis. Cele 8 coloane sunt conectate prin intermediul unui cablu plat KB2 la biții superiori ai magistralei de adrese, iar cele cinci linii, prin KB1 la ULA. Cinci rezistențe de *pull-up*, R64 ÷ R68, mențin intrările KB0 ÷ KB4 ale ULA în 1 atât timp cât nici o tastă nu este apăsată. Rutina de baleiere a tastaturii execută instrucțiuni succesive de IN de la adrese care au ca octet inferior adresa ULA (#FE). Jumătatea superioară a magistralei de adrese se modifică în așa fel încât fiecare din liniile A15 până la A8 să fie 0 pe rând, celelalte linii rămânând în 1.

Secvența de baleiere începe prin IN de la adresa #FEFE (A8=0, restul 1). Dacă una dintre tastele de pe coloana lui A8 este apăsată, atunci linia corespunzătoare va fi 0, altfel va fi 1. Dacă este apăsată o tastă de pe o altă coloană, atunci valoarea liniei respective va fi 1. Deci o linie va fi 0 numai dacă există o tastă apăsată și numai dacă adresa corespunzătoare coloanei respective este 0. Secvența se termină cu adresa #7FFE (în total opt instrucțiuni IN). Știind adresa care a fost 0 (coloana) și citind biții D0 ÷ D4 (linia), rutina de baleiere poate afla care tastă (sau combinație de taste) a fost apăsată. Adresele *port*-urilor pentru citirea directă a tastaturii sunt date în figura 2.11.

Diodele D1 ÷ D8 asigură separarea magistralei de adrese de tastatură, evitând astfel adresări eronate când se apasă o tastă.

2.3.3. GENERAREA SUNETULUI ȘI INTERFAȚA DE CASETO FON

Calculatorul *Spectrum* are un generator de sunet foarte simplu, constând din conectarea la un difuzor a bitului D4 din *port*-ul de ieșire cu adresa 254 (#FE). Sunetul este produs prin comutarea stării acestui bit la intervale regulate de timp.

Circuitul ULA folosește o singură linie (28 MIC/TAPE) pentru interfața cu difuzorul și casetofonul, conexiune atât cu rol de intrare, cât și de ieșire. Acest lucru este posibil datorită faptului că operațiile de LOAD și SAVE se exclud reciproc. Direcția transferului de date pe această conexiune depinde de ultima operație I/O cu *port*-ul 254 (#FE). Dacă aceasta a fost IN, atunci conexiunea respectivă este intrare, iar dacă a fost OUT, atunci conexiunea este ieșire. În timpul operației de încărcare de pe casetă, microprocesorul execută instrucțiuni

În citind intrarea EAR prin intermediul bitului D6 al *port*-ului. La salvarea pe casetă, microprocesorul execută instrucțiuni OUT, starea conexiunii fiind dictată de bitul D3 (D4 este în acest caz 0).

Înregistrarea magnetică a datelor se face în codul Manchester II. Un exemplu de semnal este reprezentat în figura 2.12.

Tensiunea la ieșirea de sunet a circuitului ULA depinde de combinația de biți D3–D4, după cum se arată în tabelul 2.2.

TABELUL 2.2. Tensiunea la ieșirea de sunet a circuitului ULA

D3	D4	$V_{ULA\ 28}$
0	0	0,2 V ÷ 0,75 V
1	0	1,3 V
0	1	3,3 V
1	1	3,5 V

Difuzorul cu impedanța de 40 Ω este conectat la MIC/TAPE prin tranzistorul TR7, folosit ca repetor pe emitor și dioda D9. Pentru a obține tensiune pe bornele difuzorului este necesară deschiderea a două joncțiuni polarizate direct (bază-emitor TR7 și D9), ceea ce necesită aproximativ 1,4 V. Din această cauză, la salvarea pe casetă, difuzorul este practic blocat. În schimb, când se execută o instrucțiune BEEP, tensiunea pe difuzor ajunge la 3,5 V – 1,4 V = 2,1 V și se produc sunete.

La încărcare, dacă tensiunea pe intrarea MIC depășește 1,4 V, difuzorul este de asemenea activat, permițând astfel o apreciere calitativă a nivelului semnalului de la casetofon. Pentru a nu distruge ULA prin aplicarea unei tensiuni negative mari, conexiunea MIC/TAPE este protejată folosind dioda D13, care se deschide dacă tensiunea este mai mică decât –0,7 V.

În concluzie, ULA răspunde la adresa I/O 254 (#FE), semnificațiile biților de date fiind următoarele:

Ieșire

D0, D1, D2 – culoarea *border*-ului (000=negru, 001=albastru ...

111=alb);

D3 – ieșire casetofon;

D4 – difuzor;

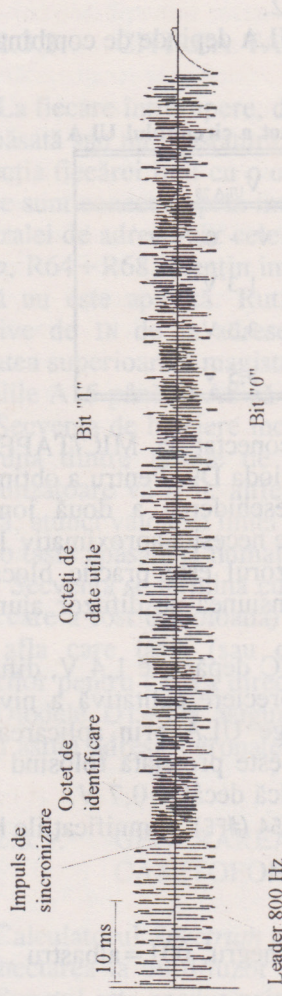
D5, D6 și D7 – nefolosiți.

Intrare

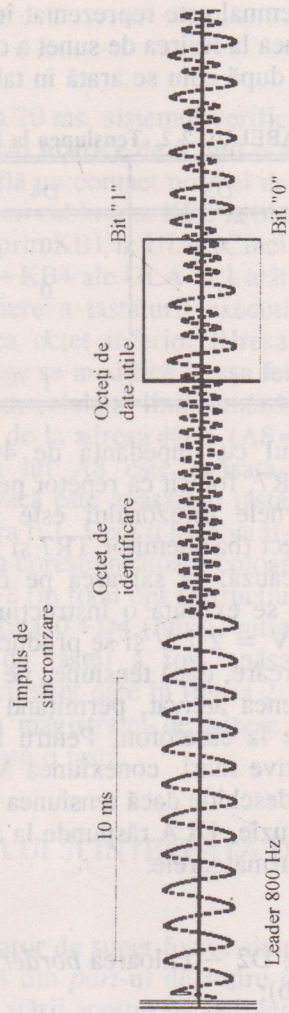
D0, D1, D2, D3, D4 – citire tastatură (0=tastă apăsată, 1=tasta neapăsată);

D6 – intrare casetofon;

D5 și D7 – nefolosiți, cu valoarea 1.



a. Forma semnalului înregistrat al unui header de fișier



b. Începutul header-ului. În medalion se observă codificarea informației "01001000"

Fig. 2.12. Semnalul de înregistrare magnetică a datelor

2.4. SURSA DE ALIMENTARE

Calculatorul primește energie de la un alimentator extern, care livrează 9 V la 1,5 A, redresat bialternanță și filtrat, dar nestabilizat. Alimentatorul este format dintr-un transformator coborâtor de tensiune, o punte redresoare și un condensator de 2200 μ F. Tensiunile stabilizate, necesare calculatorului, sunt următoarele:

- +5 V / 1 A pentru circuitele integrate logice, ULA și modulator;
- 5 V pentru alimentarea memoriilor dinamice de 16 kb;
- +12 V pentru alimentarea memoriilor dinamice de 16 kb și pentru codorul PAL.

Ele se obțin pe placa de bază folosind un circuit stabilizator integrat 7805, care furnizează +5 V stabilizat și un invertor format din TR4 și TR5. Acestea formează un oscilator de impulsuri dreptunghiulare pe colectorul lui TR4. Oscilațiile sunt redresate și filtrate de grupul D5-C44, producând +12 V pentru alimentarea circuitelor RAM dinamice de 16 kb. O filtrare suplimentară cu R62-C45, necesară pentru a elimina zgomotul electric introdus de comutarea memoriilor, produce tensiunea necesară pentru alimentarea codorului PAL. Tot tensiunea de pe colectorul lui TR4 este folosită și pentru generarea tensiunii -5 V, prin grupul de diode D16÷D12 (D16 este diodă Zener) și este de asemenea disponibilă pe conectorul de extensie, pentru alimentarea perifericelor.

2.5. METODE DE EXTINDERE A MEMORIEI INTERNE

Deși fabricat la început cu doar 16 kB de memorie RAM, s-a ajuns repede ca prin configurația standard să se înțeleagă un *Spectrum* cu 48 kB de RAM, acoperindu-se astfel întreg spațiul de adresare al lui Z80. Creșterea mai departe a memoriei RAM nu este posibilă decât folosind un mecanism de paginare. În cele ce urmează descriem pe scurt două soluții practice.

2.5.1. SPECTRUM 80K

Ideea metodei este de a folosi integral capacitatea memoriilor RAM, înlocuind circuitele 4532 (32 kB) cu 4164 (64 kB). La unele calculatoare *Spectrum* memoriile 4532 nici nu mai trebuie înlocuite, deoarece funcționează în ambele pagini. Prin urmare avem două pagini de 32 kB, comutabile în același spațiu de adresare: zona 32768÷65535 (#8000÷#FFFF). Pentru aceasta, conexiunea 11 a circuitului IC26, intrarea 3A a multiplexorului (figura 2.6), trebuie poziționată fie în 0, fie în 1. Această poziționare se poate face foarte ușor din software, cu instrucțiunea OUT, într-un *port* suplimentar specializat, la care se înscrie un bistabil de tip "D". Astfel, cele două pagini de 32 kB se pot schimba

cu instrucțiunea OUT. Condiția care trebuie respectată este ca stiva microprocesorului să fie integral în afara spațiului de manevră, mai precis trebuie să se găsească în RAM-ul video.

2.5.2. AMSTRAD SPECTRUM 128 +3

O soluție radicală care mărește memoria disponibilă la 128 (512) kB a fost folosită la *Spectrum* 128 și preluată de Amstrad la *Spectrum* +3. Paginarea se face în blocuri de 16 kB. Memoria ROM are 64 kB și folosește aceeași tehnică de paginare. Paginile ROM nu pot fi comutate în spațiul de adresare al procesorului decât de la adresa 0, însă o pagină RAM poate fi comutată aproape în oricare din cele 4 zone de 16 kB. Paginile RAM nu sunt echivalente datorită păstrării mecanismului de acces al mașinii video la memorie prin blocarea ceasului procesorului. Există două memorii video distincte de 16 kB, din care numai una singură este activă (afișată) la un moment dat. Toate comutările de pagini se realizează cu instrucțiuni OUT la *port*-uri specializate.

2.6. INTERFAȚA 1 DE COMUNICAȚIE ȘI DE DISC

Interfața 1 asigură trei funcții diferite și anume: conectarea discului Microdrive, controlul rețelei locale și *port* de comunicație compatibil RS-232C. Pentru a asigura integrarea acestora în sistemul de operare, Interfața 1 adaugă 8 kB de memorie ROM care include subrutinele necesare. Adresarea acestui ROM suplimentar se face în spațiul de adrese destinat interpretorului Basic.

Mecanismul de comutare a memoriei ROM suplimentare este asigurat de circuitele de decodificare din Interfața 1 și este, pe scurt, următorul: de câte ori este apelat mecanismul de tratare a erorilor (RST #8), un decodificator de adresă detectează acest lucru și comută ROM-ul interfeței în locul ROM-ului standard. Programul existent în ROM-ul interfeței examinează stiva procesorului și verifică dacă eroarea nu se datorează cumva întâlnirii unei instrucțiuni referitoare la noile periferice (disc, rețea, serială). Dacă da, atunci controlul nu este cedat până nu se termină execuția acestei instrucțiuni, altfel înseamnă că este o eroare într-adevăr și ea este semnalizată ca atare de ROM-ul de bază.

Schema-bloc a Interfeței 1 este prezentată în figura 2.13 și schemele electronice în figurile 2.14 și 2.15. Se remarcă faptul că aproape toate funcțiile sunt asigurate de un circuit specializat IC1 care este de tipul GI ULA 1440. În afară de acesta, interfața mai conține încă trei circuite integrate de uz general și câteva tranzistoare. Acestea sunt: IC2 — ROM-ul sistemului, cu o capacitate de 8 kB, realizat cu un circuit EA8364/01; IC3 — decodificatorul de adrese pentru activarea ROM-ului intern; IC4 — inversor folosit pentru generarea semnalului de ceas.

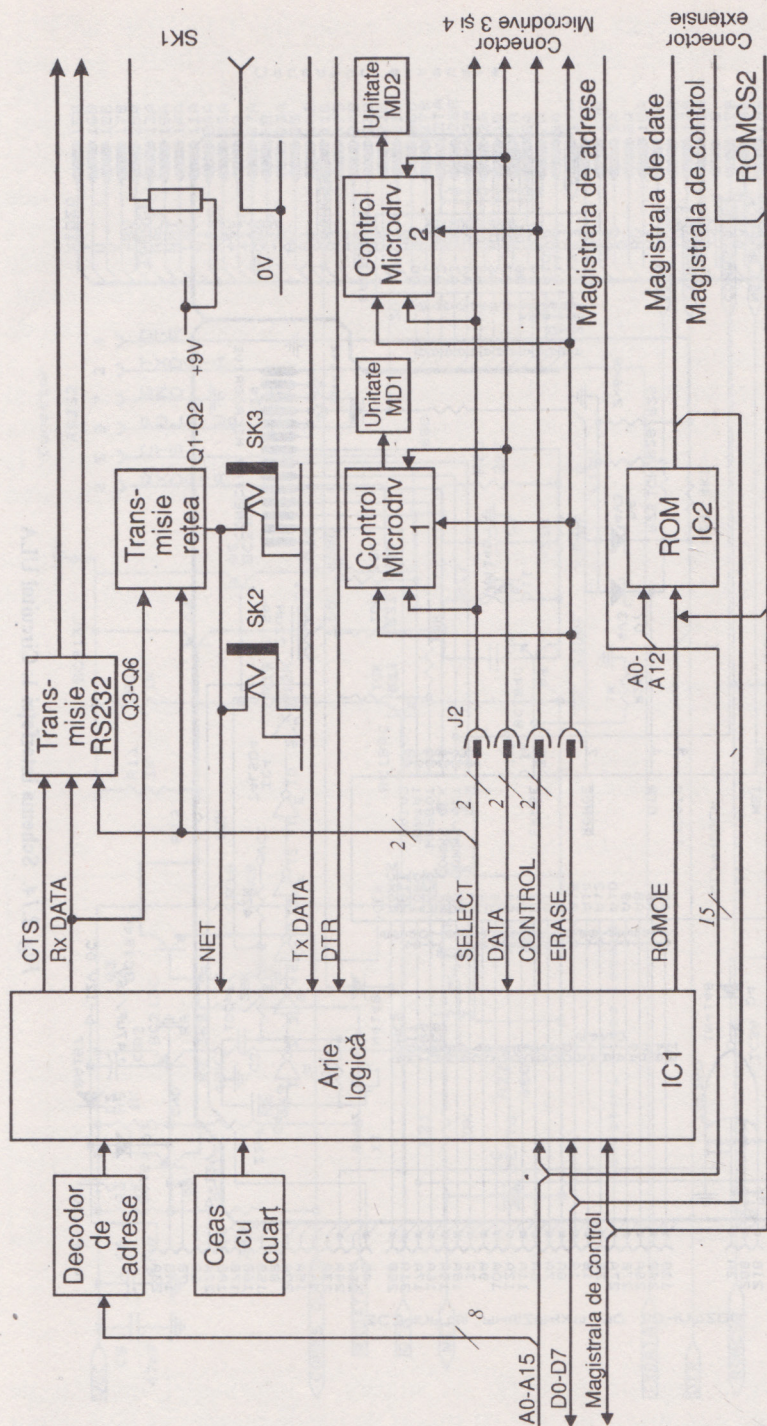


Fig. 2.13. Schema-bloc a Interfetei 1

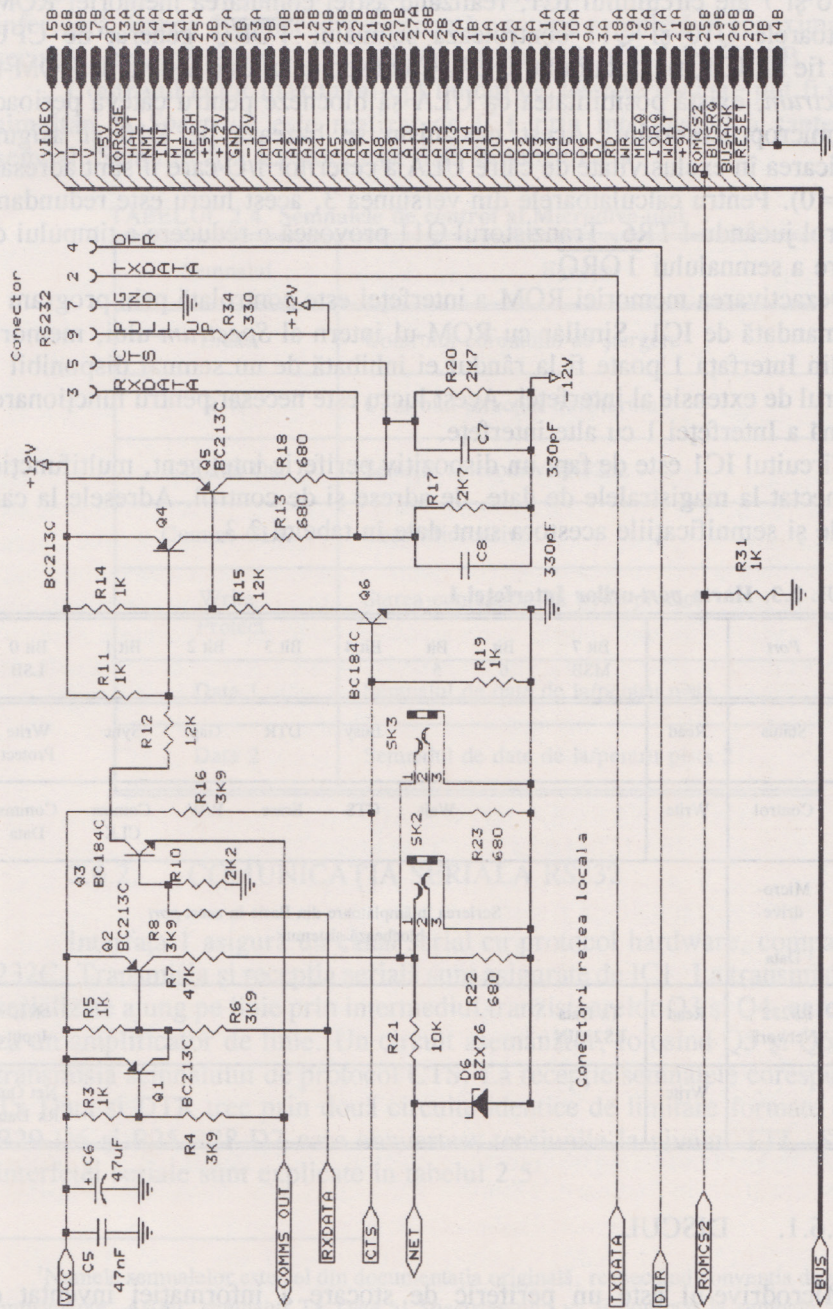


Fig. 2.15. Schema Interfeței 1: Circuitele de comunicație și cupla de extensie

Interfața este activată de decodificarea semnalului DECODE de IC3. Acest semnal comandă un *latch* care activează ROMCS și ROMEN pe conexiunile 6 și 7 ale circuitului IC1, realizând astfel comutarea memoriei ROM. Tranzistoarele Q10 și Q11 controlează semnalul TORQ generat de CPU, destinat fie lui IC1, fie circuitului ULA. Când semnalul este activat de ROM-ul din *Spectrum*, există posibilitatea ca ULA să blocheze pentru câteva perioade ceasul microprocesorului. Acest mecanism se bazează pe Q10 și asigură decodificarea în exclusivitate de către ULA a cererilor I/O care îi sunt adresate (cu A0=0). Pentru calculatoarele din versiunea 3, acest lucru este redundant, același rol jucându-l TR6. Tranzistorul Q11 provoacă o reducere a timpului de comutare a semnalului TORQ.

Dezactivarea memoriei ROM a interfeței este controlată prin program și este comandată de IC1. Similar cu ROM-ul intern al *Spectrum*-ului, memoria ROM din Interfața 1 poate fi la rândul ei inhibată de un semnal disponibil la conectorul de extensie al interfeței. Acest lucru este necesar pentru funcționarea simultană a Interfeței 1 cu alte interfețe.

Circuitul IC1 este de fapt un dispozitiv periferic inteligent, multifuncțional, conectat la magistralele de date, de adrese și de control. Adresele la care răspunde și semnificațiile acestora sunt date în tabelul 2.3.

TABELUL 2.3. Harta *port*-urilor Interfeței 1

Adresa	Port		Bit 7 MSB	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0 LSB
239 (#EF)	Status	Read				Busy	DTR	Gap	Sync	Write Protect
	Control	Write		Wait	CTS	Erase	R/W	Comms CLK	Comms Data	
231 (#E7)	Micro-drive Data		Scrierea întâmplătoare din Basic în acest <i>port</i> blochează sistemul							
247 (#F7)	RS232 Network	Read	Tx Data RS232IN							Net Input
		Write								Net Out Rx Data

2.6.1. DISCUL

Microdrive-ul este un periferic de stocare a informației inventat de Sinclair, cu rol de disc magnetic, dar cu suport de bandă magnetică fără sfârșit

(cartuș). Viteza de derulare este mult mai mare decât în cazul casetofonului, ceea ce asigură o bandă mai largă de frecvențe. Din punctul de vedere al operării, microdrive-ul are toate proprietățile discului, însă la o viteză inferioară. Banda din microdrive execută o rotație completă în aproximativ șapte secunde, ceea ce asigură o capacitate neformatată în jur de 150 kB.

Controlul celor maximum opt unități de *Microdrive* care pot fi conectate simultan la *Spectrum* este realizat de IC1 prin intermediul a șapte linii de semnal. Semnificația acestora este dată în tabelul 2.4.

TABELUL 2.4. Semnalele de control al Microdrive-ului

Semnalul	Funcția
Erase	Controlul curentului de ștergere
R/W	Controlul direcției transferului
Comms Out	Selecție Microdrive/RS232
Comms CLK	Ceas Microdrive
Write Protect	Starea comutatorului write protect
Data 1	Semnalul de date de la/pe/pentru pista 1
Data 2	Semnalul de date de la/pe/pentru pista 2

2.6.2. COMUNICAȚIA SERIALĂ RS232

Interfața 1 asigură un canal serial cu protocol hardware, compatibil RS-232C. Transmisia și recepția serială sunt asigurate de IC1. La transmisie, datele serializate ajung pe linie prin intermediul tranzistoarelor Q3 și Q4, care formează un amplificator de linie. Un circuit asemănător, folosind Q5 și Q6, asigură transmisia semnalului de protocol CTS. La recepție semnalele corespunzătoare Tx Data și DTR trec prin două circuite identice de limitare formate din R24-R29-D6 și R25-R28-D7 care convertesc tensiunile la nivelul TTL. Semnalele interfeței seriale sunt explicate în tabelul 2.5¹.

¹Numele semnalelor este cel din documentația originală, respectând convenția de sens pentru imprimante. Astfel, semnalul Tx Data al *Spectrum*-ului va fi conectat la semnalul omonim al imprimantei.

TABELUL 2.5. Semnalele interfeței seriale RS232

Linia	Funcția
Rx Data <i>Received Data</i>	Date transmise de <i>Spectrum</i> către periferic
Tx Data <i>Transmitted Data</i>	Date recepționate de <i>Spectrum</i> de la periferic
CTS <i>Clear To Send</i>	Avertizează perifericul că <i>Spectrum</i> vrea să transmită date
DTR <i>Data Terminal Ready</i>	Semnal de la periferic indicând faptul că poate primi date

2.6.3. REȚEAUA LOCALĂ

Interfața 1 permite crearea unei rețele locale omogene de date prin interconectarea a maximum 64 de calculatoare *Spectrum*. Fiecare interfață are doi conectori de tip *jack*, notați SK2 și SK3, care sunt legați în paralel la conexiunea NET a IC1. Rețeaua este cu emitor comun și dacă ambii conectori sunt folosiți, linia nu se încarcă. Dacă un conector rămâne nefolosit (cazul primului și ultimului calculator) atunci acesta pune linia la masă printr-o rezistență de 680 Ω ceea ce asigură menținerea unei impedanțe constante de 340 Ω , indiferent de numărul stațiilor conectate. Dioda D6 (figura 2.15) protejează intrarea NET de supratensiuni periculoase.

2.7. PREZENTAREA GENERALĂ A COMPATIBILELOR ROMÂNEȘTI

Principalele compatibile realizate în țară sunt¹:

– Mărăcineanu 1. Printre primele calculatoare compatibile *Spectrum*. Proiectat la ITC București de Eugen Mărăcineanu. Subsistemul video respectă mecanismul de acces la memorie cu blocarea ceasului microprocesor. Cei 48 kB de memorie sunt realizați din trei bancuri de 16 kB, iar memoria ROM din opt circuite EPROM 2716. Nu a fost realizat în producție de serie însă este relativ răspândit în special printre constructorii amatori din București. Schema sa este prezentată în [29].

– TIM-S. Proiectat de un colectiv al Institutului Politehnic "Traian Vuia" din Timișoara (Crișan Strugaru, Dumitru Pănescu, Cezar Morun) și construit

¹ Prezentarea este departe de a fi completă. Acolo unde calculatorul nu are un nume comercial, am folosit numele proiectantului. Ordinea este cronologică.

la Fabrica de Memorii Timișoara. Are 80 kB de memorie RAM din care numai 64 sunt accesibili microprocesorului, și 16 kB ROM. Memoria video este dublată, ceea ce permite accesul la citire al procesorului fără blocarea ceasului. La scriere însă acesta trebuie în continuare blocat. Este primul calculator din familia *Spectrum* construit cu microprocesorul Z80 B, cu posibilitate de accelerare (turbo). Chiar la frecvența normală de 3,5 MHz, viteza de calcul este cu aproximativ 5% mai mare decât cea a *Spectrum*-ului. În schimb, la frecvența de 6 MHz, creșterea de viteză este într-adevăr semnificativă (cu aproximativ 80%). Permite comutarea din mers a ceasului microprocesorului de la 3,5 MHz la 6 MHz și invers. Memoria ROM nu este folosită decât la pornire, apoi este copiată în RAM. Tehnica se numește *shadow* și este utilizată pe scară largă la calculatoarele PC. Este posibilă încărcarea de pe casetă a ROM-ului original de *Spectrum*. Inițial a fost dotat cu o tastatură plată, fiind apoi înlocuită cu o tastatură mecanică. Nu a fost echipat cu un codor PAL. A fost realizat și un echivalent al Interfeței 1, Timext, proiectat de Cezar Morun, care permite conectarea a două unități de disc de 5"¼, fiecare emulând 4 unități de Microdrive. A fost produs în serie, din 1985 până în 1990. Schemele sale sunt prezentate în [51].

– HC85. Proiectat la FCE București, în colaborare cu Institutul Politehnic București. Colectivul a fost condus de A. Petrescu. Este o "copie" fidelă a *Spectrum*-ului, cu excepția bitului de strălucire, care la acest calculator nu are nici un efect. Ca urmare, imaginea sa color PAL are doar 8 culori. Placa de cablaj imprimat este în patru straturi, cu un număr mic de circuite integrate, dispuse compact. A fost realizat și un echivalent al Interfeței 1 care însă nu este total compatibil cu aceasta și care permite conectarea unei unități de disc, o interfață de rețea și o interfață serială. A fost produs în serie între anii 1985-1988. Schemele sale sunt prezentate în [50].

– Cobra 1. Proiectat la ITC Brașov de Wagner Bernd Hansgeorg, ajutat de câțiva colaboratori. Este un calculator cu dublă funcționalitate: compatibil *Spectrum* și compatibil CP/M. Are $2 \div 18$ kB de ROM și $64 \div 80$ kB RAM accesibili procesorului prin paginare. Interfața de disc se află pe o placă de extensie ce se montează într-un conector special. Există mai multe versiuni de sistem de operare compatibile *Spectrum*, unele permițând accesul la disc din Basic. Dacă interfața de disc este prezentă, calculatorul nu necesită decât 2 kB de RAM. Are o tastatură specială cu 61 de taste unele dublând combinații de taste. Calculatorul a fost produs artizanal, pe scară largă. Este compatibilul cu cea mai mare priză la constructorii amatori.

– Cobra 2. O versiune minimală de calculator compatibil *Spectrum*. Subsistemul video a fost realizat cu circuite ROM bipolare, având un singur banc de 64 kB RAM și un circuit ROM 27128. Placa de circuit imprimat se apropie ca dimensiuni de cea a *Spectrum*-ului original. Nu s-au construit decât câteva exemplare.

– HC88. Proiectat la FCE București. Compatibil dual – *Spectrum* și CP/M. Ecranul are două moduri grafice (256×192 și 512×192). Include interfața de disc și mecanismele necesare pentru Interfața 1. Are 128 kB RAM și doar 2 kB ROM, sistemul de operare încărcându-se de pe dischetă. Afișarea

nu se poate face decât pe propriul monitor. A fost produs în serie scurtă între anii 1988–1990.

– CIP. Realizat la Întreprinderea "Electronica" București. Proiectarea sa se remarcă prin economicitate, sistemul de operare este rezident pe casetă și nu în ROM. Un ROM de 2 kB are rolul de a încărca de pe casetă sistemul. Tastatura mecanică este printre cele mai reușite la compatibilele românești.

– JET. Realizat la Întreprinderea "Electromagnetica" din București. Intenția proiectanților a fost de a realiza o mașină de jocuri TV, cu interfață de joystick încorporată. Tastatura este nerecomandabilă pentru introducerea de texte. Este destul de compatibil, datorită cerinței de a rula toate jocurile.

– TIM-S PLUS. Proiectat de Dumitru Pănescu, a fost realizat la Fabrica de Memorii din Timișoara. Este compatibil *Spectrum*, Microdrive și CP/M. Are două unități de disc floppy de 5"¼. S-au realizat puține exemplare.

– HC90. O versiune reproiectată a lui HC85. Principala noutate constă în modul de funcționare a mașinii video și utilizarea memoriei EPROM de 16 kB. Accesul procesorului la memoria video are prioritate maximă și se realizează fără blocarea ceasului sau stări de WAIT. Acest lucru este posibil printr-o complexitate ceva mai mare a *controller*-ului video, dar permite în schimb utilizarea unui singur banc de memorie DRAM de 64 kB. În rest, calculatorul menține toate trăsăturile lui HC85: stabilizator încorporat, compatibilitate cu periferia *Spectrum* etc. A fost produs în serie, între anii 1990–1991.

– HC91. Este ultima versiune¹ de compatibil *Spectrum* aflată în producție de serie la FCE și constă într-o reproiectare a lui HC90. De asemenea, se produce un echivalent funcțional al Interfeței 1, echipat cu disc de 5"¼ sau 3"½.

¹Este vorba de momentul în care a fost scrisă cartea, ianuarie 1993.

PROGRAMAREA ÎN LIMBAJ DE ASAMBLARE PE COMPATIBILELE SPECTRUM

3.1. INSTRUCȚIUNILE MICROPROCESORULUI Z80

Instrucțiunile microprocesorului Z80 sunt descrise în diferite lucrări ([27], [29], [37], [2]), a căror consultare va trebui să însoțească lectura acestui capitol. Considerăm totuși ca fiind utilă prezentarea unei sinteze a instrucțiunilor în două moduri suplimentare, care prezintă avantaje în practică.

3.1.1. TABELELE MATRICEALE ALE INSTRUCȚIUNILOR

Tabelele matriceale ale instrucțiunilor, după codurile de instrucțiune, sunt deosebit de utile la dezasamblarea manuală, rapidă, a unor segmente scurte de cod. De asemenea, clarifică în mare măsură organizarea codurilor de instrucțiuni.

În anexa A se dă tabelul principal al instrucțiunilor, cu 16 linii și 16 coloane, numerotate de la 0 la F. Linia corespunde cifrei hexazecimale de rang superior, iar coloana corespunde cifrei de rang inferior. Astfel, la intersecția dintre linia 3 și coloana 7 se găsește instrucțiunea SCF (codul operației #37). Deoarece microprocesorul Z80 are mai mult de 256 de instrucțiuni, patru din codurile de operație, cele pe fond negru, sunt de fapt prefixe pentru extinderea numărului de coduri.

Două din prefixe (#DD și #FD) redirecționează operațiile în care este implicat registrul HL, către registrul IX, respectiv către registrul IY. Totodată, operațiile care utilizează modul indirect via registrul HL se transformă în operații cu modul de adresare indexat, via registrul IX, respectiv IY, plus un deplasament imediat de 1 octet (d). De exemplu, LD HL,HL devine LD IX,IX, iar SUB (HL) devine SUB (IX+d).

Celelalte două prefixe (#CB și #ED) trimit către anexele B și C, construite pe aceeași structură. Codurile de operație care apar pe fond închis nu trebuie folosite, deoarece au fost excluse din documentația oficială Zilog din cauza unor imperfecțiuni. Acestea, precum și altele care nu au mai fost reprezentate, constituie așa numitele *instrucțiuni ascunse* [37].

3.1.2. TABELUL INSTRUCȚIUNILOR ÎN ORDINEA VITEZEI DE EXECUȚIE

Tabelul instrucțiunilor în ordinea vitezei de execuție se găsește în anexa D și grupează instrucțiunile după criteriul duratei lor în perioade de ceas de unitate centrală (T). Cele mai rapide instrucțiuni apar pe prima linie și durează 4 T. Pentru a converti durata unei instrucțiuni din perioade de ceas (T) în microsecunde, se va folosi următoarea formulă:

$$t = \frac{n T}{f_{CPU} \text{ MHz}} \mu s$$

Frecvența ceasului unității centrale (CPU) la calculatoarele *Spectrum* este în general de 3,5 MHz. Excepție fac calculatoarele TIM-S, care pot funcționa și la 6 MHz. Reiese că durata minimă a unei instrucțiuni mașină complete, de exemplu NOP, va fi de 1,143 μs , iar durata maximă a unei instrucțiuni, de exemplu EX (SP).IX, va fi de 6.571 μs , adică de 6 ori mai mult. Acest tabel este util în primul rând pentru că oferă programatorului un criteriu foarte important în alegerea instrucțiunilor, criteriul vitezei. Atunci când există mai multe variante de rezolvare a aceleiași funcții, vom prefera varianta cu instrucțiuni mai rapide. Pe de altă parte, la programarea în timp real trebuie să echilibrăm durata execuției unui program pe diversele ramuri, prin adăugarea unor instrucțiuni de întârziere, fără alt efect. Din tabelul prezentat se poate alege direct instrucțiunea sau combinația de instrucțiuni convenabilă.

3.2. PARTICULARITĂȚI DE PROGRAMARE A MICROPROCESORULUI Z80 LA COMPATIBILELE *SPECTRUM*

Față de alte calculatoare cu Z80, compatibilele *Spectrum* impun programatorilor anumite restricții, dar oferă și o importantă colecție de subrutine utile și accesibile, în ROM (§ 3.3.). În cele ce urmează vom analiza restricțiile mai importante.

3.2.1. LOCALIZAREA PROGRAMELOR

Programele în cod mașină trebuie amplasate conform următoarelor reguli:

— dacă se intenționează revenirea controlului în Basic, atunci trebuie evitate zona variabilelor de sistem, zona de program Basic și variabile Basic, până la RAMTOP. Există și o excepție, a codului mașină ascuns într-o linie de comentariu Basic (REM), sau într-o variabilă, dar aceste soluții presupun precauții speciale în ceea ce privește localizarea și calculul adresei punctului de intrare.

Modul cel mai recomandabil de a respecta această regulă este de a da o comandă CLEAR nn, și de a amplasa codul mașină al utilizatorului de la adresa nn+1 (folosind directiva ORG nn+1). Atunci când se folosesc caracterele grafice definite de utilizator (UDG-urile), codul mașină nu trebuie să se suprapună peste zona rezervată acestora (de la variabila de sistem UDG);

– dacă execuția programului trebuie să se facă în timp real, atunci codul mașină va fi amplasat în afara zonei de memorie video (#4000-#7FFF), deoarece la majoritatea compatibililor *Spectrum*, accesul unității centrale în această zonă este întârziat prin încetinirea ceasului.

3.2.2. REGISTRE REZERVATE

Rutina standard de tratare a întreruperilor mascabile (de la #0038) utilizează registrul IY ca *pointer* în zona variabilelor de sistem (valoarea sa este #5C3A). Din acest motiv, registrul nu trebuie folosit în alte scopuri decât pe perioada dezactivării întreruperilor (cu DI) și cu condiția refacerii conținutului său înainte de reactivarea întreruperilor (cu EI).

De asemenea, se va evita alterarea registrului I, care în mod normal conține (#3F). În modul 1 de întreruperi, propriu *Spectrum*-ului, aparent nu are importanță ce valoare conține registrul I. Totuși, datorită logicii particulare a mașinii video (ULA), biții 16 și 17 ai registrului I au o semnificație aparte la *Spectrum*, (§ 3). Efectul poziționării lor pe 1 va fi provocarea unor dereglări ale imaginii (*snow*). În schimb, alterarea sistematică a registrului R nu poate provoca dereglări ale înprospătării, deoarece la *Spectrum* aceasta este asigurată de ULA și nu de CPU, cum ar fi normal.

În sfârșit, registrul H'L' la întoarcerea în Basic trebuie refăcut cu valoarea #2758, dacă cumva a fost utilizat.

3.2.3. REGIMUL ÎNTRERUPERILOR

După cum am arătat, modul de întreruperi cu care se inițializează calculatoarele compatibile *Spectrum* este IM 1. Cu precauțiile arătate la §3.3.9., programatorii pot beneficia de redirectarea rutinei de tratare a întreruperilor prin modul 2. Pe perioada dezactivării întreruperilor, ceasul de timp real al calculatorului (variabila de sistem FRAMES) nu va fi actualizată și prin urmare acest ceas va rămâne în urmă (de exemplu, în timpul încărcării sau salvării pe casetă, generării sunetelor etc.).

Înainte de revenirea în Basic este obligatorie reactivarea întreruperilor (cu EI), altfel interpretorul nu poate citi nici o tastă, deoarece acest serviciu este asigurat chiar de rutina RST #38. În schimb, interpretorul poate rula un program Basic chiar cu întreruperile dezactivate, dacă acesta nu solicită citirea tastaturii.

Instrucțiunile de întoarcere din rutinele de tratare a întreruperilor (RETI și RETN) nu au nici un efect la *Spectrum*, deoarece acesta nu conține alte circuite din familia Z80 (PIO, SIO).

Întreruperea nemascabilă (NMI) nu poate fi exploatată la *Spectrum*, deoarece la adresa #0066 este o secvență care provoacă inițializarea (salt la #0000). Inițial, intenția autorilor sistemului de operare *Spectrum* (colectivul lui Steven Vickers) a fost de a permite redirectarea tratării întreruperilor nemascabile via variabila de sistem NMIADD (#5CB0). Dar după experiența pirătăriei programelor la Sinclair ZX-81 (predecesorul *Spectrum*-ului), s-a renunțat la această facilități, care ar fi servit copierii neautorizate a programelor, indiferent de metodele de protecție la încărcare. Unii autori [24] consideră în mod eronat această situație una din greșelile sistemului de operare, de fapt este o modificare intenționată.

3.3. PRINCIPALELE SUBROUTINE DIN ROM

Programatorii în limbaj de asamblare care doresc să folosească în propriile programe rutine ale interpretorului pot apela aceste rutine cu următoarele precauții:

- variabilele-sistem trebuie să rămână intacte (sau rescrise cu conținutul inițial);
- la fel și registrul IX; el conține în mod uzual valoarea #5C3A¹, care este baza adresării indexate a variabilelor de sistem;
- registrul H'L conține adresa buclei principale a interpretorului și poate fi modificat doar dacă nu se intenționează întoarcerea în Basic;
- în general rutinele sunt insensibile la starea întreruperilor;
- anumite rutine (de casetă, de sunet) afectează starea întreruperilor;
- rutinele de citire a tastaturii funcționează numai cu întreruperile activate.

3.3.1. SUBROUTINELE DE CASETĂ

Informația se stochează pe casetă (sau bandă magnetică) sub formă de fișiere cu următoarea structură la nivelul logic de organizare:

- un preambul de identificare de lungime fixă (#11 = 17), denumit *header*;
- blocul de date (informația efectivă) de lungime variabilă, denumit *block*.

Structura *header*-ului este următoarea:

¹ La un calculator fără Interfața 1.

0	1	11	13	15
ID	NAME	LEN	ADR	PROG

Câmpurile au următoarele semnificații:

ID — octet de identificare a tipului de fișier: ID=0 pentru program Basic (*Program*); ID=1 pentru tablou numeric (*Number Array*); ID=2 pentru tablou de caractere (*Character Array*); ID=3 pentru cod mașină sau imagine memorie (*Bytes*).

NAME — numele fișierului, de maximum 10 caractere, completat cu blankuri.

LEN — lungimea blocului de date (2 octeți în format Intel).

ADR — în funcție de octetul de identificare, acest parametru conține: (pentru ID=0) numărul de linie Basic unde se va face automat GO TO după terminarea încărcării (linia de AUTO-RUN); (pentru ID=1 sau ID=2, în octetul 14) o structură de biți care furnizează numele implicit al variabilei în format intern Basic; (pentru ID=3) adresa implicită de unde va începe încărcarea blocului în memorie.

PROG — este utilizat numai în cazul ID=0 și conține lungimea programului Basic fără variabile.

La nivelul fizic de organizare, atât *header*-ul cât și *block*-ul au aceeași structură, compusă din următoarele elemente:

- *leader* (ton de start) cu frecvența de 800 Hz și lungime de 2 secunde pentru *block* și 5 secunde pentru *header*; durata unui impuls de *leader* este 254% din durata unui "0";

- *sync pulse* (impuls de sincronizare) cu durata 82% din durata unui "0";

- *flag byte* (octet de identificare) care conține 0 pentru *header* și #FF pentru *block*;

- *data bytes* (octeți de date utile);

- *checksum byte* (octet de sumă de control) care conține rezultatul operațiilor succesive SAU EXCLUSIV asupra tuturor octeților de date și octetului de identificare;

- *delay* (pauză finală) care durează 400% din durata unui "0".

Codificarea informației pe casetă se face prin modulație în frecvență. Fiecare octet este serializat în biții componenți, începând cu cel mai semnificativ bit. Între octeți nu se lasă nici o pauză. Dacă bitul are valoarea 0, se generează un impuls cu durata de 0,48857 ms ($2 \cdot 855T$). Dacă bitul are valoarea 1, se generează un impuls cu durata dublă față de cea a unui "0" (0,97714 ms sau $2 \cdot 1710T$). Polaritatea nu este importantă, impulsurile fiind simetrice. La încărcare, semnalul de la casetofon nu trebuie să aibă fronturi abrupte, semnalul nedistorsionat fiind sinusoidal. De asemenea, este important ca în domeniul de frecvență al semnalului (500÷2500 Hz) să nu apară schimbări importante în faza semnalului (caracteristica fază-frecvență să fie cât mai plată). Acest fapt recomandă folosirea unui casetofon cât mai simplu, fără corecții complexe de ton sau filtre pentru reducerea zgomotului.

3.3.1.1. Încărcarea de pe casetă (LD_BYTES)

Subrutina de încărcare a unui bloc de octeți de pe casetă are punctul normal de intrare la adresa #0556 (1366) și este folosită atât pentru încărcarea *header*-ului cât și pentru *block*. Pentru aplicații speciale se utilizează un alt punct de intrare la adresa #0562 (1378). Înainte de apelarea acestei subrutine, registrele IX, DE și AF trebuie preîncărcate cu următorii parametri de intrare:

IX — adresa unde va începe încărcarea (între 0 și #FFFF)¹;

DE — numărul de octeți (între 0 și #FEFF); dacă numărul de octeți care trebuie încărcăți nu este cunoscut apriori, atunci registrul D va fi încărcat cu #FF, iar registrul E nu mai contează;

A — (*Flag Byte*): A = 0 pentru încărcarea unui *header*; A = #FF pentru încărcarea unui *block*;

CY — comutator de mod de lucru: CY = 1 pentru operația de încărcare propriu-zisă (LOAD); CY = 0 pentru operația de verificare, adică de comparare a conținutului memoriei cu înregistrarea de pe casetă (VERIFY).

Ieșirea din subrutină se poate face în două moduri: normal (cu sau fără eroare de încărcare) sau prin excepție, la apăsarea tastei BREAK. În cel de-al doilea caz, controlul nu mai revine programului apelant, acesta fiind cedat interpretorului Basic prin rutina de eroare (RST 8), v. § 3.3.8. La ieșirea normală, registrele vor conține următorii parametri:

CY — indicator de eroare: CY = 1 pentru încărcare (verificare) fără eroare; CY = 0 pentru eroare la încărcare (verificare);

IX — adresa următoare (după ultimul octet încărcat);

DE — numărul de octeți care au mai rămas de încărcat (0 în cazul ieșirii normale fără eroare);

H — rezultatul operației de verificare a integrității octeților încărcăți prin suma de control: H = 0 dacă suma de control este corectă; H ≠ 0 dacă suma de control este incorectă.

În mod uzual, informația din registrul H este inutilă deoarece o eroare în suma de control va fi semnalată prin CY = 0.

3.3.1.2. Salvarea pe casetă (SA_BYTES)

Subrutina de salvare a unui bloc de octeți pe casetă are punctul de intrare la adresa #04C2 (1218) și este folosită atât pentru salvarea *header*-ului cât și pentru *block*. Programele de copiere folosesc un alt punct de intrare, la #04C6. Înainte de apelarea subrutinei registrele IX, DE și A trebuie încărcate cu următorii parametri de intrare:

IX — adresa de unde va începe salvarea (între 0 și #FFFF);

¹ Bineînțeles că încărcarea se face efectiv numai în domeniul de adrese ocupat de RAM, adică de la #4000 la #FFFF inclusiv.

DE — numărul de octeți ce urmează a fi salvat (între 0 și #FFFF);

A — *Flag Byte*: A = 0 pentru salvarea unui *header*; A = #FF pentru salvarea unui *block*.

În punctul de vedere al subrutinei, singura deosebire dintre *header* și *block* este lungimea tonului de start (*leader*), respectiv 5 s față de 2 s. *Header*-ul este recunoscut la încărcare numai dacă are 17 octeți lungime și respectă structura impusă.

Ieșirea din subrutină se poate face în două moduri: normal sau prin excepție, la apăsarea tastei BREAK. În cel de-al doilea caz, controlul nu mai revine programului apelant, fiind cedat interpretorului Basic, prin rutina de eroare (RST 8) v. §3.3.8. La ieșirea normală, conținutul registrelor nu are importanță.

3.3.1.3. Exemple de apelare

Încărcarea unui bloc fără *header* în memoria video:

```
ADRSTART EQU #4000      : adresa de început a memoriei video
LUNGIME   EQU #1B00      : lungimea blocului de date egală cu dimensiunea
                          : memoriei video
LD_BYTES  EQU #0556      : adresa rutinei de încărcare
ENT $     : punctul de intrare
LD IX,ADRSTART : transferul parametrilor
LD DE,LUNGIME
LD A,#FF      : block
SCF           : încărcarea propriu-zisă
CALL LD_BYTES : apelul rutinei de încărcare
JR NC,EROARE  : în caz de eroare, CY=0
RET           : rutina de tratare a erorii de încărcare
EROARE     RST #08      : rutină de eroare a Basic-ului
          DEFB #1B      : mesaj 'Tape loading error'
```

Salvarea unui fișier de cod (*header* + *block*) ce conține ecranul:

```
SA_BYTES EQU #04C2      : adresa rutinei de salvare
HEADER:   : urmează cei 17 octeți ai header-ului
ID        DEFB #03      : CODE
NAME      DEFB 'ECRAN'  : numele fișierului (10 octeți)
LEN       DEFW #1B00     : dimensiune memorie video
ADR       DEFW #4000     : adresă de început memorie video
PROG      DEFW #0000     : nefolosit pentru CODE
          ENT $          : punctul de intrare
          LD IX,HEADER   : salvare header (17 octeți de la adresa
                          : HEADER)
          LD DE,#11
          LD A,#00
          CALL SA_BYTES
          EI
          LD B,25        : 0.5 s pauză între header și block
PAUZA:    HALT
          DJNZ PAUZA
```



```

LD IX,(ADR)      : salvare bloc de la adresa și cu lungimea
                  : specificate în HEADER
LD HL,(LEN)
EX DE,HL
LD A,#FF
CALL SA_BYTES
RET

```

3.3.2. GENERATORUL DE SUNET

3.3.2.1. Apelarea subrutinei

Calculatorul *Spectrum* are un generator de sunet simplu, realizat prin conectarea unuia din biții *port*-ului de ieșire la un difuzor, mai exact bitul D4 din *port*-ul #FE. Controlul difuzorului se realizează prin software, prin poziționarea succesivă pe 1 și pe 0 a acestui bit. Intervalul dintre două poziționări determină frecvența sunetului emis. De exemplu, pentru a genera un ton cu frecvența de 1000 Hz este necesar să activăm bitul D4 din *port*-ul #FE timp de 0,5 ms și să-l dezactivăm pentru următoarele 0,5 ms ($0,5 + 0,5 = 1 \text{ ms} = 1/1000 \text{ Hz}$). Ceasul microprocesorului fiind de 3,5 MHz, recurgem la o buclă de întârziere de $0,5 \text{ ms} \times 3,5 \text{ MHz} = 1750 \text{ T}$. Întreruperile vor fi dezactivate pentru a obține un ton pur, altfel la fiecare 20 ms ar interveni tratarea întreruperii mascabile de citire a tastaturii.

Rutina din ROM poate fi apelată în două moduri. Prima posibilitate este prin punctul de intrare #03B5 (BEEPER). Înainte de apelarea acestei subrutine, registrele DE și HL trebuie încărcate cu următorii parametri de intrare:

DE — numărul de impulsuri care formează sunetul; se poate estima cu formula $\text{INT}(f \cdot t)$, unde f este frecvența sunetului, iar t , durata sa;

HL — numărul de stări T pentru bucla de așteptare; se poate estima cu formula $\text{INT}(437500/f - 30.125)$.

Starea registrelor la ieșirea din subrutină este nedefinită. De remarcat că indiferent de starea lui IFF1 la intrare, după apelul acestei subrutine întreruperile vor fi activate.

O a doua posibilitate de apel este punctul de intrare #03F8 (BEEP, rutina de comandă a interpretorului Basic). Parametrii în acest caz sunt însă ceva mai greu de transferat deoarece trebuie folosit calculatorul aritmetic. Argumentele (durata și nota) trebuie depuse în stiva calculatorului aritmetic, în ordinea: durată, notă. Formatul este cel al instrucțiunii Basic. În cazul în care produsul durată-frecvență depășește 65535, controlul este cedat rutinei de eroare (RST #08), cu mesajul "Integer out of range", și sunetul nu este emis. Starea registrelor la intrarea în subrutină nu contează, iar la ieșire este nedefinită.

Efectul produs de cele două puncte de intrare este diferit datorită faptului că în al doilea caz, trecerea de la o notă la alta este sesizabilă (necesită un timp apreciabil mai mare, din cauza calculului în virgulă mobilă).

3.3.2.2. Exemple de utilizare

Generarea notei do pentru 0,5 s (echivalent cu BEEP 0.1.0) folosind punctul de intrare #03B5:

```
NRPULS EQU 131 : INT(261.63*0.5) frecvență*durată
DELAY EQU 1642 : INT(437500/261.63-30.125).nr. T
BEEPER EQU #03B5 : adresa punctului de intrare
ENT $
LD HL,DELAY
LD DE,NRPULS
CALL BEEPER
RET
```

Generarea aceluiași sunet folosind punctul de intrare #03F8:

```
BEEP EQU #03F8 : adresa punctului de intrare
STACK_NUM EQU #33B4 : subrutina de copiere a unui număr în
: virgulă mobilă din memorie în vârful
: stivei calculatorului aritmetic.
DURATA DEFB #7F,#00,#00,#00,#00 : reprezentarea lui 0.5 (5 octeți)
NOTA DEFB #00,#00,#00,#00,#00 : reprezentarea lui 0 (5 octeți)
ENT $
LD HL,DURATA : pregătește adresa duratei
CALL STACK_NUM : numărul 0.5 este pus pe stiva calcula-
: torului aritmetic
LD HL,NOTA : pregătește adresa notei
CALL STACK_NUM : numărul 0 este pus pe stiva calculato-
: rului aritmetic
CALL BEEP : apelul subrutinei
: la ieșire, stiva calculatorului arit-
: metic va fi descărcată
RET
```

3.3.3. SCRIEREA UNUI CARACTER

3.3.3.1. Apelarea subrutinei

Modul normal de afișare a unui caracter pe ecran sau la imprimantă exploatează subrutina de la adresa #0010 printr-o instrucțiune de apel în pagina zero modificată (RST #10). Pentru a scrie pe ecran, trebuie deschis în prealabil canalul de ecran (CHAN_OPEN) cu parametrul 2 pentru partea superioară și 0 sau 1 pentru cea inferioară. Pentru trimiterea unui caracter la imprimantă se folosește canalul 3. Dacă este prezentă Interfața 1, atunci se pot folosi și alte canale inițializate în prealabil, pentru scrierea în fișiere pe *microdrive* sau pe disc.

La intrare, subrutina de scriere a unui caracter are nevoie de un singur parametru:

A — codul caracterului; dacă registrul A conține un cuvânt-cheie (> 164), atunci el este expandat (descompus în literele care îl formează); dacă registrul

A conține un cod de control, atunci se execută acțiunea corespunzătoare codului respectiv; parametrii codurilor de control sunt verificați și dacă ies din domeniul permis, controlul este cedat rutinei de eroare (RST #08).

Caracterul se trimite în poziția și cu atributele canalului curent. Dacă este cazul, pe canalul 2 apare mesajul "scroll?". Tentativa de a scrie mai mult de 22 de linii în partea de editare — pe canalele 0 sau 1 — duce la eroare, cu ieșire prin RST #08.

Forma caracterelor pe ecran este determinată de generatorul de caractere folosit. Adresa generatorului de caractere este variabila de sistem CHARS, care poate fi modificată, v. anexa F.

Rutina RST #10 afectează acumulatorul și registrele alternative. Dacă valoarea acumulatorului este codul unui cuvânt-cheie, atunci sunt afectate toate registrele.

3.3.3.2. Exemple de utilizare

Scrierea unui caracter ('*') pe ecran în poziția și cu atributele curente:

```
CHAN_OPEN    EQU #1601      : adresa subrutinei comutare canal
ENT $
LD A,#02     : ecran
CALL CHAN_OPEN : canalul curent este #02
LD A,'*'     : codul caracterului în A
RST #10      : afișează '*'
RET
```

Scrierea unui caracter ('*') pe ecran în linia 5 coloana 10 cu atributul de clipire:

```
CHAN_OPEN    EQU #1601      : adresa subrutinei de comutare canal
AT           EQU #16        : codul de control pentru AT
LIN          EQU #5         : poziția
COL          EQU #10        :
FLASH       EQU #12        : codul de control pentru FLASH
ON           EQU #01        :
OFF          EQU #00        :
ENT $
LD A,#02     : canalul curent este #02 - ecranul
CALL CHAN_OPEN
LD A,AT      : AT LIN,COL
RST #10
LD A,LIN
RST #10
LD A,COL
RST #10
LD A,FLASH   : FLASH 1
RST #10
LD A,ON
RST #10
LD A,'*'     : afișare caracter
RST #10
LD A,FLASH   : FLASH 0
RST #10
```



```
LD A,OFF
RST #10
RET
```

Pentru lista caracterelor de control și semnificația lor vezi anexa E.

3.3.4. SCRIEREA UNUI MESAJ

3.3.4.1. Apelarea subrutinei

Afișarea unui mesaj în poziția curentă se poate face folosind subrutina RST #10 pentru a scrie literă cu literă. Cum această funcție este des folosită, există, în cazul în care se respectă o anumită convenție, posibilitatea de a folosi rutina standard a interpretorului Basic. Aceasta are punctul de intrare #0C0A (PO_MSG) și necesită următorii parametri:

DE — adresa de început a tabeli de mesaje;

A — numărul mesajului în tabelă.

Subrutina folosește tot RST #10 pentru a scrie fiecare caracter. Toate registrele sunt afectate. Nu este posibilă scrierea unui caracter al cărui cod este mai mare decât #7F. Tabela de mesaje trebuie să înceapă cu #80. Sfârșitul de mesaj este semnalizat prin punerea bitului 7 al ultimului caracter pe 1.

3.3.4.2. Exemple de utilizare

Scrierea unui mesaj:

```
CHAN OPEN EQU #1601 : adresa subrutinei de comutare canal
PO MSG EQU #0C0A : adresa subrutinei de afișare mesaj
MESAJ: DEFB #80 : început tabelă
DEFM 'Un exemplu de folosire!!';mesaj
DEFB #80+#0D : ultimul caracter are bitul 7 setat (în
: cazul acesta ENTER)
: eventual alte mesaje
ENT $
LD A,#02 : canalul curent este #02
CALL CHAN OPEN
LD DE,MESAJ : adresa tabela în DE
LD A,0 : număr mesaj (0) în A
CALL PO_MSG : afișează mesajul
RET
```

Afișarea unui titlu subliniat în linia 1, coloana 12:

```
CHAN OPEN EQU #1601 : adresa subrutina de comutare canal
PO MSG EQU #0C0A : adresa subrutinei de afișare mesaj
AT EQU #16 : cod de control AT
LIN EQU #01 : poziția
COL EQU #12
INK EQU #10 : cod de control INK
RED EQU 2 : roșu
BLUE EQU 1 : albastru
```


TITLU:	DEFB #80	: titlu care conține și caractere de control. Poate fi oricât de lung.
	DEFB AT,LIN,COL	
	DEFB INK,RED	
	DEFM 'TITLU ECRAN'	
	DEFB INK,BLUE	
	DEFB AT,LIN+1,COL	: o linie mai jos
	DEFM '-----'	
	DEFB #80+#0D	: sfârșit mesaj
	ENT \$	
	LD A,#02	: canalul curent #02 - ecranul
	CALL CHAN OPEN	
	LD DE,TITLU	: adresa tabela
	LD A,0	: primul mesaj (0)
	CALL PO_MSG	
	RET	

3.3.5. SUBRUTINELE GRAFICE

Pentru majoritatea aplicațiilor în cod-mașină, rutinele grafice ale interpretorului Basic sunt fie prea lente, fie impun restricții. Ele oferă însă o modalitate comodă de lucru atunci când viteza nu este criteriul principal.

3.3.5.1. Ștergerea ecranului

În funcție de rezultatul dorit, există mai multe puncte de intrare:

Punctul #0D6B (CLS). Este chiar subrutina de comandă a interpretorului. Nu este necesar nici un parametru. Apelarea acestui punct de intrare este identică cu efectuarea CLS din Basic. Ecranul este șters cu atributele permanente curente. Ultimele două linii (cele de editare) vor avea atributele corespunzătoare *border*-ului. Este deschis canalul #FE ('S') cu adresa PRINT_OUT forțată la #09F4 și apoi #FD ('K') cu adresa PRINT_OUT la #09F4 și cea de KEY_INPUT la #10A8, indiferent de adresele trecute în tabela de descriere a canalelor respective.¹ De asemenea, sunt resetate:

- poziția cursorului (în colțul din stânga sus);
- punctul grafic curent (la 0,0, stânga jos);
- contorul de linii înainte de apariția mesajului "Scroll?";
- atributele temporare (la cele permanente);
- dimensiunea zonei de editare (la 2 linii).

Subrutina afectează majoritatea registrelor precum și o parte din variabilele sistem.

¹Astfel se explică necesitatea ca rutinele care permit 64 coloane pe ecran (de exemplu SYS64) să funcționeze în IM2 și faptul că acestea ratează imediat după CLS (au nevoie de cel puțin 1/50s - PAUSE 1 pentru a forța adresa proprie în variabila de sistem CURCHL).

Punctul #0DAF (CL_ALL). Acest punct de intrare permite ștergerea întregului ecran (inclusiv a zonei de editare) cu atributele permanente curente. Subrutina nu necesită parametri de intrare. Spre deosebire de CLS, canalul #FD ('K') nu este inițializat. Sunt reșetate:

- poziția cursorului (stânga sus);
- punctul grafic curent (0,0);
- contorul de linii pentru "Scroll?".

Subrutina afectează majoritatea registrelor precum și o parte din variabilele de sistem.

Punctul #0E44 (CL_LINE). Subrutină folosită de CLS și CL_SCROLL pentru a șterge partea de jos a ecranului.

Parametrul de intrare este numărul primei linii care urmează să fie ștearsă, numerotată de la 0 de sus în jos. Acest parametru este dat în registrul B. Atributele folosite depind de variabila de sistem TV_FLAG. Dacă bitul 0 al acesteia este 0, atunci atributele sunt cele permanente ale ecranului, altfel sunt cele pentru *border*. Subrutina afectează majoritatea registrelor (cu excepția lui B), dar nu modifică variabilele sistem.

Un mod simplu de a șterge ecranul, fără a utiliza rutinele de sistem, este:

```

ECRAN      EQU #4000      : adresă de început ecran
LUNG_ECR   EQU #1800      : lungime zonă imagine
LUNG_ATTR  EQU #300       : lungime zonă atribute
ATRIBUT    EQU 8*7+0      : PAPER alb, INK negru
ENT $
LD HL,ECRAN
LD DE,ECRAN+1
LD BC,LUNG_ECR
LD (HL),#00
LDIR                      : umple zona cu 0
LD BC,LUNG_ATTR-1
LD (HL),ATRIBUT
LDIR                      : umple zona cu ATRIBUT
RET

```

3.3.5.2. Aprinderea unui pixel (PLOT)

Punctul de intrare cel mai util este #22E5 (PLOT_SUB) care realizează aprinderea sau stingerea unui pixel funcție de atributele de culoare temporare. Subrutina necesită următorii parametri:

- B — coordonata Y, cuprinsă între 0 și 175;
- C — coordonata X, cuprinsă între 0 și 255.

Depășirea valorii maxime produce ieșirea prin rutina de eroare (cu mesajul "Integer out of range").

Subrutina afectează majoritatea registrelor. Poziția grafică curentă (variabila de sistem COORDS) este modificată corespunzător. Dacă se dorește aprinderea pixelului fără modificarea poziției curente, se poate folosi punctul de

intrare #22E8. Dacă parametrii X și Y se află în stiva calculatorului aritmetic, atunci punctul de intrare poate fi #22DC. De remarcat că această subrutină este relativ lentă și prin urmare improprie pentru programe în timp real (animație). De asemenea, partea de jos a ecranului nu poate fi accesată.

3.3.5.3. Trasarea unei linii (DRAW)

Punctul de intrare ce poate fi folosit pentru o apelare comodă este #24BA (DW_LINE), cu următorii parametri:

- B — ABS(ΔY) modulul deplasării relative pe axa Y;
- C — ABS(ΔX) modulul deplasării relative pe axa X;
- D — SGN(ΔY) semnul deplasării relative pe axa Y (#01 pozitiv, #FF negativ);
- E — SGN(ΔX) semnul deplasării relative pe axa X (#01 pozitiv, #FF negativ).

Subrutina folosește PLOT_SUB. Linia generată are culoarea determinată de atributele temporare. Depășirea limitelor ecranului determină ieșirea prin rutina de eroare. Punctul inițial este cel dat de variabila de sistem COORDS. La ieșirea din subrutină DE este nemodificat (restul registrelor sunt alterate). Variabila de sistem COORDS este afectată în mod corespunzător.

Dacă parametrii X și Y sunt în stiva calculatorului aritmetic, atunci se poate folosi ca punct de intrare #24B7 (DRAW_LINE).

3.3.5.4. Trasarea unui cerc

Subrutina de trasare a unui cerc existentă în interpretorul Basic este deosebit de lentă deoarece folosește calculatorul aritmetic pentru a calcula poziția fiecărui punct de aproximare (cercul este aproximat cu un poligon în așa fel încât diferențele să fie mai mici de 2 pixeli). Din această cauză, folosirea sa din programe în cod mașină nu este recomandată. Punctul de intrare este #232D (DW_CIRCLE). Rutina necesită transferul parametrilor în stiva calculatorului aritmetic (trei valori numerice în ordinea X_centrul, Y_centrul și Raza).

3.3.5.5. Poziționarea atributelor de culoare

Atributele de culoare precum și modul în care funcționează rutinele de afișare se pot preciza folosind caracterele de control corespunzătoare trimise pe canalul a cărui culoare vrem să o schimbăm. Poziționările sunt temporare (afectează numai atributele temporare). Pentru a transforma atributele curente în atribute permanente se poate folosi punctul de intrare #1CAD (SET_PERMS). Subru-

tina consideră că atributele curente sunt ale ecranului (2) și nu ale părții de editare (1)¹. La ieșire, atributele temporare vor fi identice cu cele permanente.

3.3.5.6. Exemplu de utilizare

Programul care urmează șterge ecranul, apoi desenează un pătrat și o serie de cercuri concentrice în culori diferite. El este echivalent cu următorul program Basic:

```
10 CLS
20 INK 2
30 PLOT 125,33
30 DRAW 100,0: DRAW 0,100: DRAW -100,0: DRAW 0,-100
40 INK 0: OVER 1
60 FOR I=2 TO 76 STEP 2:
70 CIRCLE INK I-8*INT(I/8):175.88,I
80 NEXT I
```

Valorile numerice pentru CIRCLE trebuie reprezentate pe 5 octeți (se va folosi calculatorul aritmetic).

NR175	DEFB 0,0.175.0,0	: definirea celor 3 argumente ale lui
		: CIRCLE
NR88	DEFB 0,0.88.0,0	
RAZA	DEFB 0,0.2.0,0	
CLS	EQU #0D6B	: adresele rutinelor folosite
PLOT SUB	EQU #22E5	
DW LINE	EQU #24BA	
DW CIRCLE	EQU #232D	
SET_PERMS	EQU #1CAD	
CHAN OPEN	EQU #1601	
STACK_NUM	EQU #33B4	
INK	EQU #10	: caractere de control și culori
OVER	EQU #15	
RED	EQU 2	
BLACK	EQU 0	
PATRAT	DEFB 1,100,1,0	: 100,0 memorat ca SGN(X),ABS(X),SGN(Y),
		: ABS(Y)
	DEFB 1,0,1,100	: 0,100
	DEFB #FF,100,1,0	: -100,0
	DEFB 1,0,#FF,100	: 0,-100
	DEFB 0	: sfârșit desen
	ENT \$	: punctul de intrare
START	CALL CLS	: șterge ecranul
	LD A,#02	: INK 2
	CALL CHAN_OPEN	
	LD A,INK	
	RST #10	
	LD A,RED	
	RST #10	
	CALL SET_PERMS	: atribute permanente

¹Acest lucru presupune ca ultima scriere să se fi făcut pe canalul 2.

	LD BC,#217D	: Y=33(#21) ,X=125(#7D)
	CALL PLOT SUB	: PLOT 125,33
	LD IX.PATRAT	: pointer în vectorul de date
INCA	LD A,(IX+0)	: testează dacă nu s-a terminat
		: (SGN(X)<0!)
	OR A	
	JR Z,GATA	: dacă da, iese din ciclu
	LD E,A	: transferă parametrii
	LD C,(IX+1)	
	LD D,(IX+2)	
	LD B,(IX+3)	
	PUSH IX	: salvează pointerul
	CALL DW LINE	: DRAW X,Y
	POP IX	: avansează 4 octeți în vectorul de date
		: (următorul DRAW)
	LD BC,#0004	
	ADD IX,BC	
GATA	JR INCA	: continuă cu linia următoare
	LD A,INK	: INK 0
	RST #10	
	LD A,BLACK	
	RST #10	
	LD A,OVER	: OVER 1
	RST #10	
	LD A,#01	
	RST #10	
	CALL SET_PERMS	: attribute permanente
FOR	LD A,2	: primul cerc are raza 2
	LD (RAZA+2),A	: actualizează raza cercului
	PUSH AF	: salvează contorul ciclului
	LD A,INK	: modifică temporar attributele
	RST #10	
	POP AF	: culoarea cercului se află din contor
	PUSH AF	
	AND %00000111	: calculează culoarea A=A mod 8
	RST #10	
	LD HL,NR175	: pregătește apelul lui CIRCLE
	CALL STACK_NUM	: stiva trebuie să conțină X,Y,Z
	LD HL,NR88	: această instrucțiune este redundantă
		: deoarece HL pointează la ieseirea din
		: STACK_NUM pe octetul următor care este
		: chiar NR88
	CALL STACK_NUM	: oricum, acest lucru este valabil numai
		: în cazul de față
	LD HL,RAZA	: la fel ca mai sus, dar dacă ordinea de
		: definire ar fi fost alta, instruc-
		: țiunile erau necesare
	CALL STACK_NUM	
	CALL DW_CIRCLE	: CIRCLE 175,88,RAZA
	POP AF	: în A avem contorul
	ADD A,2	: următoarea valoare
	CP 76	: gata ?
	JR C,FOR	: dacă nu, reia de la FOR
	RET	

3.3.6. CALCULATORUL ARITMETIC

Calculatorul aritmetic este apelat de interpretorul Basic la evaluarea expresiilor. El constituie o entitate software de sine stătătoare. Toate rutinele de calcul folosesc o stivă specială pentru a primi și transmite date (această stivă nu

are nici o legătură cu stiva microprocesorului, fiind de fapt un vector de cuvinte de câte 5 octeți). Apelarea subrutinelor de calcul se face prin "instrucțiuni" ale calculatorului aritmetic (de fapt *hook*-coduri transmise ca parametri în linie, v. §3.4.4). Există un număr de 82 de astfel de "instrucțiuni" care acoperă toate funcțiile disponibile din Basic, comparații, salturi, rutine de conversie, etc. Calculatorul aritmetic dispune de asemenea de un număr de "registre" de 5 octeți, care sunt folosite de anumite rutine de calcul (din acest motiv nu sunt utile programatorului). Totodată sunt definite câteva constante frecvent utilizate în calcule (0, 1, 0.5, $\pi/2$, 10).

Apelul calculatorului aritmetic se face printr-o instrucțiune RST #28 urmată de "instrucțiuni" aritmetice. Calculul se termină cu "instrucțiunea" END_CALC (#38). Controlul se întoarce programului apelant la adresa următoare lui END_CALC. Este necesar ca înainte de apelul calculatorului să se salveze registrul H'L iar apoi acesta să fie refăcut dacă se intenționează revenirea în Basic.

#00 - JUMP_TRUE	#10 - STR & NO	#20 - COS	#30 - NOT	#86 - SERIES_06
#01 - EXCHANGE	#11 - STR I EQL	#21 - TAN	#31 - DUPLICATE	#88 - SERIES_08
#02 - DELETE	#12 - STR GR EQ	#22 - ASN	#32 - N MOD M	#8C - SERIES_0C
#03 - SUBSTRACT	#13 - STRS NEQL	#23 - ACS	#33 - JUMP	#A0 - STK_ZERO
#04 - MULTIPLY	#14 - STR GRTR	#24 - ATN	#34 - STK_DATA	#A1 - STK_ONE
#05 - DIVISION	#15 - STR LESS	#25 - LN	#35 - DEC_JR_NZ	#A2 - STK_HALF
#06 - TO_POWER	#16 - STRS EQL	#26 - EXP	#36 - LESS_0	#A3 - STK_HALFPI
#07 - OR	#17 - STRS_ADD	#27 - INT	#37 - GREATER_0	#A4 - STK_TEN
#08 - NO & NO	#18 - VAL\$	#28 - SQR	#38 - END_CALC	#C0 - ST_MEM_0
#09 - NO I EQL	#19 - USR \$	#29 - SGN	#39 - GET_ARGT	#C1 - ST_MEM_1
#0A - NO GR EQ	#1A - READ_IN	#2A - ABS	#3A - TRUNCATE	#C2 - ST_MEM_2
#0B - NOS NEQL	#1B - NEGATE	#2B - PEEK	#3B - FP_CALC_2	#C3 - ST_MEM_3
#0C - NO GRTR	#1C - CODE	#2C - IN	#3C - E_TO_FP	#C4 - ST_MEM_4
#0D - NO LESS	#1D - VAL	#2D - USR_NO	#3D - RE_STACK	#C5 - ST_MEM_5
#0E - NOS EQL	#1E - LEN	#2E - STR\$		#E0 - GET_MEM_0
#0F - ADDITION	#1F - SIN	#2F - CHR\$		#E1 - GET_MEM_1
				#E2 - GET_MEM_2
				#E3 - GET_MEM_3
				#E4 - GET_MEM_4
				#E5 - GET_MEM_5

Pe lângă funcțiile de mai sus mai sunt necesare un număr de rutine care să manipuleze numere în reprezentare pe 5 octeți. Cele mai utile sunt:

1. Subrutine care "mută" numere în stiva calculatorului aritmetic. Aceste subrutine "cresc" stiva, adăugând un nou număr (5 octeți). Dacă nu mai este loc în stivă (stiva calculatorului aritmetic este alocată în memoria Basic), atunci controlul nu mai revine programului apelant, ci se iese prin rutina de eroare, cu mesajul "Out of memory".

STK_STORE - #2AB6

Transferă un număr (sau parametrul unui șir) din registrele procesorului în stivă. Registrele folosite sunt A, BC și DE (în ordinea A,E,D,C,B). Adresa vârfului stivei este actualizată. La ieșire, HL este egal cu variabila de sistem STKEND.

STACK_NUM - #33B4

Copiază un număr (5 octeți) de la adresa din HL în stiva

calculatorului aritmetic, actualizând pointer-ul la adresa vârfului stivei. La ieșire HL este incrementat cu 5, iar DE este egal cu variabila de sistem STKEND.

STACK_BC - #2D2B

Convertește conținutul registrului BC (valoare binară absolută) în format aritmetic (5 octeți). Rezultatul este lăsat în stiva calculatorului aritmetic.

STACK_A - #2D28

Convertește conținutul lui A (valoare binară absolută) în format aritmetic. Rezultatul este depus în stiva calculatorului aritmetic.

STK_DIGIT - #2D22

Convertește caracterul din A (dacă este o cifră!) în format aritmetic. Dacă codul din A nu este o cifră, atunci stiva calculatorului aritmetic nu este afectată. La ieșirea din subrutină, CY=0 dacă conversia a reușit și CY=1 dacă valoarea din A nu reprezintă o cifră.

DEC_TO_FP - #2C9B

Convertește un șir de caractere reprezentând un număr, în format intern, pe 5 octeți. Adresa de început a șirului se comunică prin variabila de sistem CH_ADD, a cărei valoare trebuie salvată și apoi restaurată (CH_ADD este folosită de interpretorul Basic pentru a memora adresa caracterului ce urmează a fi interpretat). Registrul A trebuie să conțină primul caracter din șir (cel de la adresa din CH_ADD).

2. Subrutine care "mută" numere din stiva calculatorului aritmetic. În cele ce urmează vom presupune că valoarea aflată în vârful stivei este x. Aceste subrutine "scad" stiva.

STK_FETCH - #2BF1

Transferă vârful stivei în registrele microprocesorului. Se folosesc BC, DE și A în această ordine: B = (STKEND), C=(STKEND-1) ... A=(STKEND-4). Cei 5 octeți transferați pot reprezenta un număr sau parametrii unui șir.

FP_TO_BC - #2DA2

Convertește vârful stivei în format întreg. La ieșire registrul BC conține valoarea ABS(INT(x)). Numărul este șters din stivă. Parametrii de întoarcere sunt:

BC = ABS(INT(x))

Z=0 dacă $x < 0$

Z=1 dacă $x \geq 0$

CY=0 dacă conversia a fost făcută

CY=1 depășire ($\text{ABS}(x) > 65535.5$; BC este nedefinit!)

FP_TO_A - #2DD5

Convertește vârful stivei în format întreg. La ieșire registrul A conține ABS(INT(x)). Numărul este șters din stivă. Parametrii de întoarcere sunt:

$A = \text{ABS}(\text{INT}(x))$
 $Z = 0$ dacă $x < 0$
 $Z = 1$ dacă $x \geq 0$
 $CY = 0$ dacă conversia a putut fi făcută
 $CY = 1$ depășire ($\text{ABS}(x) > 255.5$; A este nedefinit!)

3. Diverse

SET_WORK - #16BF
 Inițializează stiva calculatorului aritmetic și șterge spațiul de lucru.
 SET_STK - #16C5
 Inițializează stiva calculatorului aritmetic (fără a șterge spațiul de lucru).
 PF_INT_FP - #2DE3
 Tipărește numărul din vârful stivei. Oricare ar fi valoarea lui x formatul de afișare nu depășește 14 cifre. Subrutina se poate folosi atât pentru a afișa un număr cât și pentru a-l converti în șir (STR\$(x)).

3.3.6.1. Exemplu de utilizare

```

; Rezolvarea ecuației de gradul 2 prin apelarea calculatorului aritmetic
; A*x^2+B*x+C=0
      ORG 64000
      ENT $
      JR START
; definirea coeficienților
COEFA  DEFM '-0.1'      ; coeficienții sunt definiți ca șiruri de caractere terminate cu CR
      DEFB #0D
      ORG COEFA
      DEFS 16           ; rezervă maximum 16 octeți pentru un număr
COEFB  DEFM '0.98823'
      DEFB #0D
      ORG COEFB
      DEFS 16
COEFC  DEFM '-1.59'
      DEFB #0D
      ORG COEFC
      DEFS 16
; rutine ROM folosite
STACK BC EQU #2D2B
STK_FETCH EQU #2BF1
STK_STORE EQU #2AB6
PF_INT_FP EQU #2DE3
DEC_TO_FP EQU #2C9B
SET_WORK EQU #16BF
CHAN_OPEN EQU #1601
; variabile de sistem
CH_ADD EQU 23645
; punctul de intrare
START  EXX              ; salvarea H'L' este obligatorie pentru reîncărcarea în Basic
      ; toarcerea în Basic
  
```



```

PUSH HL
EXX
LD HL,COEFA      : tipărește coeficienții sub forma A=-0.1 ...
CALL STKNUM
LD A,"A"
CALL PR
LD HL,COEFB
CALL STKNUM
LD A,"B"
CALL PR
LD HL,COEFC
CALL STKNUM
LD A,"C"
CALL PR
LD HL,COEFB      : pregătește calculul discriminantului
CALL STKNUM      : D=B*B-4*A*C
LD BC,#0004
CALL STACK BC
LD HL,COEFC
CALL STKNUM
LD HL,COEFA
CALL STKNUM
                                : stiva: B 4 C A
RST #28          : apel calculator aritmetic
DEFB #31         : DUPLICATE      : stiva: B 4 C A A
DEFB #00,CONT-$  : JUMP_TRUE CONT : stiva: B 4 C A

DEFB #38         : END CALC
RST #08         : dacă A=0 raportează 'Parameter error'
DEFB #19

CONT
DEFB #04         : MULTIPLY      : stiva: B 4 C*A
DEFB #04         : MULTIPLY      : stiva: B 4*C*A
DEFB #01         : EXCHANGE      : stiva: 4*C*A B
DEFB #31         : DUPLICATE      : stiva: 4*C*A B B
DEFB #04         : MULTIPLY      : stiva: 4*C*A B*B
DEFB #01         : EXCHANGE      : stiva: B*B 4*C*A
DEFB #03         : SUBTRACT      : stiva: B*B-4*C*A=D
DEFB #31         : DUPLICATE      : stiva: D D
DEFB #36         : LESS_0 ?      : stiva: D D<0
DEFB #30         : NOT -         : stiva: D D>=0
DEFB #00,REALE-$: JUMP_TRUE REALE : stiva: D

IMAG
DEFB #1B         : NEGATE      : stiva: -D
DEFB #28         : SQR        : stiva: SQR(-D)
DEFB #38         : END_CALC
LD BC,#02
CALL STACK BC    : stiva: SQR(-D) 2
LD HL,COEFA
CALL STKNUM      : stiva: SQR(-D) 2 A
RST #28          : apel calculator aritmetic
DEFB #04         : MULTIPLY      : stiva: SQR(-D) 2*A
DEFB #05         : DIVISION      : stiva: SQR(-D)/(2*A)
DEFB #38         : END_CALC
CALL MBP2A       : calculează -B/2A (partea reală)
CALL STORE       : și o stochează în MEMORY
LD A,"I"         : tipărește partea imaginară
CALL PR          : I=SQR(-D/2A)
CALL RECALL      : partea reală pe stivă
LD A,"R"         : tipărește partea reală
CALL PR          : R=-B/2A
JP RETURN

REALE
DEFB #28         : SQR        : stiva: SQR(D)
DEFB #38         : END_CALC

```



```

LD BC,#02
CALL STACK BC           : stiva: SQR(D) 2
LD HL,COEFA
CALL STKNUM             : stiva: SQR(D) 2 A
RST #28
DEFB #04                : MULTIPLY      : stiva: SQR(D) 2*A
DEFB #05                : DIV          : stiva: SQR(D)/(2*A)
DEFB #31                : DUPLICATE     : stiva: SQR(D)/(2*A)
                        : SQR(D)/(2*A)
DEFB #1B                : NEG          : stiva: SQR(D)/(2*A)
                        : -SQR(D)/(2*A)
DEFB #38                : END_CALC
CALL MBP2A              : calculează -B/2A
                        : stiva: SQR(D)/(2*A)
                        : -SQR(D)/(2*A) -B/(2*A)
RST #28                : apel calculator aritmetic
DEFB #31                : DUPLICATE
                        : stiva: SQR(D)/(2*A) -SQR(D)/(2*A)
                        : -B/(2*A) -B/(2*A)
DEFB #38                : END_CALC
CALL STORE              : stochează -B/2A (ultimul!)
RST #28
DEFB #0F                : ADDITION      : stiva: SQR(D)/(2*A) -SQR(D)
                        : /(2*A)+(-B/(2*A))=X1
DEFB #01                : EXCHANGE      : stiva: X1 SQR(D)/(2*A)
DEFB #38                : END_CALC
CALL RECALL             : stiva: X1 SQR(D)/(2*A)
                        : -B/(2*A)
RST #28
DEFB #0F                : ADDITION      : stiva: X1
                        : SQR(D)/SQR(2*A)+(-B/(2*A))=X2
DEFB #38                : END_CALC
CALL STORE              : salvează temporar X2 în MEMORY
LD A,"X"                : tipărește soluția X1 primul
CALL PR
CALL RECALL             : X2 înapoi pe stivă
LD A,"X"                : tipărește X2
CALL PR

RETURN
EXX                     : înainte de întoarcerea în Basic se reface
                        : registrul H'L'
POP HL
EXX
RET
; definirea subrutinelor folosite
MBP2A : calculul lui -B/2A
LD HL,COEFB
CALL STKNUM             : stiva: B
LD BC,#0002
CALL STACK BC           : stiva: B 2
LD HL,COEFA
CALL STKNUM             : stiva: B 2 A
RST #28
DEFB #04                : MULTIPLY      : stiva: B 2*A
DEFB #05                : DIVISION      : stiva: B/(2*A)
DEFB #1B                : NEG          : stiva: -B/(2*A)
DEFB #38                : END_CALC
RET
STKNUM                  : pune în stivă un număr în ASCII din memorie de la adresa HL
XOR A                  : numărul poate să înceapă cu "-"
LD (NEGAT),A           : memorează acest lucru temporar în NEGAT
LD A,(HL)
CP "-"
JR NZ,STK              : număr pozitiv
LD A,#01

```



```

LD (NEGAT).A
INC HL          : pentru numerele negative se sare peste minus
STK
LD DE,(CH_ADD) : salvează poziția curentă a interpretorului
PUSH DE
LD (CH_ADD).HL : folosește interpretorul pentru a realiza
                : conversia
LD A.(HL)
CALL DEC_TO_FP : nr. este acum în stivă
POP DE
LD (CH_ADD).DE : restaurează vechea valoare a lui CH_ADD
LD A.(NEGAT)   : a avut "-" în față
AND A
RET Z          : dacă nu, conversia este încheiată
RST #28        : altfel cheamă calculatorul aritmetic
DEFB #1B       : NEGATE schimbă semnul numărului din stivă
DEFB #38       : END_CALC
RET
STORE : decartează vârful stivei și îl stochează la MEMORY
CALL STK_FETCH : vârful stivei în A,BC și DE
LD HL.MEMORY   : adresa de salvare
LD (HL).A      : stochează registru cu registru
INC HL
LD (HL).E
INC (HL)
LD (HL).D
INC HL
LD (HL).C
INC HL
LD (HL).B
RET
RECALL : copiază valoarea de la MEMORY în vârful stivei
LD HL.MEMORY
LD A.(HL)
INC HL
LD E.(HL)
INC HL
LD D.(HL)
INC HL
LD C.(HL)
INC HL
LD B.(HL)
CALL STK_STORE
RET
PR      : tipărește o variabilă sub forma "A=valoare"
RST #10 : tipărește numele
LD A,"=" : apoi semnul "="
RST #10
CALL PF_INT_FP : apoi numărul
LD A,#0D
RST #10 : avansează pe rândul următor
CALL SET_WORK : inițializează stiva calculatorului aritmetic
RET
: variabile folosite
NEGAT
MEMORY
DEFB 0
DEFB 5

```

3.3.7. CITIREA TASTATURII

Calculatorul *Spectrum* are o tastatură matriceală a cărei stare este verificată (în funcționare normală) de 50 de ori pe secundă de către rutina de tratare a întreruperilor mascabile. Pentru a afla starea tastaturii este deci

suficient să activăm întreruperile și să verificăm variabilele de sistem corespunzătoare. Acestea sunt:

LAST_K — 23560 memorează codul ultimei taste apăsate;

FLAGS — 23611 bitul 5 din FLAGS semnalizează faptul că a fost apăsată o tastă nouă.

O posibilă rutină GET_KEY care să aștepte apăsarea unei taste este:

```

LAST_K      EQU 23560      : variabile de sistem folosite
FLAGS       EQU 23611
IYBASE      EQU #5C3A
ENT $
GET_KEY     : rutina întoarce codul caracterului în
            : registrul A
EI
WAIT        HALT
            BIT 5, (IY+FLAGS-IYBASE)
            JR Z, WAIT
            LD A, (LAST_K)
            RES 5, (IY+FLAGS-IYBASE)
            RET

```

Dezavantajul acestei subrutine este că nu întoarce și caracterele de schimbare de mod. Pentru schimbarea de mod, trebuie modificate corespunzător următoarele variabile de sistem:

MODE — 23617 memorează modul curent (K,L,C,E sau G);

FLAGS — 23611 bitul 3, valoarea zero semnalizează modul K.

3.3.8. TRATAREA ERORILOR

Modul standard de semnalizare a erorilor folosit de interpretorul Basic este apelul rutinei de la adresa #0008 (prin RST #08) urmat de codul erorii. Adresa rutinei de tratare a erorilor se află în RAM și poate fi deci modificată pentru a obține altceva decât mesajul de eroare.

Rutina de la adresa #0008 este:

```

#0008 ERROR_1 LD HL, (CH_ADD) : adresa simbolului care a provocat eroarea este
                                : salvată în X_PTR
#000B          LD (X_PTR), HL
#000E          JR #0053       : ERROR_2
...
#0053 ERROR_2  POP HL         : adresa următoare lui RST #08 în HL
#0054          LD L, (HL)      : numărul erorii
#0055 ERROR_3  LD (ERR_NR), L
#0058          LD SP, (ERR_SP) : poziționează stiva astfel încât la ieșirea din
                                : SET STK controlul să revină rutinei de eroare
#005B          JP #16C5       : ieșire prin SET STK. Șterge stiva
                                : calculatorului aritmetic.

```

Pentru a intercepta rutina de eroare este necesar să modificăm conținutul stivei procesorului. Mai exact, trebuie să forțăm la adresa dată de (ERR_SP), adresa noii rutine de eroare (în mod normal vechea adresă este #1303 - MAIN_4).

3.3.8.1. Exemplu de interceptare a rutinei de eroare

Următoarea rutină forțează execuția unei anumite linii Basic atunci când apare o eroare (Atenție! Mesajul O.K. este văzut de sistem tot ca o eroare și deci programul Basic nu mai poate fi oprit):

```

ERR_SP      ORG 64400
            EQU 23613      ; adresa adresei vârfului stivei în caz de
                        ; eroare
ERR_NR      EQU 23610      ; nr eroare -1
SPARE1      EQU 23681      ; un octet nefolosit în mod normal de inter-
                        ; pretorul Basic
SPARE2      EQU 23728      ; 2 octeți nefolosiți
OLDPPC      EQU 23662      ; nr liniei de la care se reia execuție la
                        ; CONTINUE
OSPPC       EQU 23664      ; nr instrucțiunii din linie de la care se reia
                        ; execuția
E_LINE      EQU 23641      ; adresa ultimei comenzi
BREAK_KEY   EQU #1F54      ; rutina testează BREAK. CY=0 dacă s-au apăsât
                        ; simultan tastele SS și SPACE
MAKE_ROOM   EQU #1655      ; rutina deschide un nou spațiu de lucru de
                        ; dimensiune BC după adresa din HL. La ieșire
                        ; (HL+1) pointează pe începutul zonei create, iar
                        ; DE pe ultimul octet din ea.
MAIN_3      EQU #12CF      ; punct de intrare în bucla principală a
                        ; interpretorului Basic după verificarea sintaxei
ENT $        ; punctul de intrare. El forțează noua adresă de
                        ; revenire în caz de eroare
            LD DE,ERR NOU   ; adresa rutinei
            LD HL,(ERR_SP)  ; adresa vârfului stivei în caz de eroare
            LD (HL),E
            INC HL
            LD (HL),D
            RET

ERR_M        ; rutina de tratare a erorii
            LD A,(ERR_NR)   ; nr erorii este salvat în SPARE1 pentru a putea
                        ; fi testat din Basic
            INC A
            LD (SPARE1),A
            LD BC,(SPARE2)  ; nr liniei la care urmează să se execute saltul
                        ; în caz de eroare
            LD (OLDPPC),BC  ; pregătește parametrii pentru CONTINUE
            LD A,#FF        ; nici o eroare
            LD (ERR_NR),A
            INC A           ; A=0
            LD (OSPPC),A    ; nr instrucțiunii din cadrul liniei
            LD HL,(E_LINE)  ; adresa zonei de editare
            LD BC,#0002
            CALL MAKE_ROOM  ; sunt necesari 2 octeți pentru a da instruc-
                        ; țiunea CONTINUE
            EX DE,HL
            LD (HL),232     ; codul instrucțiunii CONTINUE
            INC HL
            LD (HL),0D      ; ENTER
BTEST       CALL BREAK_KEY  ; așteaptă eliberarea tastei BREAK (se evită
                        ; astfel apelul recursiv)
            JP C,MAIN_3     ; executa instrucțiunea curentă care este chiar
                        ; CONTINUE
            JR BTEST
    
```


Pentru a testa rutina se poate folosi următorul program Basic:

```
10 LET ERL=9000
20 POKE 23728,ERL-256*INT(ERL/256)
30 POKE 23729,INT(ERL/256)
50 RANDOMIZE USR 64400
```

9000 PRINT "A apărut o eroare!!": BEEP 0.3,1: PAUSE 200: RUN 10

3.3.9. TRATAREA ÎNTRERUPERILOR MASCABILE

Sistemul de întreruperi al *Spectrum*-ului este deosebit de simplu. Logica video generează o întrerupere la fiecare semicadru, deci 50 de întreruperi pe secundă. Microprocesorul funcționează în modul 1 de întreruperi, deci adresa rutinei de tratare a întreruperilor este #0038. Din păcate rutina nu poate fi interceptată direct, neavând nici un pointer în RAM. Funcțiile sale sunt:

- să incrementeze "ceasul de timp real" al calculatorului (variabila de sistem FRAMES);

- să scaneze tastatura pentru a vedea dacă există o tastă apăsată; dacă da, atunci variabilele LAST_K, FLAGS și KSTATE sunt modificate corespunzător.

Pentru a putea intercepta întreruperea (lucru necesar pentru un foarte mare număr de aplicații) trebuie utilizat modul 2 de întreruperi. În acest mod, adresa rutinei de tratare a întreruperii este dată de o tabelă ce se poate afla oriunde în memorie. Pointer-ul spre această tabelă este format de registrul I (octetul cel mai semnificativ) și de perifericul care a cerut întreruperea (octetul cel mai puțin semnificativ). Din păcate magistrala de date nu are în momentul lansării întreruperii o valoare precizată (nu există nici un periferic care să forțeze datele pe magistrală). Pentru majoritatea calculatoarelor *Spectrum*, precum și pentru compatibilele românești, această valoare este #FF. Din motive de compatibilitate nu ne putem însă baza pe această valoare. Soluția constă în utilizarea unei tabele de 257 de octeți identici. În acest fel adresa rutinei de tratare va trebui să aibă octetul superior egal cu cel inferior (de exemplu, #FEFE, #FBFB etc). Dacă vrem ca interpretorul Basic să funcționeze în continuare este necesară "înlanțuirea" rutinei cu cea standard (#0038).

3.3.9.1. Exemplu de interceptare a întreruperii mascabile

Exemplul interceptează întreruperea pentru a afișa în colțul din dreapta sus ora curentă în formatul hh:mm:ss.

TABELA	ORG 65040	
LUNGTAB	EQU #FD00	: adresa tabelii cu adrese pentru modul 2
INTERUP	EQU #0101	: lungimea tabelii
SEPARAT	EQU #F7F7	: adresa noii rutine de întreruperi
ATRCEAS	EQU ":-" - "0"	: separatorul ":-"
ADR_ECRAN	EQU #C7	: FALSH 1, BRIGHT 1, INK 7, PAPER 0
ADR_ATTR	EQU #4018	: adresa în ecran de unde începe afișarea
	EQU #5818	: adresa corespunzătoare în zona de atribute video

OLD INT
CHARS

```

EQU #0038      : adresa rutinei de sistem
EQU #5C36      : adresa generatorului de caractere
ENT $          : punctul de intrare pentru declanșarea ceasului
DI
LD HL.TABELA   : pregătește tabela cu adresa rutinei de în-
                  : trerupere
PUSH HL
POP DE
INC DE
LD BC.LUNG
LD (HL). INTERRUPT / #100 : 257 octeți vor fi umpluți cu #F7
LDIR
LD A.TABELA / #100      : I=#FD - începutul tabelui
LD I.A
IM 2
EI
RET

ORG INTERRUPT
PUSH IX      : salvează starea procesorului
PUSH AF      : întreruperea trebuie să fie transparentă
PUSH BC
PUSH DE
PUSH HL
LD A.(SUTIMI) : numără întreruperile
DEC A
LD (SUTIMI).A
JP NZ.EXITINT : continuă numai o dată pe secundă
LD A.50        : 50 întreruperi = o secundă
LD (SUTIMI).A
LD A.(SECEUNDE) : incrementează secundarul
AND A
ADC A.#01      : direct în aritmetică BCD
DAA
LD (SECEUNDE).A
CP #60         : 60 secunde = 1 minut
JP NZ.DISPLAY  : afișează direct, nu a trecut un minut
XOR A
LD (SECEUNDE).A : pune secundarul pe 0
LD A.(MINUTE)  : numărul de minute
AND A
ADC A.#01      : incrementează minutarul
DAA
LD (MINUTE).A
CP #60         : 60 minute = 1 ora
JP NZ.DISPLAY  : afișează direct, nu a trecut o oră
XOR A
LD (MINUTE).A  : resetează minutarul
LD A.(ORE)     : numărul de ore
AND A
ADC A.#01      : incrementează numărul de ore
DAA
LD (ORE).A
CP #24         : dacă a trecut o zi, nr de ore = 0
JP NZ.DISPLAY
XOR A
LD (ORE).A

DISPLAY
LD IX.ADR_ECRAN : pregătește adresa zonei unde se afișează
                  : ceasul
LD A.(ORE)
CALL DISP_2DIGIT : întâi afișează numărul de ore
LD A.SEPARAT
CALL DISP_1DIGIT : afișează ':'
LD A.(MINUTE)
CALL DISP_2DIGIT : apoi numărul de minute

```



```

LD A,SEPARAT
CALL DISP_1DIGIT: încă o dată separatorul
LD A,(SECUNDE)
CALL DISP_2DIGIT: numărul de secunde
LD HL,ADR_ATTR : actualizează și zona de atribute
LD B.#08       : 8 caractere

FILLATTR
LD (HL).ATRCAS
INC HL
DJNZ FILLATTR

EXITINT
POP HL          : restaurează starea procesorului
POP DE
POP BC
POP AF
POP IX
JP OLD_INT      : ieșire prin sistem

DISP_2DIGIT
: subrutină pentru afișarea unui număr în format BCD (două cifre)
PUSH AF        : salvează temporar valoarea ce trebuie afișată
SRL A          : păstrează numai o niblă (cea mai semnifica-
: tivă) în poziția cea mai puțin semnificativă

SRL A
SRL A
SRL A
CALL DISP_1DIGIT: afișează o cifră
POP AF
AND #0F        : cealaltă niblă
CALL DISP_1DIGIT
RET

DISP_1DIGIT
PUSH IX        : salvează temporar adresa de afișare
LD HL,(CHARS) : adresa generatorului de caractere
LD DE.#0180    : deplasamentul relativ în generatorul de
: caractere al cifrei "0"
ADD HL,DE      : HL pointează pe începutul descrierii lui "0"
EX HL,DE
LD L,A        : cifra ce trebuie afișată
LD H,0
ADD HL,HL     : fiecare caracter ocupă 8 octeți HL = A*2
ADD HL,HL     : HL = A*4
ADD HL,HL     : HL = A*8
ADD HL,DE     : HL = A*8 + adresă "0"
LD DE.#0100   : diferența de adresă între 2 octeți alăturați
: în ecran pe verticală
LD B,8        : un caracter are 8 octeți

DISP_CAR
LD A,(HL)     : din generator
XOR #FF       : INVERSE 1
LD (IX),A     : în ecran
ADD IX,DE     : IX=IX+100 adresa următorului octet în ecran
INC HL        : HL=HL+1 adresa următorului octet în generato-
: rul de caractere
DJNZ DISP_CAR
POP IX        : se reface poziția
INC IX        : următorul caracter în următoarea coloană
RET

```

3.4. TEHNICI DE PROGRAMARE

Învățarea unui limbaj de programare sau a unui program complex (de exemplu a unui procesor de texte), în general a oricărui instrument software,

pare dificilă atunci când se ignoră un fapt foarte simplu: majoritatea problemelor se rezolvă cu o minoritate a instrucțiunilor. Multe instrucțiuni vor fi folosite foarte rar sau chiar deloc. În consecință, nu merită investit efortul considerabil de a învăța toate instrucțiunile înainte de a trece la lucru. Preferabil este de a ataca probleme concrete cu un set minim de instrucțiuni "de bază", exact cum în comunicarea într-o limbă străină este adeseori suficient un vocabular de bază. Din păcate, aproape nici un manual nu stabilește un asemenea set, păstrând neutralitatea în ceea ce privește importanța relativă a instrucțiunilor. Neutralitatea este explicabilă prin faptul că "setul de bază" variază de la o aplicație la alta și de la un stil de programare la altul. Singurul în măsură să stabilească "setul de bază" este chiar cel care învață, pentru că el își formează un stil propriu și cunoaște totodată cazurile concrete de aplicare.

Ne-am propus în acest subcapitol să prezentăm unele "tehnici de programare", adică un set de reguli de construcție care stau la baza proiectării programelor într-un anumit limbaj. Aceste reguli se deprind din experiența practică și au o importanță cu atât mai mare, cu cât nivelul limbajului este mai coborât, mai aproape de mașină. În cazul limbajului de asamblare, importanța lor este în orice caz considerabilă și se traduce prin efortul și timpul consumat de un programator începător, din momentul în care cunoaște sintaxa și semantica tuturor instrucțiunilor microprocesorului, până în momentul în care reușește să scrie un program util. Pe de altă parte, tehnicile de programare acumulate reprezintă explicația pentru care scrierea unui program pare din ce în ce mai ușoară pe măsură ce experiența de programare crește. Dacă recurgem din nou la comparația cu o limbă străină, setul de instrucțiuni ar putea fi echivalat cu vocabularul, iar tehnicile de programare cu regulile de construcție gramaticală și stilistică necesare elaborării unei fraze.

Din păcate, majoritatea programatorilor își însușesc aceste tehnici în mod reflex și nu reușesc întotdeauna să le conceptualizeze în scopul publicării lor (sau nu au un interes să o facă). În cele ce urmează vom încerca să prezentăm cele mai importante tehnici de programare dobândite de autori în cei aproximativ 12 ani de experiență de programare a microprocesorului Z80.

Un aspect care nu trebuie neglijat este faptul că tehnicile de programare concentrează o anumită experiență practică și prin urmare sunt utile doar celor care învață pentru a aplica, nu și celor care învață pentru a ști pur și simplu. Pentru cititorii în criză de subiecte de aplicare, am inclus în acest capitol și câteva exerciții simple.

Pentru un studiu eficient al acestui capitol, recomandăm consultarea în paralel a cel puțin uneia din următoarele lucrări: [29], [37], [2].

3.4.1. TRANSFERURI PE 16 BIȚI

Deși microprocesorul Z80 este de 8 biți, el operează cu registre duble și execută mai multe instrucțiuni pe 16 biți: ADD, ADC, SBC, INC, DEC, PUSH, POP. Motivul acestei extinderi este clar: magistrala de adrese fiind de 16 biți, toate

calculele de adrese necesită astfel de instrucțiuni. Mai curios la prima vedere este faptul că tocmai instrucțiunile de transfer în modul direct registru pe 16 biți nu au fost prevăzute în setul Z80. Există numai instrucțiuni de transfer pe 16 biți în modurile imediat și direct memorie și anume:

```
LD rr,nn      : transfer pe 16 biți în modul imediat
LD rr,(nn)    : ... și în modul direct memorie
```

De asemenea, există unele instrucțiuni de transfer pe 16 biți spre registrul special SP (indicatorul de stivă):

```
LD SP,HL
LD SP,IX
LD SP,IY
```

În schimb, nu sunt valide în limbajul de asamblare Z80 construcții de forma:

```
LD HL,DE      : incorrect
LD BC,(HL)    : incorrect
```

Pentru transferuri directe în registre de 16 biți, unii programatori sunt tentați să folosească o secvență PUSH-POP, ca în exemplul următor:

```
PUSH DE      : LD HL,DE în 21 T
POP HL
```

Datorită incrementării și decrementării implicite a stivei și folosirii memoriei pentru stocare temporară, această soluție, din păcate destul de răspândită, durează prea mult. Recomandăm următoarea soluție, de 2,5 ori mai rapidă:

```
LD H,D       : LD HL,DE în 8 T.
LD L,E
```

În cazul în care unul sau ambele registre implicate în transfer este un registru de index (IX sau IY), metoda PUSH-POP rămâne singura aplicabilă. Cu toate că Z80 execută și transferuri pe jumătăți de registre IX și IY (de exemplu: LD E,IXL / LD D,IXH), aceste instrucțiuni nu apar în documentația de referință a microprocesorului, făcând parte din așa numitele instrucțiuni "ascunse". Prin urmare, majoritatea asamblor nu le recunosc¹ și utilizarea lor trebuie evitată.

Transferul cu adresare indirectă pe 16 biți se face utilizând ca pointer doar registrele HL, IX sau IY. De exemplu, LD DE,(HL) se poate efectua prin secvența următoare:

¹Singurul asamblor pe *Spectrum* care recunoaște instrucțiunile de transfer pe jumătăți de registre de index este EDITAS (C)1982 Picturesque.


```
LD E,(HL)      : LD DE.(HL) în 26 T
INC HL
LD D,(HL)
INC HL          : sau DEC HL dacă dorim păstrarea conținutului
                : inițial în HL
```

Dacă registrul HL este ocupat, se poate recurge la comutarea registrelor alternative, prin EXX. Dacă și H'L' este ocupat, atunci se va recurge la IX sau IY, ca în exemplul următor:

```
LD E,(IX)      : LD DE.(IX) în 38 T
LD D,(IX+1)
```

Exemple

a) Exemplul care urmează ilustrează incrementarea unei variabile întregi reprezentate pe 2 octeți (16 biți), aducând mai întâi conținutul variabilei VAR în registrul DE, incrementând acest registru, apoi transferând la loc în memorie noua valoare:

```
LD HL,VAR
LD E,(HL)
INC HL
LD D,(HL)      : LD DE.(HL)
INC DE         : incrementarea propriu-zisă a variabilei
LD (HL),D      : procedând simetric se economisesc instrucțiuni
                : nile INC/DEC
DEC HL
LD (HL),E
```

```
VAR    DEFW 999      : după rularea secvenței de mai sus, valoarea
                    : variabilei VAR va fi 1000
```

b) Ne propunem să încărcăm registrele BC, DE și HL cu aceeași valoare, să spunem 555, folosind mai întâi modul de adresare imediat:

```
LD BC,555
LD DE,555
LD HL,555      : folosind modul de adresare imediat, secvența
                : durează 30 T
```

Folosind transferuri pe 16 biți, secvența devine:

```
LD BC,555
LD E,C
LD D,B
LD L,C
LD H,B         : folosind transferuri pe 16 biți, secvența
                : durează 26 T
```

În concluzie, cea de-a doua variantă, deși pare mai neclară, este ceva mai rapidă și totodată mai eficientă, deoarece schimbarea valorii "555" cu altă valoare se face o singură dată în programul sursă.

c) Ne propunem să încărcăm registrele HL, IX și IY cu aceeași valoare, de exemplu 555, folosind modul de adresare imediat:

```
LD HL,555
LD IX,555
LD IY,555      ; folosind modul de adresare imediat, secvența
                ; durează 38 T
```

Folosind transferuri pe 16 biți prin metoda PUSH-POP (singura aplicabilă în acest caz), obținem:

```
LD HL,555
PUSH HL
POP IX
PUSH HL
POP IY          ; folosind transferuri pe 16 biți, secvența
                ; durează 60 T
```

Observăm că lucrurile stau cu totul altfel decât la exemplul precedent. La registrele index, transferurile pe 16 biți sunt foarte defavorabile. Ca o concluzie generală, se poate afirma că metoda PUSH-POP trebuie evitată cu orice preț.

Exerciții

Programați o secvență care să adune două variabile întregi pe 16 biți, denumite OPER1 și OPER2, rezultatul depunându-se într-o variabilă REZ. Cele trei variabile pot fi alocate oriunde în memorie, deci nu trebuie profitat de o anume ordine și adiacență a celor trei.

Refaceți programul utilizând registrul IX în loc de HL, precum și modul de adresare indexat în loc de modul de adresare indirect prin HL. Calculați cât durează această versiune în raport cu cea anterioară.

3.4.2. CICLURI

Ponderea ciclurilor în structurile de program este la fel de mare în limbajul de asamblare, ca și în limbajele de nivel înalt. Un ciclu este o repetare a unei secvențe de instrucțiuni de n ori sau până la îndeplinirea unei condiții. Astfel, avem DO în Fortran, FOR în C, FOR, WHILE și REPEAT în Pascal. De cele mai multe ori, numărul de iterații (n) este cunoscut. În setul Z80 avem instrucțiunea DJNZ (Decrease and Jump if Not Zero, adică "scade cu 1 și salt dacă rezultatul nu este zero"). Această instrucțiune folosește implicit registrul B drept contor, deci permite maximum 256 iterații. Iată un exemplu de ciclu cu 7 iterații, în interiorul căruia se incrementează un registru:

```
LD DE,5
LD B,7          ; 7 iterații
loop INC DE
DJNZ loop
cont ...        ; DE va conține 5+7=12
```


Dacă B se încarcă inițial cu 0, atunci numărul de iterații va fi 256 (nu zero!), deoarece în instrucțiunea DJNZ mai întâi se decrementează valoarea lui B și apoi se verifică dacă a ajuns la 0.

Dacă în secvența de instrucțiuni din ciclu avem nevoie de registrul B, vom utiliza stiva, astfel:

```

loop    LD B,0           : 256 de iterații
        PUSH BC
        POP BC
        DJNZ loop

```

Saltul pe care îl execută DJNZ pentru a reveni la începutul ciclului este relativ (tip JR). În consecință, secvența de instrucțiuni din ciclu nu trebuie să fie mai lungă de 126 octeți. Pentru a scăpa de această condiție se folosește artificul următor:

```

loop    LD B,77
next    JP begin
        DJNZ loop
begin   ...              : corpul ciclului de peste 126 B
end     JP next

```

În practică se utilizează cicluri contorizate pe doi octeți, care nu mai prezintă inconveniente și merg până la 65536 iterații. Orice registru general poate fi folosit drept contor. Refacem unul din exemplele anterioare în această idee:

```

loop    LD DE,7
        LD BC,1000       : 1000 de iterații
        INC DE
        DEC BC
        LD A,B            : testarea unui registru dublu dacă este 0
        OR C
        JP NZ,loop

```

Se observă modalitatea testării unui registru dublu dacă este 0 sau nu prin operația SAU bit cu bit între cei doi octeți componenți. Numai dacă ambii sunt zero, *flag*-ul Z va fi poziționat pe 1, "setat", și saltul la "loop" nu se mai efectuează.

3.4.3. UTILIZAREA REGISTRELOR ALTERNATIVE

Registrele alternative A'F', B'C', D'E' și H'L' reprezintă o soluție pentru schimbări de context rapide, care nu reclamă recursivitate. Uneori, prezența registrelor alternative este interpretată ca o dublare a capacității de stocare în registre, ceea ce nu este tocmai adevărat. Deși pe producători i-a avantajat impresia creată că Z80 are de fapt două acumulatoare și 6 registre duble

generale, experiența practică nu a validat această idee. În primul rând, transferul direct între un registru și un registru alternativ este complicat, trebuind găsit un spațiu de stocare intermediar. În al doilea rând, registrele alternative nu au o identitate proprie. De fapt, nici un program nu poate determina la un moment dat în care din cele două seturi de registre lucrează. Singurul avantaj legat de registrele alternative este rapiditatea comutărilor cu instrucțiunile specializate:

```
EX AF,AF'      : comută AF cu A'F' în 4 T
EXX             : comută BC cu B'C', DE cu D'E' și HL cu H'L' în
                : 4 T
```

Pentru a contracara lipsa de identitate a registrelor alternative, toate programele care fac comutări trebuie să păstreze un riguros echilibru între instrucțiunile de comutare, indiferent pe ce ramură ar merge programul. Acest lucru impune programatorilor un efort suplimentar, care uneori nu se justifică. Varianta cu variabile în RAM este adesea preferabilă celei cu registrele alternative utilizate ca variabile.

De altfel, dacă intenția proiectanților microprocesorului Z80 ar fi fost aceea de a dubla efectiv numărul de registre de lucru, cele noi ar fi primit nume distincte, dar intenția lor nu a fost aceasta pentru că, față de celelalte microprocesoare din aceeași clasă (Intel 8080, Intel 8085 și Motorola 6800), Z80 are și așa prea multe registre, ceea ce înseamnă timp suplimentar consumat la decodarea instrucțiunilor și un număr mare de coduri de operație.

Schimbările de context însă reprezintă o cu totul altă problemă față de utilizarea registrelor alternative ca simple variabile. Pentru a înțelege diferența, vom porni de la faptul că instrucțiunea DJNZ (Decrease and Jump if Not Zero) folosește implicit registrul B drept contor. Dacă o subrutină MAIN conține un ciclu cu DJNZ în interiorul căruia avem un apel la o subrutină SBR, care de asemenea conține un ciclu cu DJNZ, apare un conflict asupra registrului B. Rezultatul programului următor, scris de un începător, este de-a dreptul dezastruos:

```
MAIN      LD A,0           : inițial A:=0
          LD B,5           : repetă de 5 ori:
loopx     CALL SBR         : A:=A+6
          DJNZ loopx       : B nu mai conține ce trebuie!
          RET              : în final ar trebui ca A să aibă valoarea
                          : 5×6=30

SBR       LD B,6           : repetă de 6 ori
loopy     INC A            : A:=A+1
          DJNZ loopy       :
          RET              : în final A:=A+6
```

La începutul subrutinei SBR ar trebui făcută o schimbare de context, pentru a putea utiliza registrul B din nou în subrutină. În versiune clasică, schimbarea de context se face prin stivă, astfel:

```
SBR       PUSH BC         : B din MAIN este salvat în stivă
          LD B,6
loopy     INC A
          DJNZ loopy
          POP BC           : este readus B din MAIN
```


RET : durata schimbării de context: 21 T

Schimbarea de context se poate realiza și cu registrele alternative, astfel:

SBR	EXX	: B din MAIN devine B'
	LD B,6	
loopy	INC A	
	DJNZ loopy	
	POP BC	: este readus B din MAIN
	RET	: durata schimbării de context: 8 T

Schimbarea de context folosind registrele alternative este de 3 până la 8 ori mai rapidă, în funcție de câte registre avem de salvat. În schimb, această metodă nu se poate aplica întotdeauna. De exemplu, dacă subrutina are nevoie de valorile registrelor din programul apelant, soluția PUSH-POP este ideală, pentru că PUSH HL nu schimbă conținutul registrului HL, pe când EXX o face. De asemenea, în cazul unei adâncimi mai mari a nivelului de subrutine, sau în cazul subrutinelor recursive, metoda PUSH-POP este singura practicabilă. Astfel, dacă subrutina SUBR din exemplul de mai sus ar chema la rândul ei în ciclu o altă subrutină care ar avea nevoie de registrul B, schimbarea de context cu registrele alternative devine imposibilă. Din păcate, majoritatea cazurilor practice nu îndeplinesc aceste condiții și necesită schimbări de context prin stivă.

Prezentăm în continuare o tehnică de transfer între registrele principale și cele alternative omonime, prin intermediul stivei:

PUSH HL	
EXX	
POP HL	
EXX	: secvența este echivalentă cu H'L':=HL și
	: durează 29 T
PUSH AF	
EX AF,AF'	
POP AF	
EX AF,AF'	: secvența este schivalentă cu A'F':=AF și
	: durează 29 T

Se observă că durata acestor transferuri este considerabilă.

3.4.4. SUBROUTINE CU PARAMETRI ÎN LINIE (*in-line*)

În limbajele de nivel înalt (C, Pascal, Fortran) se utilizează frecvent subrutine cu parametri. Acestea permit execuția aceluiași instrucțiuni cu diferite seturi de date, ceea ce este mai util decât execuția aceluiași instrucțiuni cu același set de date, ca în Basic și majoritatea limbajelor de asamblare. În limbajul de asamblare Z80 putem crea totuși subrutine cu parametri. Tehnica subrutinelor cu parametri în linie (*in-line*) permite transmiterea de date direct după apelul de subrutină, prin DEFB, DEFW sau DEFM, ca în exemplele de mai jos. Această tehnică a ajuns să fie foarte des utilizată, fiind principala beneficiară a instrucțiunii EX (SP).HL (schimbarea conținutului între registrul HL și vârful stivei).

Parametrii care se pot transmite sunt fie adrese ale variabilelor din memorie (transmiterea parametrilor prin adresă, ca în exemplul din §3.4.4.2), fie valori imediate (transmiterea parametrilor prin valoare, ca în exemplul din §3.4.4.1). Deosebirea esențială dintre transmiterea parametrilor prin adresă și cea prin valoare este că în primul caz subrutina poate modifica valorile unor parametri și poate întoarce prin urmare o informație. În cel de-al doilea caz, subrutina doar utilizează parametrii transmiși, fără a-i modifica și fără a-i returna în programul apelant. Distincția este familiară celor care cunosc limbajele C și Pascal, limbaje ce suportă ambele moduri de transmitere a parametrilor. Limbajul Fortran nu dispune decât de transmiterea prin adresă. Metoda expusă nu este aplicabilă în cazul în care parametrii sunt nume de registre.

Un caz particular de subrutine cu parametri în linie îl reprezintă cel al selecției după parametru (v. §3.4.4.3).

3.4.4.1. Transmiterea parametrilor prin valoare

În exemplul următor prezentăm subrutina WSTR care scrie pe ecran un șir de caractere transmis în linie, prin valoare. Se utilizează convenția specifică limbajului C de a termina șirurile de caractere cu caracterul cu codul 0.

	CALL WSTR	:	subrutina scrie pe ecran șirul de caractere
		:	dat
	DEFM "Spectrum"	:	în linie, într-o definiție de mesaj (DEFM).
	DEFB 0	:	încheiată cu 0 (marker de sfârșit de șir)
cont	...	:	din acest loc trebuie să se reia execuția la
		:	întoarcerea din subrutină
WSTR	EX (SP),HL	:	aduce în HL ultima adresă din stivă
loop	LD A,(HL)	:	se ia câte un caracter din șir
	INC HL	:	se pregătește adresa următorului
	AND A	:	se verifică dacă este 0 adică sfârșit de șir
	JR Z,rtn	:	dacă da, atunci salt în afara buclei
	RST #10	:	la Spectrum subrutina de scriere a unui caract-
		:	ter
	JR loop	:	repetă pentru caracterul următor
rtn	EX (SP),HL	:	se reface în stivă adresa corectă de întoarce-
		:	re (cont)
	RET	:	

Singura condiție pentru exploatarea acestei tehnici este ca înainte de întoarcerea din subrutină, adresa de întoarcere să fie corect calculată în HL și depusă în stivă cu EX (SP),HL. Menționăm că valoarea registrului HL din programul apelant nu este afectată în cursul executării subrutinei WSTR.

3.4.4.2. Transmiterea parametrilor prin adresă

Subrutina ADDINT din exemplul care urmează adună două variabile întregi reprezentate pe 1 octet. Adresele celor două variabile sunt date ca parametri în linie. Rezultatul este depus în prima variabilă. Programul principal va apela de

două ori consecutiv subrutina ADDINT, rezultatul final fiind suma a trei variabile, alfa, beta și gama, depusă în alfa.

```

CALL ADDINT      : alfa:=alfa+beta
DEFW alfa
DEFW beta
CALL ADDINT      : alfa:=alfa+gama
DEFW alfa
DEFW gama
cont            ...
alfa           DEFB 4
beta           DEFB 5
gama           DEFB 8
               ...

ADDINT         EX (SP),HL
               LD E,(HL)
               INC HL
               LD D,(HL)
               INC HL      : DE conține adresa operandului 1
               LD C,(HL)
               INC HL
               LD B,(HL)
               INC HL      : BC conține adresa operandului 2
               EX DE,HL    : acum HL conține adresa operandului 1
               LD A,(BC)   : A conține valoarea operandului 2
               ADD A,(HL)  : este adunată cu valoarea operandului 1
               LD (HL),A   : rezultatul este depus la adresa operandului 1
               EX DE,HL    : adresa pentru întoarcere revine mai întâi în
               : HL ...
               EX (SP),HL  : ... apoi în vârful stivei
               RET

```

3.4.4.3. Selecție după variabilă sau condiție

În acest caz, ca parametri în linie se dau mai multe adrese de salt posibile. Subrutina testează o variabilă sau o condiție de selecție și se "întoarce" printr-un salt la una din adresele date ca parametri. Echivalentul acestei tehnici de programare este instrucțiunea CASE din Pascal, switch din C, respectiv GO TO calculat din Fortran. De asemenea, instrucțiunea IF aritmetic din Fortran IV este tot o selecție și vom vedea în exemplul care urmează o modalitate de a o scrie în limbaj de asamblare Z80. Subrutina IFARTM are trei parametri în linie și anume trei adrese. Apelând IFARTM, dacă A<C, atunci se execută un salt la prima adresă (maimic), dacă A=C, atunci se execută un salt la a doua adresă (egal), iar dacă A>C, atunci se execută un salt la cea de-a treia adresă (maimare). A și C reprezintă conținutul registrelor respective. Această structură de program este chiar mai generală decât IF aritmetic din Fortran, cu care este echivalentă în cazul particular C=0. Exemplul de apelare cu A=12 și C=5 va avea ca efect salt la ultima adresă (maimare).

```

LD A,12
LD C,5
CALL IFARTM

```



```

DEFW maimic
DEFW egal
DEFW maimare
egal      ...
maimic    ...
maimare   ...
IFARTM    EX (SP),HL
           CP C
           JR NC,teste
           LD DE,0      ; pe ramura A<C adresa de întoarcere va fi la
                           ; (HL)
           JR cont
teste     JR NZ,aGTc
           LD DE,2      ; pe ramura A=C adresa de întoarcere va fi la
                           ; (HL+2)
aGTc      JR cont
           LD DE,4      ; pe ramura A>C adresa de întoarcere va fi la
                           ; (HL+4)
cont      ADD HL,DE     ; se calculează adresa la care vom găsi adresa
                           ; de întoarcere
           LD E,(HL)
           INC HL
           LD D,(HL)    ; DE conține adresa de întoarcere
           EX DE,HL
back      EX (SP),HL    ; adresa de întoarcere este depusă în vârful
                           ; stivei
           RET

```

Notă: Nu ar fi fost bine JP (HL) în loc de EX (SP),HL | RET deoarece stiva ar fi rămas încărcată cu o valoare inutilă și prin urmare ar fi fost dezechilibrată. Totuși, dacă am fi preluat valoarea din stivă printr-un POP, am fi putut face și JP (HL), de exemplu:

```

back      POP DE
           JP (HL)

```

Exerciții

a) Extindeți ideea subrutinei ADDINT din §3.4.4.2, construind o subrutină SIGMA cu parametri în linie, pentru a aduna mai multe numere întregi reprezentate pe 1 octet. Primul parametru în linie dat prin DEFB va fi numărul de operanzi ($2 \div 255$). Următorii parametri dați prin DEFW vor fi adresele operanzilor. Rezultatul va fi depus la adresa primului operand. Astfel, exemplul din §3.4.4.2 va putea fi rescris astfel:

```

CALL SIGMA
DEFB 3
DEFW alfa
DEFW beta
DEFW gama
cont     ...

```

În general, procedurile (subrutine sau funcții) cu număr variabil de parametri

poartă numele de variadice. Exemple celebre de proceduri variadice sunt `write` și `read` din Pascal și `printf` și `scanf` din C.

b) Scrieți o subrutină IFIN cu doi parametri în linie reprezentați de două caractere ASCII date prin DEFB. Subrutina trebuie să întoarcă CY=1 dacă acumulatorul conține un caracter ASCII cu codul cuprins între cele două date ca parametri, altfel trebuie să întoarcă CY=0. De exemplu, testarea dacă un caracter din A este cifră (între "0" și "9") se va face cu următoarea secvență:

```
LD A,5                ; dar dacă ar fi LD A,"5"?
CALL IFIN
DEFB "0"
DEFB "9"
JP C,cifra
altceva ...
cifra ...
```

3.4.4.4. Tehnica *hook-code*

O aplicație a tehnicii transmiterii parametrilor în linie o reprezintă și tehnica *hook-code*. În cazul unei subrutine complexe, care execută mai multe servicii, se poate transmite în linie un parametru de 1 octet, *hook-code*-ul, codul de agățare, care selectează serviciul cerut. Să analizăm un exemplu de subrutină care execută trei servicii diferite, constând din scrierea unor mesaje de eroare. Parametrul transmis în linie identifică serviciul cerut. În subrutina mesaj, selecția se face pe baza unei tabele de salturi (v. §3.4.6).

```
CALL mesaj
DEFB 2
...
mesaj EX (SP),HL
      LD A,(HL)      ; în A avem serviciul cerut
      INC HL
      EX DE,HL       ; de registrul HL vom avea nevoie
      CP 3
      JP NC,err      ; serviciile disponibile sunt 0, 1 sau 2
      LD HL,tabela   ; selecția serviciilor prin tabelă de salturi (v.
                      ; §3.4.6)
      ADD A,A        ; intrările în tabelă sunt la fiecare 2B
      LD C,A         ; adunarea la adresa de început a tabelii
      LD B,0
      ADD HL,BC      ; în HL avem adresa adresei unde se face saltul
      LD C,(HL)      ; LD BC,(HL) (v. 3.4.1)
      INC HL
      LD B,(HL)
      INC HL         ; în BC avem adresa unde se face saltul
      LD HL,m_err
      CALL WSTR      ; (v. 3.4.4)
      DEFM 'EROARE:
      DEFB 0
      LD L,C
```



```

LD H,B
JP (HL)      ; în 12 T. față de PUSH BC | RET în 21 T
tabela      DEFW serv0      ; tabela de salturi este o succesiune de cuvinte
                                ; de 2B
                                ; reprezentând adresele serviciilor
serv0        DEFW serv1
              DEFW serv2
              CALL WSTR
              DEFM 'sintaxa incorecta.'
              DEFB 0
serv1        JR back
              CALL WSTR
              DEFM 'depasire.'
              DEFB 0
              JR back
serv2        CALL WSTR
              DEFM 'identificator definit a doua oara.'
              DEFB 0
              JR back
back         OR A            ; CY=0 pentru întoarcerea fără eroare
err          EX DE,HL        ; aici ajunge cu CY=1, întoarcere cu eroare
              EX (SP),HL
              RET

```

Execuția secvenței de mai sus provoacă scrierea pe ecran a următorului mesaj:

EROARE: identificator definit a doua oara.

Această metodă este mai complicată decât simpla apelare directă a serviciului respectiv ca subrutină (în cazul nostru, CALL serv2). În schimb se rezolvă două probleme foarte spinoase. În primul rând, serviciile pot avea părți comune (în exemplul de mai sus, scrierea începutului de mesaj: 'EROARE: '). În al doilea rând, reasamblarea subrutinei mesaj (cu modificarea adreselor serviciilor) se poate face fără a fi necesară reasamblarea întregului program principal. Acest fapt facilitează scrierea unor aplicații complexe de mai mulți programatori, care lucrează în paralel. Chiar și pentru un singur programator este mai comod să împartă aplicația în module și să asambleze numai câte un modul, în loc să reasambleze de fiecare dată întreg programul. De asemenea, structurarea pe subrutine și servicii este un mod de dezvoltare mai confortabil decât structurarea numai pe subrutine, întrucât se grupează mai multe operații înrudite în același apel de subrutină.

La calculatoarele *Spectrum* se practică tehnica *hook-code* pentru subrutina de tratare a erorilor (RST 8) și pentru apelarea subrutinelor din ROM-ul ascuns al Interfeței 1.

3.4.5. ÎNMULȚIREA RAPIDĂ A NUMERELOR ÎNTREGI

Microprocesorul Z80, ca și altele din clasa sa (Intel 8080, Intel 8085, Motorola 6800), nu are instrucțiuni directe de înmulțire. În schimb, există instrucțiuni de deplasare la stânga (SLA, RL), ceea ce echivalează cu o înmulțire cu 2 a unui întreg fără semn. Două deplasări succesive vor echivala cu înmulțirea cu 4. Se vede ușor că înmulțirile cu 2^k se traduc prin k deplasări succesive la stânga ale variabilei care conține deînmulțitul.

Pentru variabile întregi fără semn, pe 1 octet ($0 \div 255$), este suficientă repetarea instrucțiunii SLA, ca în exemplul următor:

```
LD HL,var      : var conține inițial 5
SLA (HL)       : var conține  $5 \cdot 2 = 10$ 
SLA (HL)       : var conține  $5 \cdot 4 = 20$ 

var            DĖFB 5
```

Pentru variabile întregi fără semn, reprezentate pe 2 octeți ($0 \div 65535$), se folosește *flag*-ul CY pentru transport, și se combină instrucțiunea SLA cu RL. Următorul exemplu tratează o înmulțire cu 4 a unei variabile conținute în registrul DE:

```
LD DE,500      : DE conține inițial 500
SLA E          : E se înmulțește cu 2
RL D           : D se înmulțește cu 2 și se adună transportul
               : CY
SLA E          : se repetă secvența
RL D           : în final DE va conține  $500 \cdot 4 = 2000$ 
```

Dacă variabila este alocată în memorie, atunci secvența devine:

```
LD HL,var      : var conține inițial 500
SLA (HL)
INC HL
RL (HL)         : var conține  $500 \cdot 2 = 1000$ 
DEC HL
SLA (HL)
INC HL
RL (HL)         : var conține  $500 \cdot 4 = 2000$ 

var            DĖFW 500
```

Exemplul de mai sus se constituie și într-un răspuns la întrebarea pe care și-o pun mulți programatori începători: de ce instrucțiunile INC rr și DEC rr nu modifică *flag*-urile? Pentru că de obicei INC rr și DEC rr trebuie intercalate între instrucțiuni care au nevoie să-și comunice valorile acestor *flag*-uri.

În cazul particular al variabilelor pe 1 octet stocate în acumulator, sau al variabilelor pe 2 octeți stocate în registrul HL, vom utiliza în loc de SLA, respectiv SLA-RL, instrucțiunile directe de adunare ADD, ca în exemplele următoare:

```
LD HL,333
SLA L
RL H           : înmulțirea cu 2 în 16 T
ADD HL,HL      : înmulțirea cu 2 în 11 T

LD A,33
SLA A          : înmulțirea cu 2 în 8 T
ADD A,A        : înmulțirea cu 2 în 4 T
```

Se observă, în acest caz particular, câștigul de viteză obținut prin folosirea instrucțiunii ADD. Din păcate, nu întotdeauna avem A sau HL disponibile.

Înmulțirea rapidă cu numere care nu sunt puteri ale lui 2 se face practic printr-o combinație de înmulțiri cu puteri ale lui 2 și adunări. Exemplele următoare ilustrează înmulțirea rapidă pe 2 octeți, cu diverse numere, a unei variabile conținute în registrul HL.

```
LD HL,13          : înmulțirea rapidă cu 3
LD D,H
LD E,L            : DE:=13
ADD HL,HL         : HL:=HL*2 (13*2=26)
ADD HL,DE         : HL:=HL+DE (26+13=39)
```

```
LD HL,13          : înmulțirea rapidă cu 10
ADD HL,HL         : HL:=HL*2 (13*2=26)
LD D,H
LD E,L            : DE:=26
ADD HL,HL         : HL:=HL*2 (26*2=52)
ADD HL,HL         : HL:=HL*2 (52*2=104)
ADD HL,DE         : HL:=HL+DE (104+26=130)
```

Înmulțirile rapide au multe aplicații practice, dintre care am ales pentru exemplificare subrutina READ10. Adesea este necesară conversia unui număr întreg reprezentat ca șir de caractere ASCII, în reprezentarea sa binară. De exemplu, numărul 183 trebuie convertit de la reprezentarea:

```
"1"    001100012
"8"    001110002
"3"    001100112
```

la reprezentarea:

```
18310 00000000 101101112
```

Vom avea în vedere cazul numerelor întregi fără semn, reprezentabile pe 2 octeți (între 0 și 65535). Se observă că între codul ASCII al unei cifre și reprezentarea sa binară există un decalaj constant și anume 48. Deci dacă vom scădea codul cifrei "0" din codul cifrei respective, vom obține reprezentarea binară a cifrei (de exemplu: "1" - "0" = 00110001₂ - 00110000₂ = 00000001₂). Rutina de conversie va trebui să însumeze într-o variabilă întreagă pe 2 octeți fiecare cifră în reprezentarea binară cu rezultatul precedent înmulțit cu 10. Variabila va fi inițializată la 0. De exemplu, în cazul nostru:

```
VAR := 0
VAR := VAR * 10 + 1 rezultat: 1
VAR := VAR * 10 + 8 rezultat: 18
VAR := VAR * 10 + 3 rezultat: 183
```

La intrarea în subrutina READ10, HL va indica spre începutul șirului de cifre ASCII. Sfârșitul șirului este marcat de orice caracter nenumeric (inclusiv codul 0, standard C).

```
READ10  EX DE,HL
        LD HL,0          : inițializează numărul la 0
```


loop	LD A,(DE)	
	SUB "0"	: transformă cifra din ASCII în reprezentarea ei
		: binară
	JP M,rtn	: dacă are codul mai mic decât o cifră, atunci
		: s-a terminat
	CP 10	
	JP NC,rtn	: dacă are codul mai mare decât o cifră, de
		: asemenea
	ADD HL,HL	
	RET C	: toate instrucțiunile RET C care urmează produc
		: întoarcerea în cazul erorii de depășire (over-
		: flow): depășirea poate surveni la oricare din
		: adunări
	LD B,H	
	LD C,L	
	ADD HL,HL	
	RET C	
	ADD HL,HL	
	RET C	
	ADD HL,BC	: conținutul inițial al lui HL se înmulțește cu
		: 10
	RET C	
	LD B,0	
	LD C,A	: se convertește cifra pe 16 biți
	ADD HL,BC	: se adună cifra la HL
	RET C	
	INC DE	
	JR loop	: următoarea cifră
rtn	OR A	: înainte de întoarcere se resetează Carry
		: pentru a
	RET	: semnaliza "întoarcere fără eroare"

3.4.6. SELECTII PRIN TABELE DE SALTURI

O aplicație a tabelelor de salturi a fost prezentată deja în §3.4.4.4. Aceste tabele permit selectarea unui anumite operații dintr-un grup de operații înrudite, numite servicii. De asemenea, fac posibilă reasamblarea doar a unor anumite module dintr-un program mare. În plus, tabelele de salturi facilitează depanarea programelor în sensul că anumite servicii pot fi ușor redirectionate în momentul rulării (fără reasamblare), schimbând numai valori în tabelă. În sfârșit, este o centralizare utilă a tuturor punctelor de intrare într-un modul mai mare, altfel greu de gestionat.

Pentru practicile curente de *reversed engineering* în software, adică de dezasamblare și studiere a programelor executabile, identificarea tabelelor de salturi (dacă există) este prima operație care trebuie întreprinsă. Aproape toate programele executabile mai complexe conțin unul sau mai multe astfel de tabele.

Această tehnică este ilustrată în exemplul din §3.4.4.4.

3.4.7. TEHNICA DECUPĂRII DE BIȚI

În limbajul de asamblare Z80, tipurile de date curente sunt BYTE (1B), octetul, și WORD (2B), cuvântul. Uneori însă se lucrează la nivel de bit, de exemplu în grafică, unde un pixel monocrom corespunde unui bit, sau pentru

reprezentarea economică a unui vector de variabile logice. În aceste situații sunt necesare operații cu biți independenți. Instrucțiunile specializate SET și RES permit poziționarea pe "1", respectiv pe "0" a unui anumit bit din registrul specificat, iar instrucțiunea BIT permite testarea valorii unui anumit bit. Totuși, dacă bitul vizat se găsește în acumulator, este preferabilă utilizarea instrucțiunilor OR și AND cu o "mască" adecvată, ca în exemplele următoare:

```

SET 2,A      : pune pe 1 bitul D2 din acumulator în 8T
OR 2         : aceeași operație în 7T: 2 = 000000102

RES 7,A      : pune pe 0 bitul D7 din acumulator în 8T
AND #7F      : aceeași operație în 7T: #7F = 011111112

BIT 5,A      : testează bitul D5 din acumulator în 8T
              : acumulatorul rămâne intact
AND #20      : aceeași operație în 7T, dar A se distruge
              : #20 = 001000002

```

Uneori trebuie complementat un anumit bit, ceea ce nu se poate face eficient decât cu instrucțiunea XOR cu mască, astfel:

```

XOR #10      : complementează bitul D4 din acumulator: #10 = 000100002

```

Decuparea de biți se referă la formarea unui octet cu biți din două registre diferite, după o anumită mască. Unul din registrele operand va fi acumulatorul. După decupare, el va conține și rezultatul. Tehnica clasică a decupării este următoarea:

```

XOR r        : 
AND masca    : masca va conține biți de 0 numai pe pozițiile
              : decupate
XOR r        : 

```

De exemplu, să presupunem că trebuie să decupăm biții D3, D4 și D5 din registrul H și să-i combinăm cu biții D0, D1, D2, D6 și D7 din acumulator. Mască va fi formată din biți de "0" în pozițiile 3÷5 și în rest din biți de "1", adică 11000111₂ = #C7.

```

LD H,#AA      : 101010102
LD A,#1B      : 000110112
XOR H         : 
AND #C7       : 110001112
XOR H         : rezultatul în A: 001010112

```

Această tehnică este exploatată cu predilecție la rutinele grafice.

APLICAȚII ALE COMPATIBILELOR *SPECTRUM*

4.1. INTERFAȚĂ DE IMPRIMANTĂ

Interfața pe care o prezentăm în continuare poartă numele de *Tuttiprint* și permite accesul calculatoarelor compatibile *Spectrum* la orice imprimantă paralelă (*Centronics*) sau serială (EIA RS232). De asemenea, poate fi utilizată și la comunicații unidirecționale cu alte periferice, precum și pentru achiziția de date, putând fi configurată ca interfață cu 9 intrări digitale.

În continuare vom descrie partea de software a interfeței. Programul este compatibil cu Interfața 1 și *Microdrive*, precum și cu *TimExt*.

În modul grafic, programul poate adresa trei categorii de imprimante:

- RCD 9335 Scamp și compatibile;
- Epson FX-80, Star LC-10, IBM Proprinter, Robotron K6313;
- Robotron K6311, Romom.

În modul text, programul lucrează cu orice imprimantă paralelă cu interfață de tipul *Centronics* sau serială conform specificațiilor EIA RS232 și CCITT V-24.

Programul *Tuttiprint* este compatibil și co-rezident cu următoarele programe: Hisoft Devpac (Gens și Mons), Hisoft Pascal, Hisoft C. Utilitarele și *overlay*-urile asociate sunt compatibile și co-rezidente cu următoarele programe: Masterfile, Beta Basic 3.1, Tasword Three, Laser Genius. De asemenea, programul recunoaște și tipărește următoarele tipuri de fișiere: Tasword Two, Masterfile, The Writer, Spectral Writer, fișiere ecran (SCREEN\$).

4.1.1. DEFINIREA TERMENILOR REFERITORI LA IMPRIMANTĂ

Dăm în continuare o listă cu termenii cei mai des întrebuințați pentru definirea funcționării și interfațării unei imprimante matriceale.

ASCII (*American Standard Code for Information Interchange*) – standard internațional (ISO-7) de reprezentare internă în echipamentele de calcul a caracterelor, pe 7 biți.

Baud — unitate de măsură reprezentând o tranziție pe linia de comunicații într-o secundă. Se utilizează pentru sincronizarea comunicației seriale de date.
Caractere tipăribile — caractere care pot fi tipărite la imprimantă (litere, cifre, simboluri).

Caractere netipăribile — caractere care nu pot fi tipărite la imprimantă; pot fi caractere de control sau caractere grafice.

Caractere de control — caractere netipăribile prin care se controlează poziția cursorului la tipărire, culoarea, densitatea, forma literelor ș.a.m.d.

Caractere grafice — caractere nestandard, a căror tipărire la imprimantă nu se poate face în modul text; de exemplu blocurile semigrafice, *UDG*-urile, simbolul © (copyright).

Carriage Return (CR) — caracter de control pentru întoarcerea capului de tipărire la începutul liniei; prin analogie cu mașina de scris: retur de car.

CCITT — Comitetul Consultativ Internațional pentru Telefonie și Telegrafie.

Comunicație serială — mod de comunicație digitală în care biții sunt transmiși succesiv, pe o singură linie de date.

Comunicație paralelă — mod de comunicație digitală în care biții care compun un caracter sunt transmiși simultan, pe mai multe linii de date.

Configurare software a imprimantei — transmiterea de parametri unei imprimante de către calculatorul gazdă, prin caractere de control.

Configurare hardware a imprimantei — transmiterea de parametri unei imprimante de către operator, prin acționarea unor microcomutatoare.

Driver — subprogram de cel mai coborât nivel prin apelul căruia se realizează programarea și exploatarea unui anumit periferic.

DTR (Data Terminal Ready) — linie de protocol hardware în comunicația serială după *EIA RS232*, prin starea căreia receptorul (imprimanta) semnalizează dacă este sau nu pregătit pentru preluarea de noi caractere.

EIA (Electronic Industries Association) — organizație din Statele Unite care elaborează standarde în comunicații de date.

Expandarea token-urilor — înlocuirea unui *token* cu succesiunea corespunzătoare de litere și simboluri.

Form feed (FF) — caracter de control pentru avansul hârtiei la o pagină nouă.

Formatter — subprogram de nivel mai înalt, care realizează organizarea datelor într-un anumit format.

Formatter text — *formatter* care prelucrează datele de tip text.

Formatter grafic — *formatter* care prelucrează informația grafică.

Header — titlul unui document, a cărui tipărire pe prima linie a fiecărei pagini, asigurată de *formatter*, permite identificarea documentului tipărit.

Line feed (LF) — caracter de control pentru avansul hârtiei la o linie nouă.

Overlay — program co-rezident care extinde, modifică sau adaptează funcțiile programului de bază.

Parallel Input Output (PIO) — circuit din familia microprocesorului *Z80*, specializat pentru comunicația paralelă de date.

Program de generare — program interactiv utilizat pentru configurarea unui alt program și corelarea acestuia cu mediul hardware în care va rula.

Protocol — convenție respectată atât de emițător, cât și de receptor, având drept scop principal posibilitatea receptorului de a cere emițătorului oprirea sau repornirea fluxului de date.

RS232 — standardul EIA de comunicație serială de date.

STROBE — semnal de sincronizare a comunicației paralele.

Token — element lexical al unui limbaj de programare (cuvânt-cheie), aflat pe același nivel structural cu literele, cifrele și simbolurile, având însă forma unei succesiuni de litere și simboluri. De exemplu, cuvintele cheie din Basic: PRINT, CONTINUE etc.

TxD (*Transmit Data*) — linia de ieșire serială de date, conform EIA RS232.

V24 — recomandările CCITT pentru comunicații seriale de date.

Viteza de transmisie — parametru al comunicației de date, exprimat la nivelul fizic în *bauds* (biți pe secundă). La comunicația serială asincronă, emițătorul și receptorul trebuie configurate separat pe aceeași viteză de transmisie.

4.1.2. PROGRAMUL INTERFETEI

Programul *Tuttiprint* este un instrument software adaptabil într-o diversitate de medii hardware și software, destinat exploatării imprimantei. Utilizarea acestui program asigură generalitatea aplicațiilor în raport cu tipărirea la imprimantă, precum și un standard destul de ridicat pentru calitatea informației tipărite (ca mod de organizare). Strategia de proiectare a urmărit să reunifice pe de o parte diversele posibilități de tipărire la imprimantele uzuale, pe de altă parte diversele interfețe existente la mai multe calculatoare compatibile *Spectrum* (Tim-S, HC-85, Cobra ș.a.). Pe un alt plan, s-a urmărit compatibilitatea cu software-ul de bază care rulează pe *Spectrum*. Ca urmare, orice aplicație software care este proiectată cu ieșirea la imprimantă prin *Tuttiprint* este portabilă pe o multitudine de configurații hardware. Această concepție depășește faza de pionierat, în care fiecare context utilizator-calculator-interfață-imprimantă-driver-aplicație era unic. Prin *Tuttiprint*, programele utilizatorilor câștigă în generalitate, ceea ce le apropie de principalul lor obiectiv.

Tuttiprint se compune dintr-un segment principal în cod mașină, rezident în RAM la adresă fixă (64000÷65368), un program de generare interactiv în Basic și o serie de extensii sub formă de *overlay*-uri, în cod mașină sau în Basic. Segmentul principal conține funcțiile de bază: tipărirea unui caracter (*driver*), *formatter* de text, *formatter* grafic, tipărirea unui fișier text (Tasword Two). Funcțiile și facilitățile avansate, precum și adaptabilitatea la programele care ocupă zona de RAM de la 64000 se realizează cu ajutorul extensiilor. Caracteristicile segmentului principal sunt următoarele:

- *driver* serial cu viteza programabilă de 50, 110, 300, 600, 1200, 2400, 4800 bauds, protocol DTR;

- *driver* paralel cu protocol BUSY;

- *formatter* de text adresat prin fluxul de date #3 (funcționează LPRINT și LLIST din Basic și RST #10 din cod mașină); are opțiuni pentru *header* și numerotarea paginilor; formatul paginii este programabil; are opțiunea pentru imprimare pe coli de scris A4, cu așteptarea unei taste după fiecare pagină; are opțiunea pentru imprimarea pe ambele fețe ale foi (cu margine de îndosariere alternantă); lucrează în două moduri: "T", text (normal) și "B", fără formatare (pentru caractere de control); expandează *token*-urile Basic; interpretează TAB, AT (relativ la poziția curentă); interpretează virgula din PRINT ca salt la începutul următoarei zone de 16 câmpuri; ignoră caracterele de control cu excepția CR, pe care îl transmite CR + LF, și a FF (CHR 12); transmite UDG-urile, semnul © și caracterele semigrafice ca spațiu sau alt caracter la alegere; ignoră atributele; recunoaște CHR 0 pentru reinițializarea *formatter*-ului (începutul primei pagini) și schimbarea *header*-ului; recunoaște CHR 1 pentru schimbarea *header*-ului; permite configurarea software programabilă a imprimantei la fiecare reinițializare (de exemplu, schimbarea setului de caractere, schimbarea densității); listează fișiere Tasword la imprimantă, fără încărcarea Tasword-ului; permite listarea înlănțuită de fișiere Tasword pentru redactarea de lucrări (până la 9.999 de pagini);

- *formatter* grafic; produce *hardcopy* grafic al întregului ecran 256 x 192 pixeli (inclusiv cele 2 linii inferioare), în 3 variante dimensionale: X1 (90 x 68 mm), X2 (180 x 136 mm), X3 (270 x 204 mm, numai pe A3); este scris în cod mașină automodificabil; adresează la alegere una din imprimantele grafice menționate la începutul paragrafului.

4.1.2.1. Încărcarea și activarea programului

Programul TuttiPrint V4.7 este un cod salvat sub numele "tpc", care se încarcă cu:

```
CLEAR 63999: LOAD "tpc"CODE <ENTER>
```

El ocupă memoria RAM de la adresa 64000 (deasupra RAM_TOP-ului), până la 65386 (începutul zonei de UDG-uri). După încărcare, el se activează cu:

```
RANDOMIZE USR 64000 <ENTER>
```

Dacă imprimanta este conectată și alimentată, ea va răspunde prin avansarea cu o linie (LF). Dacă la activare calculatorul se blochează, înseamnă că imprimanta nu este ON LINE (vezi manualul imprimantei).

În acest stadiu se pot încerca LPRINT și LLIST din Basic, sau RST #10 pe fluxul #3 din cod mașină.

Din nefericire, datorită recursivității sale, programul TuttiPrint nu poate fi întrerupt cu BREAK sau cu altceva. Întreruperea programului ar fi creat posibilitatea acumulării de balast în stiva mașinii și neplăcerile ar fi fost mai mari. Pentru a evita consumul inutil de hârtie la o comandă greșită de listare, se deconectează imprimanta, lăsând programul să termine listarea în gol (eventual se aplică interfeței un fals semnal READY sau NON-BUSY).

Înainte de a începe imprimarea, se așază de fiecare dată hârtia în poziția de pagină nouă (capul imprimantei în partea de sus a paginii) și se inițializează *formatter*-ul de text cu comanda:

```
LPRINT CHR$ 0 <ENTER>.
```

4.1.2.2. *Formatter*-ul de text

În funcție de tipul hârtiei folosite și de așezarea dorită în pagină, se pot schimba următorii parametri, ale căror valori inițiale au fost alese la generare: NR_SPA = 64012. Numărul de spații trimise înaintea fiecărei linii, pentru margine de îndosariere la stânga; ($> = 1$);

NR_LIN = 64013. Numărul de linii pe pagină;

NR_COL = 64014. Numărul de caractere pe linie (exclusiv spațiile trimise înaintea liniei);

NR_SPX = 64015. Numărul de spații trimise înaintea fiecărei linii, pentru margine de îndosariere la dreapta, la imprimarea foilor pe verso; ($> = 1$);

OPTN = 64017. Următoarele opțiuni de 1 bit:

bitul 0 (+1) fișiere Tasword segmentate; (0) fișiere Tasword independente;

bitul 1 (+2) cu *header* și număr de pagină; (0) fără *header* și număr de pagină;

bitul 2 (+4) așteptarea unei taste după fiecare pagină (pentru coli A4); (0) fără așteptare (pentru hârtie continuă);

bitul 3 (+8) margine de îndosariere alternantă pentru coli A4 pe ambele fețe; (0) margine normală pentru coli A4 pe o singură față;

Valoarea care se introduce la OPTN reprezintă suma numerelor din parantezele care corespund opțiunilor respective. De exemplu, pentru fișiere Tasword segmentate, cu *header* și număr de pagină, pentru hârtie continuă, se face POKE 64017,1+2.

Parametrii NR_SPA, NR_LIN, NR_COL și NR_SPX se calculează în funcție de formatul hârtiei (lungimea H, lățimea L), suprafața activă dorită (lungimea h, lățimea l), distanța dorită între marginea stângă a hârtiei și începutul suprafeței active (marginea de îndosariere d) și densitățile de tipărire CPI (numărul de caractere pe inch) și LPI (numărul de linii pe inch). Densitățile standard ale majorității imprimantelor sunt CPI=10 și LPI=6, putând fi configurate și la alte valori, ca: CPI=12.5, CPI=16, LPI=8. Având dimensiunile H, L, h, l și d în milimetri, putem calcula parametrii cu următoarele relații (rotunjind la întreg):

$$NR_SPA = \frac{d \cdot CPI}{25,4} \quad NR_COL = \frac{l \cdot CPI}{25,4}$$

$$NR_LIN = \frac{h \cdot LPI}{25,4}$$

$$NR_SPX = \frac{(L - l - d) \cdot CPI}{25,4}$$

Lăţimea h rtiei (L) se m soar   ntre pozi ia cea mai din st nga a capului de imprimare  i pozi ia cea mai din dreapta a capului.  n exemplul care urmeaz  se consider  cazul cel mai comun, al colii de scris A4, cu H=297 mm  i L=205 mm. Pentru o suprafa  activ  h=245 mm, l=183 mm  i d=20 mm, cu densit ile standard CPI=10, LPI=6, se ob in: NR_SPA=8, NR_COL=72, NR_LIN=58, NR_SPX=1.

Modificarea parametrilor NR_SPA, NR_COL, NR_LIN  i NR_SPX devine efectiv  numai dup  ini IALIZAREA cu LPRINT CHR\$ 0 sau cu RANDOMIZE USR 64000.

Un efect interesant  l are schimbarea num rului de spa ii NR_SPA  i anume scrierea la dou  r nduri (sau la mai multe). Scrierea la dou  r nduri, util  de exemplu pentru corectur , se poate ob ine ad ug nd la NR_SPA, num rul maxim de caractere care pot fi tip rite pe o linie de imprimant . De exemplu, la Romom,  n densitatea standard se pot tip ri 80 de caractere pe linie. Dac   n loc de 8, NR_SPA va con ine 88, atunci scrierea se va face la dou  r nduri, iar dac  va con ine 168, scrierea se va face la trei r nduri.

Dac  OPTN bitul 2 este 1, atunci dup  fiecare pagin  se a teapt  schimbarea h rtiei  i ap sarea oric rei taste. Se recomand  utilizarea  n acest scop a tastelor CAPS SHIFT  i SYMBOL SHIFT, pentru a nu interveni accidental  n rularea anumitor programe, de exemplu Gens Assembler. Dac  OPTN bitul 1 este 1, atunci fiecare pagin  va avea pe prima linie un *header* (titlu sau antet) aliniat la st nga  i num rul de pagin ,  n formatul PAG NN, aliniat la dreapta. La fiecare ini IALIZARE, num rul de pagin  este adus la 1. Dac  se dore te  nceperea tip ririi cu pagina N>1, atunci se ini ializeaz   i apoi se dau comenzile urm toare:

```
INPUT n: LET o=1000: FOR i=64043 TO 64046: LET q=INT (n/o):
POKE i,q+48: LET n=n-o*q: LET o=o/10: NEXT i <ENTER>.
```

De fapt, programul de mai sus scrie  n loca iile de la 64043 la 64046, codurile ASCII ale cifrelor miilor, sutelor, zecilor  i respectiv unit ţilor num rului de pagin , N. Citirea num rului curent de pagin  se face astfel:

```
FOR i=64043 TO 64046: PRINT CHR$ PEEK i:: NEXT i: PRINT <ENTER>.
```

Dac  tip rirea se face  n op iunea cu margine alternant , pe ambele fe e ale colii de h rtie, atunci paginile impare vor avea marginea la st nga NR_SPA, iar paginile pare, NR_SPX. Este evident c  dac  se  ncepe tip rirea de la un NN

par, atunci rolurile parametrilor NR_SPA și NR_SPX se inversează, ca în exemplul următor, în care tipărirea trebuie să înceapă de la pagina 10 (de pe verso):

```
LET NRSPA=PEEK 64012: POKE 64012,PEEK 64015:
POKE 64015,NRSPA: LPRINT CHR$ 0:
POKE 64045, CODE "1": POKE 64046, CODE "0"<ENTER>.
```

Header-ul reprezintă titlul lucrării listate, iar modificarea lui se face cu comanda:

```
LPRINT CHR$ 1:"Un nou header" <ENTER>.
```

sau direct la inițializare:

```
LPRINT CHR$ 0:"Alt nou header" <ENTER>.
```

Dacă după CHR\$ 0 urmează direct <ENTER>, se păstrează *header*-ul existent. Pentru ștergerea *header*-ului este suficientă modificarea sa într-un spațiu, cu comanda: LPRINT CHR\$ 1:" ".

Lungimea *header*-ului este de maximum 64 de caractere, fiind condiționată și de NR_COL (< NR_COL-8). Depășirea lungimii de 64 poate altera zona de UDG-uri. Depășirea numărului NR_COL-8 are ca efect scrierea *header*-ului pe două sau mai multe linii, în loc de una.

Dacă se folosește comunicația serială, viteza de transmisie se poate modifica la BAUD=64019, după tabelul 4.1. Pentru viteza de transmisie de 1200 bauds se dă comanda POKE 64019,4 <ENTER>.

TABELUL 4.1. Selectarea vitezei de transmisie

Viteza (bauds)	50	110	300	600	1200	2400	4800
BAUD	0	1	2	3	4	5	6

Pentru configurarea software a imprimantei, se specifică mesajul de configurare, cu lungimea maximă de 23 octeți, începând cu adresa 64020. Ultimii 2 octeți din mesaj vor fi obligatoriu următorii: 13 și 138 (adică 10+128, 128 marcând sfârșitul mesajului). De exemplu, la imprimanta Romom dorim să tipărim cu setul de caractere internațional, în densitate maximă CPI=15.5. Trebuie date următoarele comenzi:

```
POKE 64020,27: POKE 64021,82:
POKE 64022,48: REM set int <ENTER>
POKE 64023,27: POKE 64024,91: POKE 64025,51: POKE 64026,32:
POKE 64027,75: REM 120 chr / lin (CPI=15.5) <ENTER>
POKE 64028,13: POKE 64029,138: REM terminatia <ENTER>.
```

Ștergerea mesajului de configurare se face automat, prin introducerea unui nou mesaj. Simpla anulare a mesajului se obține prin:

Tabelul 4.2 arată modul cum *formatter*-ul de text interpretează caracterele din setul ASCII *Spectrum*:

TABELUL 4.2. Interpretarea caracterelor ASCII *Spectrum* de către *formatter*-ul de text

Zecimal	Hexa	Interpre- tare	Efectul asu- pra variabi- lelor	Observații
0	00		LIN=0 COL=0 PAG=1	caracter de inițializare a <i>driver</i> -ului, la început de document nou; linia care urmează este considerată <i>header</i>
1	01			caracter de control al <i>driver</i> -ului: linia care urmează este considerată <i>header</i>
2÷5	02÷05			sunt filtrate, nu au nici un efect
6	06	N spații	COL=COL+N	virgula din PRINT avansează pe ori- zontală la începutul următoarei zone de 16 coloane
7÷11	07÷0B			sunt filtrate, nu au nici un efect
12	0C	CHR 12 (FF)	LIN=0 COL=0 PAG=PAG+1	salt la pagină nouă; început de pagină
13	0D	CHR 13 CHR 10 (CR)	COL=0	salt la linie nouă; început de linie
14÷15	0E÷0F			sunt filtrate, nu au nici un efect
16÷21	10÷15			ignoră următorul caracter (atribut)
22	16	1 x CR TAB c (AT)	LIN=LIN+1 COL=c	următoarele două caractere sunt luate drept 1 și c
23	17	c-COL spații (TAB)	COL=c	următorul caracter este luat drept c, apoi este ignorat un caracter
24÷31	18÷1F			sunt filtrate, nu au nici un efect
32÷126	20÷7E	CHR x	COL=COL+1	caractere tipăribile
127÷164	7F÷A4	1 spațiu	COL=COL+1	caractere grafice
165÷255	A5÷FF	N litere	COL=COL+N	<i>token</i> -urile Basic sunt expandate în secvențe de N litere sau simboluri

Variabilele LIN și COL sunt contoarele de linie și coloană ale *formatter*-ului text și se găsesc la următoarele adrese:

LIN = 64640, COL = 64641

Se observă din tabel că *formatter*-ul de text ignoră caracterele de control pentru attribute și înțelege virgula din PRINT ca salt pe orizontală la începutul următoarei zone de 16 coloane. De asemenea, înțelege TAB c ca salt pe orizontală la coloana c și, în sfârșit, înțelege AT l,c ca avans cu l linii și apoi TAB c. De observat faptul că, dacă pe ecran nu are sens TAB c, $c > 32$, *formatter*-ul interpretează orice $c < \text{NR_COL}$. Tabularea se comportă normal, adică dacă $c = \text{COL}$ atunci nu are efect, iar dacă $c < \text{COL}$, atunci se execută pe linia următoare (COL este coloana curentă).

Unele imprimante au funcții speciale de tabulare pe orizontală și pe verticală. Am evitat folosirea acestor funcții pentru respectarea principiului portabilității. AT (relativ la linia curentă) și TAB sunt interpretate de *formatter* și trimise ca spații, respectiv CR+LF. De asemenea, dintre caracterele de control este lăsat să treacă doar CHR 12 (FORM FEED), care se presupune comun tuturor imprimantelor. Pentru înlocuirea caracterelor grafice s-a utilizat spațiul, în ideea de a putea completa manual listingul cu aceste caractere. Prin POKE 64446.CODE "?" se poate opta pentru alt înlocuitor (de exemplu "?"). Pentru a trimite caractere de control la imprimantă există două metode: prima a fost prezentată deja la configurarea soft a imprimantei. A doua constă în schimbarea modului de transmisie din modul "T" (text) în modul "B" (bytes). Aceasta se realizează cu instrucțiunea:

POKE 64016.CODE "b" <ENTER> sau POKE 64016.CODE "B" <ENTER>

Modurile sunt analoge modurilor de transmisie ale Interfeței 1 Sinclair. În modul "B", *formatter*-ul text este ocolit, fiecare caracter fiind trimis ca atare. LPRINT și RST #10 funcționează, dar comanda LLIST nu este recomandabilă. Revoinirea la modul "T" se face cu:

POKE 64016.CODE "t" <ENTER> sau POKE 64016.CODE "T" <ENTER>

Un exemplu de utilizare a modului "B" îl constituie scrierea unui titlu cu litere mari. La imprimanta Romom, pentru a scrie cu litere mari, trebuie dată secvența de caractere de control:

LARGE PRINT ON: ESC, "[", "1", "m" sau CHR 27.91.49.109

Se procedează astfel:

POKE 64016.CODE "b": LPRINT CHR\$ 27: CHR\$ 91: CHR\$ 49: CHR\$ 109:: POKE 64016.CODE "t": LPRINT "TITLU"

Pentru a experimenta diverse caractere de control, consultați manualul imprimantei.

4.1.2.3. Tipărirea fișierelor Tasword

Pentru tipărirea fișierelor Tasword există două comenzi specifice. Dacă fișierul se găsește pe casetă și trebuie încărcat, se dă comanda: RANDOMIZE USR 64003 <ENTER> iar dacă fișierul se găsește în memorie, se dă comanda: RANDOMIZE USR 64006 <ENTER>.

Prima comandă (USR 64003) are următorul efect:

- șterge fișierul Tasword din memorie;
- încarcă primul fișier Tasword întâlnit; în caz de eroare la încărcare, afișează 'LOAD ERROR', șterge fișierul incomplet sau eronat și reia încărcarea;
- dacă OPTN bitul 0 = 0, atunci inițializează *formatter*-ul;
- ca *header* de pagină este luat numele fișierului, la care se adaugă extensia <Tasword>;
- trimite fișierul la imprimantă, pe 64 de coloane, fără să altereze NR_COL; dacă se întâlnește caracterul grafic CHR 143 (#8Fh) (pătratul negru), se forțează salt la pagină nouă (se pune, de exemplu, la încheierea unui capitol);
- dacă OPTN bitul 0 = 1, atunci la sfârșit trimite o linie albă (în cazul în care la generare nu s-a renunțat la separarea fișierelor cu o linie);
- dacă OPTN bitul 0 = 0, atunci la sfârșit se forțează salt la pagină nouă;
- revine în Basic.

Dacă se dorește o nouă copie, se dă comanda RANDOMIZE USR 64006, deoarece fișierul se află deja în memorie. Vom explica acum mai în amănunt ce face opțiunea OPTN bitul 0. Pentru aceasta, să presupunem că trebuie să tipărim la imprimantă fișierele Tasword de pe o casetă. În acest caz există două posibilități: fișierele să fie independente sau să fie părți ale unui text mai lung (de exemplu aceste instrucțiuni sunt stocate în 7 fișiere). În primul caz, fiecare fișier trebuie început pe o pagină nouă, iar ultima pagină trebuie încheiată cu FORM FEED. În al doilea caz, numai primul fișier este început pe o pagină nouă, iar numerotarea paginilor începe de la 1; următoarele fișiere se tipăresc de unde s-a terminat fișierul anterior, eventual cu o linie între ele (alegerea se face la generare), paginile numerotându-se în continuare. Celor două situații le corespund opțiunile OPTN bitul 0 = 0 și respectiv OPTN bitul 0 = 1. De exemplu, dacă doriți încărcarea de pe casetă și tipărirea unui text segmentat în 7 fișiere Tasword (7 segmente), se pune OPTN bitul 0 pe 1, cu:

```
POKE 64017,2*INT (PEEK 64017/2)+1 <ENTER>
```

și se dau comenzile:

```
LPRINT CHR$ 0: FOR i=1 TO 7: RANDOMIZE USR 64003: NEXT i
```

Pentru tipărirea unui fișier de pe *microdrive*, disc, rețea sau prin RS232, se șterge zona cu:

RANDOMIZE USR 64003 <ENTER> <BREAK>

apoi se încarcă fișierul Tasword la adresa 32000, din Basic:

```
LOAD *..."nume"CODE 32000 <ENTER>
```

și se inițializează *formatter*-ul, specificându-se, dacă este cazul, titlul documentului, înainte de tipărirea propriu-zisă:

```
LPRINT CHR$ 0:"TITLUL DOCUMENTULUI": RAND USR 64006 <ENTER>.
```

Ștergerea prealabilă a vechiului fișier este foarte importantă și se impune înainte de încărcarea fiecărui fișier de pe *microdrive*. Dacă vechiul fișier nu a fost șters, și noul fișier este mai scurt decât cel vechi, atunci, cu RAND USR 64006, se tipărește întregul text din memorie la imprimantă, adică noul fișier plus o rămășiță din vechiul fișier. De asemenea, este necesară și specificarea titlului documentului la încărcarea de pe disc sau *microdrive*, spre deosebire de cazul încărcării de pe casetă, unde titlul documentului îl constituie chiar numele fișierului cu terminația <Tasword>.

Dacă doriți tipărirea mai multor fișiere care se încarcă de pe *microdrive*, puteți evita prin POKE 64660,201 necesitatea apăsării de fiecare dată a combinației BREAK. De exemplu, tipărirea acestui manual segmentat în 7 fișiere, dacă acestea se găsesc pe *microdrive*-ul 1, se poate face cu următorul program simplu:

```
10 POKE 64660,201: LPRINT CHR$ 0:"MANUAL Tuttiprint"  
20 FOR i=1 TO 7  
30 RANDOMIZE USR 64003: REM se șterge vechiul fișier  
40 LOAD *"m":1:"tp"+STR$ i+".man"CODE 32000  
50 RANDOMIZE USR 64006: REM se tipărește noul fișier  
60 NEXT i
```

Exemplul de mai sus exploatează cazul particular în care numele fișierelor sunt identice, cu excepția unei cifre. Programul următor are același efect, dar este mai general. Pentru a-l folosi cu alte fișiere, schimbați linia 70 DATA și *header*-ul din linia 10.

```
10 POKE 64660,201: LPRINT CHR$ 0:"MANUAL "Tuttiprint"  
15 RESTORE  
20 READ n$: IF NOT LEN n$ THEN GO TO 100: REM terminat  
30 RANDOMIZE USR 64003: REM se șterge vechiul fișier  
40 LOAD *"m":1:n$CODE 32000  
50 RANDOMIZE USR 64006: REM se tipărește noul fișier  
60 GO TO 20  
70 DATA "tp1.man", "tp2.man", "tp3.man", "tp4.man", "tp5.man", "tp6.man", "tp7.  
man", ""
```

Dacă în memorie nu se află nici un fișier sau acesta a fost șters cu manevra RAND USR 64003 <ENTER> <BREAK>, atunci comanda RAND USR 64006 nu are sens și trebuie evitată.

Caracterul grafic 143 (pătratul negru) întâlnit într-un fișier Tasword forțează *formatter*-ul să avanseze la pagină nouă. Acest caracter poate apărea

numai la începutul liniei. Cu POKE 64835.CODE "*" se alege un alt caracter cu acest rol, noul caracter fiind specificat între ghilimele. Prin POKE-uri se obțin diverse efecte în programul Tuttiprint. În lista de mai jos sunt date câteva POKE-uri mai utile. În paranteze drepte apare conținutul inițial al adreselor respective.

Eliminarea OUT-urilor pentru interfața Pirate de comandă a casetofonului, în situația când *port*-ul 127 are altă destinație:

POKE 64660.24: POKE 64661.2: POKE 64731.24 [62,3,62]

Acceptarea fișierelor cu eroare la încărcare ("LOAD ERROR"):

POKE 64729.0: POKE 64730.0 [48,29]

Eliminarea testelor de adresă și lungime ale fișierului încărcat, în vederea acceptării fișierelor Spectral Writer și Easy Word, alături de cele Tasword:

POKE 64704.0: POKE 64710.0: POKE 64716.0 [68,62,56]

Mărirea spațiului de stocare a fișierelor de la 20480 octeți la 31500 octeți, ceea ce oferă posibilitatea ștergerii și tipăririi integrale a fișierelor mai lungi Spectral Writer și Easy Word:

POKE 64651.13: POKE 64652.123: POKE 64822.13: POKE 64823.123
[1.80.1.80]

Eliminarea liniei care desparte fișierele segmentate, atunci când se dorește tipărirea fișierelor în care textul este gata împărțit în pagini:

POKE 64839.0 [215]

Eliminarea terminației "<TASWORD>" din *header*:

POKE 64753.128 [118]

Încărcarea de pe casetă și tipărirea fișierelor segmentate cu *header* unic, independent de numele fișierelor, în vederea: a) identificării întregului document printr-un singur *header*, mai convenabilă în unele situații: b) adaptării *header*-elor la structura lucrării (împărțirea pe capitole) mai degrabă decât la structura fișierelor:

POKE 64671.224: POKE 64672.90: POKE 64684.224: POKE 64685.90
: POKE 64695.235: POKE 64696.90: POKE 64746.24: POKE 64747.4
: POKE 64785.234: POKE 64786.90: POKE 64790.225
: POKE 64791.90: POKE 64753.128

[251,254,251,254,6,255,33,252,5,255,252,254,118]

Revenirea în Basic după ștergerea vechiului fișier și încărcarea noului fișier:

POKE 64731,201 [62]

Revenirea în Basic după ștergerea vechiului fișier:

POKE 64660,201 [62]

Cu cele două POKE-uri de mai sus, se pot separa cele trei funcții principale ale programului de tipărire a fișierelor Tasword și anume: ștergerea vechiului fișier, încărcarea noului fișier de pe bandă, tipărirea propriu-zisă. Apelarea celor trei părți se face astfel:

RAND USR 64003 — ștergerea vechiului fișier

RAND USR 64664 — încărcarea de pe bandă a noului fișier

RAND USR 64006 — tipărirea la imprimantă

În cele două exemple din paginile anterioare am văzut cum este exploatată posibilitatea separării celor trei funcții. Tipărirea fișierelor Tasword Two cu programul Tuttiprint, după cum am constatat, este posibilă și chiar destul de flexibilă. Totuși, în exploatare s-au observat unele dezavantaje: a) la fișierele segmentate, dacă la un moment dat apare un incident (de exemplu se rupe hârtia), toată operația trebuie reluată de la început; b) la fișierele segmentate nu se pot scoate mai multe copii; c) nu există posibilitatea prevenirii împărțirii pe două pagini a unor zone care trebuie să apară continue (de exemplu tabele, locuri pentru figuri, formule cu indici); d) unele fișiere care conțin caracterul CHR 143 (pătratul negru) cu altă semnificație decât cea asumată de program, vor fi tipărite defectuos; e) nu se pot transmite alte caractere de control decât FF (FORM FEED), chiar dacă au fost definite corect în Tasword. Aceste dezavantaje sunt eliminate de o extensie a programului Tuttiprint specializată în tipărirea fișierelor de text.

4.1.2.4. *Formatter-ul grafic*

Dacă imprimanta este grafică, atunci se pot realiza copii grafice ale ecranului cu comanda:

RANDOMIZE USR 64009 <ENTER>.

Mărirea copiei poate fi aleasă prin variabila MAGN de la adresa 64018. MAGN poate lua valorile 1, 2 sau 3. Mărirea 1 pune în corespondență fiecărui pixel de pe ecran, un pixel pe imprimantă. La mărirea 2, unui pixel ecran îi corespunde un pătrat cu latura de 2 pixeli pe imprimantă. La mărirea 3, latura pătratului este de 3 pixeli pe imprimantă.

Avantajele *formatter*-ului grafic Tuttiprint față de alte programe similare sunt: viteza superioară, mai puțină memorie ocupată și faptul că se copiază întreg ecranul (256 × 192 pixeli). *Formatter*-ele grafice pentru Robotron K6313 și Romom lasă în stânga imaginii, pe pagină, o margine definită de NR_SPA. La CPI=10 și mărirea 2, imaginea poate ieși din cadru dacă NR_SPA>=10. La aceste imprimante de format A4, mărirea 3 iese oricum din cadru. Pentru o reprezentare cât mai completă, se recomandă NR_SPA=1.

Un efect neplăcut care apare la imprimanta Romom se manifestă prin separarea primei linii a copiei grafice de restul copiei și poate fi înlăturat prin următoarea secvență dată după introducerea hârtiei în imprimantă:

```
POKE 64016.CODE "b": LPRINT CHR$27;"[1e"::
POKE 64016.CODE "t" <ENTER>.
```

4.1.2.5. Harta memoriei

Segmentul principal Tuttiprint are următoarea structură:

64000	#FA00	ORG	Salturi în punctele de intrare
64012	#FA0C	NR_SPA	Variabilele interne principale
64049	#FA31	INIT	Activare Tuttiprint
64063	#FA3F	ASIGN3	Redirijarea fluxului #3 output la (DE)
64081	#FA51	KEYB	Așteptarea apăsării unei taste
64090	#FA5A	SENTXT	Tipărire de text de la HL până când bitul 7=1
64102	#FA66	RESET	Inițializarea <i>formatter</i> -ului de text
64176	#FAB0	CTRL1	Stochează linia de după CHR 0.1 la HEADER
64240	#FAF0	PUT_C	<i>Formatter</i> -ul text (apelată de RST #10)
64634	#FC7A	headsp	Date ale subrutinei PUT_C
64647	#FC87	TASPR	Încărcarea și tipărirea fișierelor Tasword
64796	#FD1C	PRINT	Tipărirea fișierelor Tasword
64864	#FD60	tasw	Date ale subrutinei TASPR
64897	#FD81	POINT	Testează un pixel de coordonate DE, rezultă Z
64935	#FDA7	SEND	Rutina de trimitere a unui caracter la imprimantă (începe cu 3 NOP-uri pentru o eventuală redirijare cu CALL NN)
65055	#FE1F	HCOPY	<i>Formatter</i> -ul grafic și date
65275	#FEFB	HEAD	Un octet de date și textul <i>header</i> -ului
65368	#FF58		Începutul zonei de UDG-uri

4.1.2.6. Compatibilitatea cu alte programe

Pentru ieșire la imprimantă din Hisoft Pascal, după punerea sub tensiune a calculatorului, se fac următoarele operații:


```
CLEAR 63999: LOAD "tpc"CODE : RANDOMIZE USR 64000:  
LPRINT CHR$ 1:"Hisoft PASCAL": LOAD "HP4TM16"
```

La întrebările 'Top of RAM ?' și 'Top of RAM for 'T' ?' se răspunde cu un număr mai mic sau egal cu 63999. Pentru ieșirea la imprimantă, se prevede în program ca primă instrucțiune:

```
WRITE(CHR(16));
```

Acest caracter de control comută dispozitivul de ieșire între ecran și imprimantă (fluxurile #2 și #3). Deci pentru revenirea la ecran, se folosește aceeași instrucțiune. Revenirea la ecran trebuie făcută înainte de terminarea execuției programului, altfel atât *prompter*-ul cât și comenzile introduse vor apărea tot la imprimantă. Spre deosebire de alte programe similare, *formatter*-ul Tuttiprint V4.7 interpretează corect procedura PAGE, avansând la pagină nouă. Dacă se tipărește pe coli normale, cu opțiunea de așteptare a unei taste, se recomandă înlocuirea apelurilor procedurii PAGE cu apeluri ale procedurii de mai jos:

```
PROCEDURE WPAGE:  
VAR I,J:INTEGER;  
BEGIN FOR I:=1 TO 180 DO FOR J:=1 TO 180 DO: PAGE END;
```

Schimbarea opțiunilor de imprimare se face cu instrucțiunea:

```
POKE(#FAxx,CHR(yy));
```

De exemplu, trecerea în modul "B" se face cu:

```
POKE(#FA10,'B');
```

Trimiterea caracterelor de control din Pascal se face cu procedura de mai jos:

```
CONTROL(C:CHAR):  
BEGIN  
POKE(#FA10,'B');  
INLINE(#FD,#21,#3A,#5C,#DD,#7E,2,#D7);  
POKE(#FA10,'T');  
END;
```

De exemplu, scrierea unui titlu cu litere dublu-late la imprimanta Robotron K6313, dintr-un program Pascal, se va face cu:

```
CONTROL(CHR(14)); WRITELN('TITLU');
```

Pentru copiere grafică va fi folosită instrucțiunea:

```
USER(#FA09);
```

Pentru listarea programelor care se compilează fără erori, se introduc

prima și ultima linie din listing, opțiunile de compilare următoare, apoi se dă comanda C:

```
1 {$P+}
```

```
32767 {$P-}
```

Trebuie remarcat faptul că opțiunea L- trebuie scoasă dacă există, altfel compilatorul nu va produce listing. Pentru listarea programelor, mai există posibilitatea de a comuta direct de la tastatură ieșirea pe ecran sau la imprimantă, cu tasta CS+3 (TRUE VIDEO), și de a da comanda de listare din editor (L). Această metodă are inconvenientul că listingul produs prezintă intervale goale între anumite coloane, datorate editorului. Pentru ieșire la imprimantă din Gens, după punerea sub tensiune a calculatorului, se fac următoarele operații:

```
CLEAR 63999: LOAD "tpc"CODE : RANDOMIZE USR 64000:  
LPRINT CHR$ 1:"Hisoft GENS": LOAD "GENS3M21"CODE xxxxx:  
POKE xxxxx+51,0: POKE xxxxx+50,y: RANDOMIZE USR xxxxx
```

Cele două POKE-uri modifică în Gens formatul de ieșire, ținând cont de faptul că imprimanta are NR_COL=32 coloane. Primul POKE elimină formatul de listare specific pentru 32 de coloane, iar cel de-al doilea POKE se referă la numărul de simboluri pe linie, în tabela de simboluri. În mod normal, pentru ecran acest număr este 2. Pentru imprimantă, el se dă în funcție de NR_COL (64014):

```
y = INT(PEEK(64014)/15)
```

Amintim că tabela de simboluri este afișată cu opțiunea 1 la asamblare. Pentru ca listarea să se producă la imprimantă, se va utiliza opțiunea 8. De exemplu, pentru listarea la imprimantă, cu tabela de simboluri și fără generare de cod (opțiunea 2), se dă opțiunea 11 (8+1+2). Pentru punerea la punct a unui program în limbaj de asamblare, listingurile imprimate sunt mult mai folositoare decât cele de pe ecran. Vă puteți construi o versiune de Gens specială pentru imprimantă, prin adăugarea în față a unui *loader* Basic, urmat de codul "tpc". Vă recomandăm următorul *loader* Basic:

```
10 CLEAR 25999: LOAD "tpc"CODE : RANDOMIZE USR 64000  
20 LOAD "GENS3M21"CODE 26000: POKE 26051,0:  
POKE 26050,INT(PEEK 64014/15)  
30 RANDOMIZE USR 26000  
40 RANDOMIZE USR 64000: STOP: GO TO 30
```

Pentru comoditate, comanda B (Basic) din Gens execută și o inițializare a *formatter*-ului, în vederea începerii unei noi listări. Din Basic se poate da LPRINT CHR\$ 0:"titlul", revenirea în Gens făcându-se cu CONTINUE.

Pentru dezasamblarea la imprimantă cu Mons, vă recomandăm renunțarea la opțiunea de tipărire cu *header* și cu numerotarea paginilor (OPTN bitul 1 = 0), deoarece această opțiune necesită un spațiu prea mare pentru stivă.

Pentru ieșire la imprimantă din C, după punerea sub tensiune a calculatorului, se fac următoarele operații:

```
CLEAR 63999: LOAD "tpc"CODE : RANDOMIZE USR 64000:  
LPRINT CHR$ 1;"Hisoft C": LOAD "cc"
```

Tipărirea unui fișier sursă se face din editor, cu comanda W<n1>.<n2>, unde <n1> și <n2> reprezintă numărul primei linii și respectiv al ultimei linii din program.

Pentru ieșirea la imprimantă din programele C, se vor folosi funcțiile standard de intrare/ieșire, având ca *pointer* la fișier valoarea 3, de exemplu: putc(c.3).

Programul Tuttiprint este compatibil cu orice alt program care nu ocupă zona de RAM de la 64000 la 65368 și care nu redirecționează fluxul de date (*stream-ul*) #3.

4.1.2.7. Comenzile programului de interfață Tuttiprint V4.7

Apelarea funcțiilor programului:

- USR 64000 — activarea și inițializarea *formatter*-ului de text;
- USR 64003 — încărcarea de pe casetă și tipărirea fișierelor de text Tasword Two;
- USR 64006 — tipărirea fișierelor de text Tasword Two aflate în memorie de la adresa 32000;
- USR 64009 — *hardcopy* grafic al ecranului.

Formatter-ul de text interpretează următoarele comenzi:

- LPRINT CHR\$ 0 — inițializarea *formatter*-ului de text;
- LPRINT CHR\$ 0:"header" — inițializarea *formatter*-ului de text și definirea unui nou *header*;
- LPRINT CHR\$ 1:"header" — definirea unui nou *header*.

Formatter-ul de text interpretează următoarele caractere și funcții care figurează în lista de argumente a liniei LPRINT:

- CHR\$ 12 — avans la pagină nouă;
- TAB c — avans la poziția c din linie;
- AT 1,c — avans la poziția c din linia 1 de după linia curentă;
- INK / PAPER / FLASH / BRIGHT / OVER / INVERSE a — specificațiile atributelor și caracterele de control care le corespund sunt ignorate;
- . — avans la începutul următoarei zone de 16 coloane.

Variabilele interne principale se citesc cu funcția PEEK și se modifică cu instrucțiunea POKE. Ele sunt date în tabelul următor:

Identificator	Adresa (Zecimal/Hexa)	Lungimea	Semnificația (Restricții)
NR_SPA	64012 #FA0C	1	Marginea la stânga în nr. spații (≥ 1)
NR_LIN	64013 #FA0D	1	Nr. linii pe pagină (> 4)
NR_COL	64014 #FA0E	1	Nr. caractere pe linie (≥ 32)
NR_SPX	64015 #FA0F	1	Marginea la stânga pe verso (≥ 1)
MODE	64016 #FA10	1	Modul "T" = text, sau "B" = caracter de control (CODE "T", CODE "B")
OPTN	64017 #FA11	1	Opțiuni: valoarea este suma numerelor din paranteze: (0÷15) BIT 0 (1) = fișiere Tasword segmentate (0) = fișiere Tasword independente BIT 1 (2) = cu header și numerotarea paginilor (0) = fără header și numerotarea paginilor BIT 2 (4) = așteptarea unei taste după fiecare pagină (0) = Nu este necesară așteptarea unei taste BIT 3 (8) = margine alternantă pentru imprimare verso (0) = margine fixă la stânga
MAGN	64018 #FA12	1	Mărirea copiei grafice (1, 2 sau 3)
BAUD	64019 #FA13	1	Viteza de transmisie serială, bauds (0÷6): 0=50, 1=110 2=300, 3=600, 4=1200, 5=2400, 6=4800
CFIGM	64020 #FA14	23	Mesaj de configurare soft a imprimantei (serie de caractere de control + 13 + 138)
PAG	64043 #FA2B	4	Numărul de pagină în cifre ASCII pentru mii, sute, zeci și unități (CODE "0"÷ CODE "9")
LIN	64640 #FC80	1	Contorul de linii
COL	64641 #FC81	1	Contorul de coloane
FLAGS	64636 #FC7C	1	Flag-uri interne: semnificația biților în "1": BIT 0 = urmează început de pagină; BIT 1 = urmează început de linie; BIT 2 = ignoră următorul caracter; BIT 3 = ignoră următoarele două caractere; BIT 4 = precedentul caracter a fost TAB; BIT 5 = precedentul caracter a fost AT.
HEADER	65276 #FEFC	64	Header (text terminat cu bitul 7 setat)

4.1.2.8. Listingul programului

HISOFT GEN3M2 ASSEMBLER
ZX SPECTRUM

Copyright (C) HISOFT 1983.4
All rights reserved

Pass 1 errors: 00

```

10 :TUTTIPRINT V4.7
20 :PRINTER DRIVER FORMATTER
30 :(C)1987 O Pleter
40 ;
FA00                                ORG 64000
FA00 C331FA                        60    JP  INIT      :64000
FA03 C387FC                        70    JP  TASPRT    :64003
FA06 C31CFD                        80    JP  PRINT     :64006
FA09 C31FFE                        90    JP  HCOPY     :64009
FA0C 08                          100   NR SPA  DEFB 8      :64012
FA0D 3A                          110   NR LIN  DEFB 58     :64013
FA0E 48                          120   NR COL  DEFB 72     :64014
FA0F 01                          130   NR SPX  DEFB 1      :64015
FA10 54                          140   MODE  DEFB "T"    :64016
FA11 0F                          150   OPTN  DEFB %1111   :64017
FA12 01                          160   MAGN  DEFB 1      :64018
FA13 02                          170   BAUD  DEFB 2      :64019
FA14 1B                          180   CFIGM  DEFB 27     :64020
FA15 52300D8A                    190   DEFB 82.48.13.138
FA19                             200   DEFS 18
FA2B 30303031                    210   PAG  DEFB "0"."0"."0"."1".13.13+128
220
230 INIT
240 :activare TUTTIPRINT
FA31 3E03                        250   LD  A,3
FA33 CD0116                      260   CALL #1601
FA36 CD66FA                      270   CALL RESET
FA39 3E02                        280   LD  A,2
FA3B CD0116                      290   CALL #1601
FA3E C9                          300   RET
310
320 ASSIGN
330 :dirijează canalul 3 output
340 :spre rutina indicată de DE
FA3F E5                          350   PUSH HL
FA40 D5                          360   PUSH DE
FA41 214F5C                      370   LD  HL,23631 :CHANS
FA44 5E                          380   LD  E,(HL)
FA45 23                          390   INC  HL
FA46 56                          400   LD  D,(HL)
FA47 210F00                      410   LD  HL,15
FA4A 19                          420   ADD  HL,DE
FA4B D1                          430   POP  DE
FA4C 73                          440   LD  (HL),E
FA4D 23                          450   INC  HL
FA4E 72                          460   LD  (HL),D
FA4F E1                          470   POP  HL
FA50 C9                          480   RET
490
500 KEYB
510 :așteaptă apăsarea unei taste
FA51 AF                          520   XOR  A
FA52 DBFE                        530   IN   A.(#FE)

```


FA54 2F	540	CPL	
FA55 E61F	550	AND	#1F
FA57 28F8	560	JR	Z.KEYB
FA59 C9	570	RET	
	580		
	590	SENTXT	
	600	:print	text de la HL până când
	610	:bitul	7 este găsit setat
FA5A 7E	620	LD	A,(HL)
FA5B F5	630	PUSH	AF
FA5C E67F	640	AND	#7F
FA5E D7	650	RST	#10
FA5F 23	660	INC	HL
FA60 F1	670	POP	AF
FA61 E680	680	AND	#80
FA63 28F5	690	JR	Z.SENTXT
FA65 C9	700	RET	
	710		
	720	RESET	
	730	:inițializează formatter-ul:	
	740	:LIN:=0, COL:=0, PAG:=1, MODE:="T", ASIGN3 PUT_C	
FA66 11F0FA	750	LD	DE,PUT C
FA69 CD3FFA	760	CALL	ASIGN3-
FA6C 3E42	770	LD	A,"B"
FA6E 3210FA	780	LD	(MODE),A
FA71 2114FA	790	LD	HL,CFIGM
FA74 CD5AFA	800	CALL	SENTXT
FA77 3A0CFA	810	LD	A,(NR SPA)
FA7A 324BFB	820	LD	(s b1K-1),A
FA7D 3A0EFA	830	LD	A,(NR COL)
FA80 3230FC	840	LD	(p wdTh+1),A
FA83 3A0DFA	850	LD	A,(NR LIN)
FA86 32D4FB	860	LD	(p dpTh+1),A
FA89 3E54	870	LD	A,"T"
FA8B 3210FA	880	LD	(MODE),A
FA8E 3E31	890	LD	A,"1"
FA90 322EFA	900	LD	(PAG+3),A
FA93 3D	910	DEC	A
FA94 322DFA	920	LD	(PAG+2),A
FA97 322CFA	930	LD	(PAG+1),A
FA9A 322BFA	940	LD	(PAG+0),A
FA9D AF	950	RESLC XOR	A
FA9E 327EFC	960	LD	(lin),A
FAA1 217CFC	970	LD	HL,flags
FAA4 CBC6	980	SET	0,(HL)
FAA6 AF	990	RESC XOR	A
FAA7 327FFC	1000	LD	(col),A
FAAA 217CFC	1010	LD	HL,flags
FAAD CBCE	1020	SET	1,(HL)
FAAF C9	1030	RET	
	1040		
	1050	CTRL1	
	1060	:stochează linia după CHR 0 sau 1	
	1070	:la HEADER pentru a fi tipărită pe	
	1080	:fiecare pagină	
FA80 11BFFA	1090	LD	DE,STORE
FA83 CD3FFA	1100	CALL	ASIGN3
FA86 21FCFE	1110	LD	HL,HEADER
FA89 227AFC	1120	LD	(headsp),HL
FAB8 C336FC	1130	JP	return
FABF D5	1140	STORE PUSH	DE
FAC0 E5	1150	PUSH	HL
FAC1 FE0D	1160	CP	#0D
FAC3 280B	1170	JR	Z,end c1
FAC5 2A7AFC	1180	LD	HL,(headsp)

FAC8	77	1190	LD	(HL),A
FAC9	23	1200	INC	HL
FACA	227AFC	1210	LD	(headsp),HL
FACD	E1	1220	ret_c1	POP HL
FACE	D1	1230	POP	DE
FACF	C9	1240	RET	
FAD0	2A7AFC	1250	end_c1	LD HL,(headsp)
FAD3	11FCFE	1260	LD	DE,HEADER
FAD6	A7	1270	AND	A
FAD7	ED52	1280	SBC	HL,DE
FAD9	7C	1290	LD	A,H
FADA	B5	1300	OR	L
FADB	2806	1310	JR	Z,end
FADD	2A7AFC	1320	LD	HL,(headsp)
FAE0	2B	1330	DEC	HL
FAE1	CBFE	1340	SET	7,(HL)
FAE3	11FOFA	1350	end	LD DE,PUT_C
FAE6	CD3FFA	1360	CALL	ASIGN3
FAE9	18E2	1370	JR	ret_c1
FAEB	CD66FA	1380	CTRL0	CALL RESET
FAEE	18C0	1390	JR	CTRL1
		1400		
		1410	PUT_C	
		1420	:apelață de RST #10 pentru scrierea unui	
		1430	:caracter transmis prin A	
FAF0	C5	1440	PUSH	BC
FAF1	D9	1450	EXX	
FAF2	C5	1460	PUSH	BC
FAF3	D5	1470	PUSH	DE
FAF4	E5	1480	PUSH	HL
FAF5	F5	1490	PUSH	AF
FAF6	2110FA	1500	LD	HL,MODE
FAF9	CB4E	1510	BIT	1,(HL)
FAFB	2806	1520	JR	Z,mode_t
FAFD	CDA7FD	1530	mode_b	CALL SEND
FB00	C336FC	1540	JP	return
FB03	FE02	1550	mode_t	CP 2
FB05	DA53FB	1560	JP	C.contB
FB08	FE0C	1570	CP	12
FB0A	2847	1580	JR	Z,contB
FB0C	217CFC	1590	LD	HL,flags
FB0F	CB46	1600	BIT	0,(HL) :început de pagină
FB11	282E	1610	JR	Z,contL
FB13	CB86	1620	RES	0,(HL)
FB15	2111FA	1630	LD	HL,OPTN
FB18	CB4E	1640	BIT	1,(HL)
FB1A	2825	1650	JR	Z,contL
FB1C	21FCFE	1660	LD	HL,HEADER
FB1F	CD5AFA	1670	CALL	SENTXT
FB22	212BFA	1680	LD	HL,PAG
FB25	3E30	1690	LD	A,"0"
FB27	0608	1700	LD	B,8
FB29	BE	1710	pgloop	CP (HL)
FB2A	2003	1720	JR	NZ.pgcont
FB2C	23	1730	INC	HL
FB2D	10FA	1740	DJNZ	pgloop
FB2F	3A30FC	1750	pgcont	LD A,(p_wdth+1)
FB32	90	1760	SUB	B
FB33	3281FC	1770	LD	(pagn+1),A
FB36	E5	1780	PUSH	HL
FB37	2180FC	1790	LD	HL,pagn
FB3A	CD5AFA	1800	CALL	SENTXT
FB3D	E1	1810	POP	HL
FB3E	CD5AFA	1820	CALL	SENTXT
FB41	217CFC	1830	contL	LD HL,flags

FB44 CB4E	1840	BIT	1.(HL)	:început de linie
FB46 280B	1850	JR	Z.contB	
FB48 CB8E	1860	RES	1.(HL)	
FB4A 0608	1870	LD	B.8	
FB4C 3E20	1880	s_blk LD	A." "	
FB4E CDA7FD	1890	CALL	SEND	
FB51 10F9	1900	DJNZ	s_blk	
FB53 F1	1910	contB POP	AF	
FB54 F5	1920	PUSH	AF	
FB55 217CFC	1930	LD	HL,flags	
FB58 CB5E	1940	BIT	3.(HL)	:sare 2 caractere
FB5A 280A	1950	JR	Z.contC	
FB5C CB9E	1960	RES	3.(HL)	
FB5E CBD6	1970	SET	2.(HL)	
FB60 327DFC	1980	LD	(byte).A	
FB63 C336FC	1990	JP	return	
FB66 CB66	2000	contC BIT	4.(HL)	:TAB
FB68 2821	2010	JR	Z.contD	
FB6A CBA6	2020	RES	4.(HL)	
FB6C CB96	2030	RES	2.(HL)	
FB6E 3A7DFC	2040	LD	A.(byte)	
FB71 217FFC	2050	atcnx LD	HL.col	
FB74 BE	2060	CP	(HL)	
FB75 DC85FB	2070	CALL	C.nextln	
FB78 96	2080	SUB	(HL)	
FB79 CA36FC	2090	JP	Z.return	
FB7C 47	2100	LD	B.A	
FB7D 3E20	2110	taloop LD	A." "	
FB7F D7	2120	RST	#10	
FB80 10FB	2130	DJNZ	taloop	
FB82 C336FC	2140	JP	return	
FB85 F5	2150	nextln PUSH	AF	
FB86 3E0D	2160	LD	A.13	
FB88 D7	2170	RST	#10	
FB89 F1	2180	POP	AF	
FB8A C9	2190	RET		
FB8B CB8E	2200	contD BIT	5.(HL)	:AT
FB8D 2814	2210	JR	Z.contX	
FB8F CBAE	2220	RES	5.(HL)	
FB91 CB96	2230	RES	2.(HL)	
FB93 3A7DFC	2240	LD	A.(byte)	
FB96 A7	2250	AND	A	
FB97 2806	2260	JR	Z.athrz	
FB99 47	2270	atver LD	B.A	
FB9A 3E0D	2280	atloop LD	A.13	
FB9C D7	2290	RST	#10	
FB9D 10FB	2300	DJNZ	atloop	
FB9F F1	2310	athrz POP	AF	
FBA0 F5	2320	PUSH	AF	
FBA1 18CE	2330	JR	atcnx	
FBA3 CB56	2340	contX BIT	2.(HL)	:sare 1 caracter
FBA5 2805	2350	JR	Z.contN	
FBA7 CB96	2360	RES	2.(HL)	
FBA9 C336FC	2370	JP	return	
FBAC F1	2380	contN POP	AF	
FBAD F5	2390	PUSH	AF	
FBAE FEA5	2400	CP	#A5	
FBB0 3807	2410	JR	C.single	
FBB2 D6A5	2420	token SUB	#A5	
FBB4 CD100C	2430	CALL	#0C10	
FBB7 187D	2440	JR	return	
FBB9 FE7F	2450	single CP	#7F	
FBBB 3802	2460	JR	C.eoln	
FBBD 3E20	2470	GRAPH LD	A." "	
FBBF FE0D	2480	eoln CP	#0D	

FBC1	2052	2490	JR	NZ.cntrl	
FBC3	CDA7FD	2500	CALL	SEND	
FBC6	3E0A	2510	LD	A.10	
FBC8	CDA7FD	2520	CALL	SEND	
FBCB	CDA6FA	2530	CALL	RESC	
FBCE	217EFC	2540	LD	HL.lin	
FBD1	34	2550	INC	(HL)	
FBD2	7E	2560	LD	A.(HL)	
FBD3	FE3A	2570	p_dpth CP	58	
FBD5	205F	2580	JR	NZ.return	
FBD7	3E0C	2590	form_f LD	A.12	
FBD9	CDA7FD	2600	CALL	SEND	
FBDC	CD9DFA	2610	CALL	RESLC	
FBDF	3A11FA	2620	LD	A.(OPTN)	
FBE2	E604	2630	AND	4	
FBE4	C4F9FB	2640	CALL	NZ.altn	
FBE7	013004	2650	LD	BC.#0430	
FBEA	212EFA	2660	LD	HL,PAG+3	
FBED	3E3A	2670	LD	A.#3A	
FBEF	34	2680	pilloop INC	(HL)	
FBF0	BE	2690	CP	(HL)	
FBF1	2043	2700	JR	NZ.return	
FBF3	71	2710	LD	(HL).C	
FBF4	2B	2720	DEC	HL	
FBF5	10F8	2730	DJNZ	pilloop	
FBF7	18D3	2740	JR	return	
FBF9	CD51FA	2750	altn CALL	KEYB	
FBFC	3A11FA	2760	LD	A.(OPTN)	
FBFF	E608	2770	AND	8	
FC01	C8	2780	RET	Z	
FC02	3A2EFA	2790	LD	A.(PAG+3)	
FC05	E601	2800	AND	1	
FC07	2807	2810	JR	Z.even	
FC09	3A0FFA	2820	LD	A.(NR SPX)	
FC0C	324BFB	2830	odd LD	(s_b1K-1).A	
FC0F	C9	2840	RET		
FC10	3A0CFA	2850	even LD	A.(NR_SPA)	
FC13	18F7	2860	JR	odd	
FC15	FE0C	2870	cntrl CP	12	
FC17	28BE	2880	JR	Z.form_f	
FC19	FE00	2890	CP	0	
FC1B	CAEBFA	2900	JP	Z.CTRL0	
FC1E	FE01	2910	CP	1	
FC20	CAB0FA	2920	JP	Z.CTRL1	
FC23	FE20	2930	CP	" "	
FC25	3816	2940	JR	C.cntrl	
FC27	CDA7FD	2950	CALL	SEND	
FC2A	217FFC	2960	LD	HL.col	
FC2D	34	2970	INC	(HL)	
FC2E	7E	2980	LD	A.(HL)	
FC2F	FE48	2990	p_wdth CP	72	
FC31	2003	3000	JR	NZ.return	
FC33	3E0D	3010	LD	A.13	
FC35	D7	3020	RST	#10	
FC36	F1	3030	return POP	AF	
FC37	E1	3040	POP	HL	
FC38	D1	3050	POP	DE	
FC39	C1	3060	POP	BC	
FC3A	D9	3070	EXX		
FC3B	C1	3080	POP	BC	
FC3C	C9	3090	RET		
FC3D	217CFC	3100	ctrl LD	HL.flags	
FC40	FE06	3110	CP	6	
FC42	281E	3120	JR	Z.virgo	
FC44	FE10	3130	CP	16	

FC46	38EE	3140	JR	C.return
FC48	FE16	3150	CP	22
FC4A	3808	3160	JR	C.skip1
FC4C	280A	3170	JR	Z.atc
FC4E	FE17	3180	CP	23
FC50	280C	3190	JR	Z.tabc
FC52	18E2	3200	JR	return
FC54	CBD6	3210	skip1	SET 2.(HL)
FC56	18DE	3220	JR	return
FC58	CBEE	3230	atc	SET 5.(HL)
FC5A	CBDE	3240	set3	SET 3.(HL)
FC5C	18D8	3250	JR	return
FC5E	CBE6	3260	tabc	SET 4.(HL)
FC60	18F8	3270	JR	set3
FC62	3E17	3280	virgo	LD A.23
FC64	D7	3290	RST	#10
FC65	3A7FFC	3300	LD	A.(col)
FC68	E6F0	3310	AND	#F0
FC6A	C610	3320	ADD	A.16
FC6C	2130FC	3330	LD	HL.p_width+1
FC6F	BE	3340	CP	(HL)
FC70	3005	3350	JR	NC.cr
FC72	D7	3360	vsend	RST #10
FC73	AF	3370	XOR	A
FC74	D7	3380	RST	#10
FC75	18BF	3390	JR	return
FC77	AF	3400	cr	XOR A
FC78	18F8	3410	JR	vsend
FC7A	0000	3420	headsp	DEFW 0
FC7C	03	3430	flags	DEFB %11
FC7D	00	3440	byte	DEFB 0
FC7E	00	3450	lin	DEFB 0
FC7F	00	3460	col	DEFB 0
FC80	17000050	3470	pagm	DEFB 23.0.0."P","A","G"," "+128
		3480		
		3490	TASPRT	
		3500	:încarcă un fişier TASWORD	
		3510	:de pe casetă şi îl tipăreşte	
FC87	21007D	3520	LD	HL.32000
FC8A	010150	3530	LD	BC.20481
FC8D	3600	3540	LD	(HL).0
FC8F	54	3550	LD	D.H
FC90	5D	3560	LD	E.L
FC91	13	3570	INC	DE
FC92	EDB0	3580	LDIR	
FC94	3E03	3590	LD	A.3
FC96	D37F	3600	OUT	(127).A
FC98	3E02	3610	LD	A.2
FC9A	CD0116	3620	CALL	#1601
FC9D	DD21FBFE	3630	load	LD IX.HEAD
FCA1	111100	3640	LD	DE.#11
FCA4	AF	3650	XOR	A
FCA5	37	3660	SCF	
FCA6	CD5605	3670	CALL	#556
FCA9	30F2	3680	JR	NC.load
FCAB	3AFBFE	3690	LD	A.(HEAD)
FCAE	FE03	3700	CP	3
FCB0	204E	3710	JR	NZ.basic
FCB2	DD21007D	3720	LD	IX.32000
FCB6	2106FF	3730	LD	HL.HEAD+1+10
FCB9	5E	3740	LD	E.(HL)
FCBA	23	3750	INC	HL
FCBB	56	3760	LD	D.(HL)
FCBC	3E3F	3770	LD	A.#3F
FCBE	A3	3780	AND	E

FCCB	2044	3790	JR	NZ,bytes
FCC1	23	3800	INC	HL
FCC2	7E	3810	LD	A,(HL)
FCC3	FE00	3820	CP	0
FCC5	203E	3830	JR	NZ,bytes
FCC7	23	3840	INC	HL
FCC8	7E	3850	LD	A,(HL)
FCC9	FE7D	3860	CP	#7D
FCCB	2038	3870	JR	NZ,bytes
FCCD	2160FD	3880	LD	HL,tasw
FCD0	CD0DFD	3890	CALL	titlu
FCD3	3EFF	3900	LD	A,#FF
FCD5	37	3910	SCF	
FCD6	CD5605	3920	CALL	#556
FCD9	301D	3930	JR	NC,taperr
FADB	3E02	3940	LD	A,2
FADD	D37F	3950	OUT	(127),A
FCD7	3E03	3960	LD	A,3
FCE1	CD0116	3970	CALL	#1601
FCE4	3A11FA	3980	LD	A,(OPTN)
FCE7	E601	3990	AND	1
FCE9	D7	4000	RST	#10
FCEA	21FCFE	4010	LD	HL,HEAD+1
FCEB	CD5AFA	4020	CALL	SENTXT
FCE0	2176FD	4030	LD	HL,exttas
FCF3	CD5AFA	4040	CALL	SENTXT
FCF6	1824	4050	JR	PRINT
FCF8	216AFD	4060	LD	HL,tap
FCFB	CD5AFA	4070	CALL	SENTXT
FCFE	1887	4080	JR	TASPR
FD00	21C109	4090	LD	HL,bas
FD03	1803	4100	JR	bytes2
FD05	21EC09	4110	LD	HL,byt
FD08	CD0DFD	4120	CALL	titlu
FD0B	1890	4130	JR	load
FD0D	CD5AFA	4140	CALL	SENTXT
FD10	2105FF	4150	LD	HL,HEAD+10
FD13	CBFE	4160	SET	7,(HL)
FD15	21FCFE	4170	LD	HL,HEAD+1
FD18	CD5AFA	4180	CALL	SENTXT
FD1B	C9	4190	RET	
		4200		
		4210	PRINT	
		4220	:punct	de intrare tipărire
FD1C	3E03	4230	LD	A,3
FD1E	CD0116	4240	CALL	#1601
FD21	3A11FA	4250	LD	A,(OPTN)
FD24	E601	4260	AND	1
FD26	2004	4270	JR	NZ,printB
FD28	D7	4280	RST	#10
FD29	3E0D	4290	LD	A,13
FD2B	D7	4300	RST	#10
FD2C	3E40	4310	LD	A,64
FD2E	3230FC	4320	LD	(p.wdth+1),A
FD31	21007D	4330	LD	HL,32000
FD34	AF	4340	XOR	A
FD35	010150	4350	LD	BC,20481
FD38	EDB1	4360	CPIR	
FD3A	C0	4370	RET	NZ
FD3B	37	4380	SCF	
FD3C	11007D	4390	LD	DE,32000
FD3F	ED52	4400	SBC	HL,DE
FD41	1A	4410	LD	A,(DE)
FD42	FE8F	4420	CP	#8F
FD44	CC5AFD	4430	CALL	Z,ffeed

FD47	D7	4440	RST	#10
FD48	2B	4450	DEC	HL
FD49	13	4460	INC	DE
FD4A	7C	4470	LD	A,H
FD4B	B5	4480	OR	L
FD4C	20F3	4490	JR	NZ,loop
FD4E	3E0D	4500	LD	A,13
FD50	D7	4510	RST	#10
FD51	3A11FA	4520	LD	A.(OPTN)
FD54	E601	4530	AND	1
FD56	CC5AFD	4540	CALL	Z,ffeed
FD59	C9	4550	RET	
FD5A	3E0C	4560	LD	A,12
FD5C	D7	4570	RST	#10
FD5D	3E20	4580	LD	A," "
FD5F	C9	4590	RET	
FD60	0D	4600	DEFB	13
FD61	54617377	4610	DEFM	'Tasword:'
FD69	A0	4620	DEFB	" "+128
09C1		4630	bas	EQU #9C1
09EC		4640	byt	EQU #9EC
FD6A	204C4F41	4650	tap	DEFM 'LOAD ERROR'
FD75	8D	4660	DEFB	13+128
FD76	203C5441	4670	exttas	DEFM '<TASWORD>'
FD80	8D	4680	DEFB	13+128
		4690		
		4700	POINT	
		4710		:testează un pixel de coordonate:
		4720		:D=X 0÷255, E=Y 0÷191 și rezultă:
		4730		:HL=adresa din ecran, Z dacă pixelul este 0
FD81	C5	4740	PUSH	BC
FD82	7B	4750	LD	A,E
FD83	E6C0	4760	AND	#C0
FD85	1F	4770	RRA	
FD86	37	4780	SCF	
FD87	1F	4790	RRA	
FD88	0F	4800	RRCA	
FD89	AB	4810	XOR	E
FD8A	E6F8	4820	AND	#F8
FD8C	AB	4830	XOR	E
FD8D	67	4840	LD	H,A
FD8E	7A	4850	LD	A,D
FD8F	07	4860	RLCA	
FD90	07	4870	RLCA	
FD91	07	4880	RLCA	
FD92	AB	4890	XOR	E
FD93	E6C7	4900	AND	#C7
FD95	AB	4910	XOR	E
FD96	07	4920	RLCA	
FD97	07	4930	RLCA	
FD98	6F	4940	LD	L,A
FD99	7A	4950	LD	A,D
FD9A	E607	4960	AND	7
FD9C	47	4970	LD	B,A
FD9D	04	4980	INC	B
FD9E	3E01	4990	LD	A,1
FDA0	0F	5000	ploop	RRCA
FDA1	10FD	5010	DJNZ	ploop
FDA3	4E	5020	LD	C.(HL)
FDA4	A1	5030	AND	C
FDA5	C1	5040	POP	BC
FDA6	C9	5050	RET	
		5060		
		5070	SEND	
		5080		:trimite un caracter de 7 biți la imprimantă

FDA7 000000	5090	:preluat din A	
FDA8 F5	5100	DEFB 0.0.0	
FDA9 C5	5110	PUSH AF	
FDAE 3EFF	5120	PUSH BC	
FDAE D3BE	5130	PRGM LD A,255	
FDB0 3E01	5140	OUT (190).A	
FDB2 D3BE	5150	LD A,1	
FDB4 3EFF	5160	OUT (190).A	
FDB6 D3BF	5170	LD A,255	
FDB8 3E00	5180	OUT (191).A	
FDBA D3BF	5190	LD A,0	
FDBC 0614	5200	OUT (191).A	
FDBE DBBC	5210	LD B,20	
FDC0 E601	5220	BUSY IN A,(188)	
FDC2 20FA	5230	AND 1	
FDC4 10F8	5240	BUSYPM JR NZ,BUSY	
FDC6 C1	5250	DJNZ BUSY	
FDC7 F1	5260	POP BC	
FDC8 00	5270	POP AF	
FDC9 E67F	5280	COMPL NOP	
FDCB D3BD	5290	AND #7F	
FDCD F680	5300	OUT1 OUT (189).A	
FDCF D3BD	5310	OR #80	
FDD1 E67F	5320	OUT2 OUT (189).A	
FDD3 D3BD	5330	AND #7F	
FDD5 C9	5340	OUT3 OUT (189).A	
FDD6	5350	RET	
	5360	DEFS 73	
	5370		
	5380	HCOPY	
FE1F 3E42	5390	:formatter grafic pentru ROMOM	
FE21 3210FA	5400	LD A,"B"	
FE24 3E03	5410	LD (MODE).A	
FE26 CD0116	5420	LD A,3	
FE29 3A12FA	5430	CALL #1601	
FE2C 32A4FE	5440	LD A,(MAGN)	
FE2F F680	5450	LD (printl+1).A	
FE31 32EEFE	5460	OR 128	
FE34 EE80	5470	LD (size+3).A	
FE36 1600	5480	XOR 128	
FE38 5F	5490	LD D,0	
FE39 1D	5500	LD E,A	
FE3A CB03	5510	DEC E	
FE3C 21CAFE	5520	RLC E	
FE3F 19	5530	LD HL,tabla	
FE40 5E	5540	ADD HL,DE	
FE41 23	5550	LD E,(HL)	
FE42 56	5560	INC HL	
FE43 219DFE	5570	LD D,(HL)	
FE46 73	5580	LD HL,true+1	
FE47 23	5590	LD (HL),E	
FE48 72	5600	INC HL	
FE49 47	5610	LD (HL),D	
FE4A 0E10	5620	LD B,A	
FE4C CB39	5630	LD C,#10	
FE4E CB39	5640	maloop SRL C	
FE50 10FA	5650	SRL C	
FE52 3E02	5660	DJNZ maloop	
FE54 B1	5670	LD A,2	
FE55 3271FE	5680	OR C	
FE58 328EFE	5690	LD (loop_y+4).A	
FE5B 3293FE	5700	LD (loop_x+4).A	
FE5E C6C0	5710	LD (innl-1).A	
FE60 3276FE	5720	ADD A,#C0	
	5730	LD (loop_y+9).A	

FE63	21F4FE	5740	LD	HL,norm
FE66	CD5AFA	5750	CALL	SENTXT
FE69	AF	5760	XOR	A
FE6A	32C9FE	5770	LD	(y_cd).A
FE6D	3AC9FE	5780	LD	A.(y_cd)
FE70	C606	5790	ADD	A,6
FE72	32C9FE	5800	LD	(y_cd).A
FE75	FE66	5810	CP	#C6
FE77	283F	5820	JR	Z.exit
FE79	3A0CFA	5830	LD	A.(NR_SPA)
FE7C	47	5840	LD	B.A
FE7D	3E20	5850	LD	A," "
FE7F	D7	5860	RST	#10
FE80	10FB	5870	DJNZ	spaces
FE82	21EBFE	5880	LD	HL,size
FE85	CD5AFA	5890	CALL	SENTXT
FE88	1600	5900	LD	D,0
FE8A	3AC9FE	5910	LD	A.(y_cd)
FE8D	D606	5920	SUB	6
FE8F	5F	5930	LD	E.A
FE90	0E00	5940	LD	C,0
FE92	0606	5950	LD	B,6
FE94	CD81FD	5960	CALL	POINT
FE97	37	5970	SCF	
FE98	C29CFE	5980	JP	NZ.true
FE9B	3F	5990	CCF	
FE9C	3CD0FE	6000	JP	ma_1
FE9F	1C	6010	INC	E
FEA0	10F2	6020	DJNZ	inn1
FEA2	79	6030	LD	A,C
FEA3	0601	6040	LD	B,1
FEA5	F5	6050	PUSH	AF
FEA6	D7	6060	RST	#10
FEA7	F1	6070	POP	AF
FEA8	10FB	6080	DJNZ	prloop
FEAA	14	6090	INC	D
FEAB	AF	6100	XOR	A
FEAC	BA	6110	CP	D
FEAD	C28AFE	6120	JP	NZ.loop_x
FEB0	21FOFE	6130	LD	HL,halfn1
FEB3	CD5AFA	6140	CALL	SENTXT
FEB6	18B5	6150	JR	loop_y
FEB8	21F9FE	6160	LD	HL,final
FEBB	CD5AFA	6170	CALL	SENTXT
FEBE	3E02	6180	LD	A,2
FEC0	CD0116	6190	CALL	#1601
FEC3	3E54	6200	LD	A,"T"
FEC5	3210FA	6210	LD	(MODE).A
FEC8	C9	6220	RET	
FEC9	00	6230	DEFB	0
FECA	D0FE	6240	DEFW	ma_1
FECC	D5FE	6250	DEFW	ma_2
FECE	DEFE	6260	DEFW	ma_3
FED0	CB11	6270	RL	C
FED2	C39FFE	6280	JP	conex1
FED5	F5	6290	PUSH	AF
FED6	CB11	6300	RL	C
FED8	F1	6310	POP	AF
FED9	CB11	6320	RL	C
FEDB	C39FFE	6330	JP	conex1
FEDE	F5	6340	PUSH	AF
FEDF	CB11	6350	RL	C
FEE1	F1	6360	POP	AF
FEE2	F5	6370	PUSH	AF
FEE3	CB11	6380	RL	C

FEE5 F1	6390	POP AF
FEE6 CB11	6400	RL C
FEE8 C39FFE	6410	JP conex1
FEFB 1B4B0081	6420 size	DEFB #1B,#4B,0.1+128.1+128
FEF0 1B5B3165	6430 halfnl	DEFB #1B,#5B,#31.#65
FEF4 1B5B30E0	6440 norm	DEFB #1B,#5B,#30.#60+128."2"+128
FEF9 8DB0	6450 final	DEFB 13+128."0"+128
FEFB 00	6460 HEAD	DEFB 0
FEFC 54555454	6470 HEADER	DEFM 'TUTTIPIRINT V4.7'
FFOC 5052494E	6480	DEFM 'PRINTER DRIVER FORMATTER'
FF25 28432931	6490	DEFM '(C)1987 O Plete'
FF34 F2	6500	DEFB "r"+128
FF35	6510	DEFS 30
FF53 00	6520 zzzzzz	NOP

Pass 2 errors: 00

ASIGN3 FA3F	BAUD FA13	BUSY FDBE	BUSYPM FDC2	CFIGM FA14
COMPL FDC8	CTRL0 FAEB	CTRL1 FAB0	GRAPH FBBD	HCOPY FE1F
HEAD FEFB	HEADER FEFC	INIT FA31	KEYB FA51	MAGN FA12
MODE FA10	NR COL FA0E	NR LIN FA0D	NR SPA FA0C	NR SPX FA0F
OPTN FA11	OUT1 FDCB	OUT2 FDCF	OUT3 FDD3	PAG FA2B
POINT FD81	PRGM FDAC	PRINT FD1C	PUT C FAF0	RESC FAA6
RESET FA66	RESLC FA9D	SEND FDA7	SENTXT FA5A	STORE FABF
TASLN FD50	TASPRT FC87	altn FBF9	atc FC58	atcnx FB71
athrz FB9F	atloop FB9A	atver FB99	bas 09C1	basic FD00
byt 09EC	byte FC7D	bytes FD05	bytes2 FD08	cntrl FC15
col FC7F	conex1 FE9F	contB FB53	contC FB66	contD FB88
contL FB41	contN FBAC	contX FBA3	cr FC77	ctrl FC3D
end FAE3	end c1 FAD0	coln FBBF	even FC10	exit FEB8
exttas FD76	ffeed FD5A	final FEF9	flags FC7C	form f FBD7
halfnl FEF0	headsp FC7A	innl FE94	lin FC7E	load FC9D
loop FD41	loop_x FE8A	loop_y FE6D	ma 1 FED0	ma 2 FED5
ma 3 FEDE	maloop FE4C	mode_b FAFD	mode t FB03	nextln FB85
norm FEF4	odd FC0C	p dpth FBD3	p wdth FC2F	pagm FC80
pgcont FB2F	pgloop FB29	ploop FBEF	ploop FDA0	printB FD2C
printl FEA3	prloop FEA5	ret c1 FACD	return FC36	s blk FB4C
set3 FC5A	single FBB9	size FEEB	skip FD4A	sKipl FC54
spaces FE7D	tabc FC5E	tabla FECA	taloop FB7D	tap FD6A
taperr FCF8	tasw FD60	titlu FD0D	token FBB2	true FE9C
virgo FC62	vsend FC72	y_cd FEC9	zzzzzz FF53	

Table used: 1423 from 2000

HISOFT GEN3M2 ASSEMBLER
ZX SPECTRUM

Copyright (C) HISOFT 1983.4
All rights reserved

Pass 1 errors: 00

FDA7	10	ORG #FDA7
	20	SEND
	30	:trimite serial un caracter de 7 biți
	40	:preluat din A
FA13	50	BAUD EQU #FA13
FDA7 000000	60	DEFB 0.0.0
FDA7 C5	70	PUSH BC
FDAB D5	80	PUSH DE

FDAC E5	90	PUSH HL
FDAD F5	100	PUSH AF
FDAE 060B	110	LD B, #0B
FDB0 2F	120	CPL
FDB1 4F	130	LD C, A
FDB2 CD05FE	140	CALL PRGM
FDB5 3EFF	150	LD A, 255
FDB7 D3BE	160	OUT (190).A
FDB9 3E01	170	LD A, 1
FDBB D3BE	180	OUT (190).A
FDBD 3A13FA	190	LD A, (BAUD)
FDC0 5F	200	LD E, A
FDC1 1600	210	LD D, 0
FDC3 CB23	220	SLA E
FDC5 2111FE	230	LD HL, BAUD_T
FDC8 19	240	ADD HL, DE
FDC9 5E	250	LD E, (HL)
FDCA 23	260	INC HL
FDCB 56	270	LD D, (HL)
FDCC 62	280	LD H, D
FDCD 6B	290	LD L, E
FDCE 3E00	300	LD A, 0
FDD0 D3BD	310	OUT1 OUT (189).A
FDD2 1B	320	tempol DEC DE
FDD3 7A	330	LD A, D
FDD4 B3	340	OR E
FDD5 20FB	350	JR NZ, tempol
FDD7 DBBC	360	RDY IN A, (188)
FDD9 E601	370	AND 1
Fddb 28FA	380	RDYPM JR Z, RDY
FDDD CD05FE	390	CALL PRGM
FDE0 37	400	SCF
FDE1 F3	410	DI
FDE2 CE00	420	ser1 ADC A, 0
FDE4 D3BD	430	OUT2 OUT (189).A
FDE6 54	440	LD D, H
FDE7 5D	450	LD E, L
FDE8 1B	460	tempo2 DEC DE
FDE9 7A	470	LD A, D
FDEA B3	480	OR E
FDEB 20FB	490	JR NZ, tempo2
FDED 1B	500	DEC DE
FDEE CD05FE	510	CALL PRGM
FDF1 AF	520	XOR A
FDF2 CB39	530	SRL C
FDF4 10EC	540	DJNZ ser1
FDF6 FB	550	EI
FDF7 2B	560	tempo3 DEC HL
FDF8 7D	570	LD A, L
FDF9 B4	580	OR H
FDFA 20FB	590	JR NZ, tempo3
FDfC 3E00	600	LD A, 0
FDfE D3BD	610	OUT3 OUT (189).A
FE00 F1	620	POP AF
FE01 E1	630	POP HL
FE02 D1	640	POP DE
FE03 C1	650	POP BC
FE04 C9	660	RET
FE05 3EFF	670	PRGM LD A, 255
FE07 D3BF	680	OUT (191).A
FE09 3E00	690	LD A, 0
FE0B D3BF	700	OUT (191).A
FE0D 96	710	SUB (HL)

FE0E 00	720	NOP	
FE0F 00	730	NOP	
FE10 C9	740	RET	
FE11 7F0A	750	BAUD_T	DEFB #7F, #0A :50
FE13 C204	760		DEFB #C2, #04 :110
FE15 BB01	770		DEFB #BB, #01 :300
FE17 DB00	780		DEFB #DB, #00 :600
FE19 6B00	790		DEFB #6B, #00 :1200
FE1B 3300	800		DEFB #33, #00 :2400
FE1D 1700	810	zzzzzz	DEFB #17, #00 :4800

Pass 2 errors: 00

BAUD	FA13	BAUD T	FE11	OUT1	FDD0	OUT2	FDE4	OUT3	FDFF
PRGM	FE05	PRGMI	FDB5	RDY	FDD7	RDYPM	FDD8	SEND	FDA7
ser1	FDE2	tempo1	FDD2	tempo2	FDE8	tempo3	FDF7	zzzzzz	FE1D

Table used: 189 from 234

4.1.3. SCHEMA INTERFEȚEI

Schema interfeței, figura 4.1, a fost proiectată în spiritul Sinclair, adică "minimum hardware - maximum software".

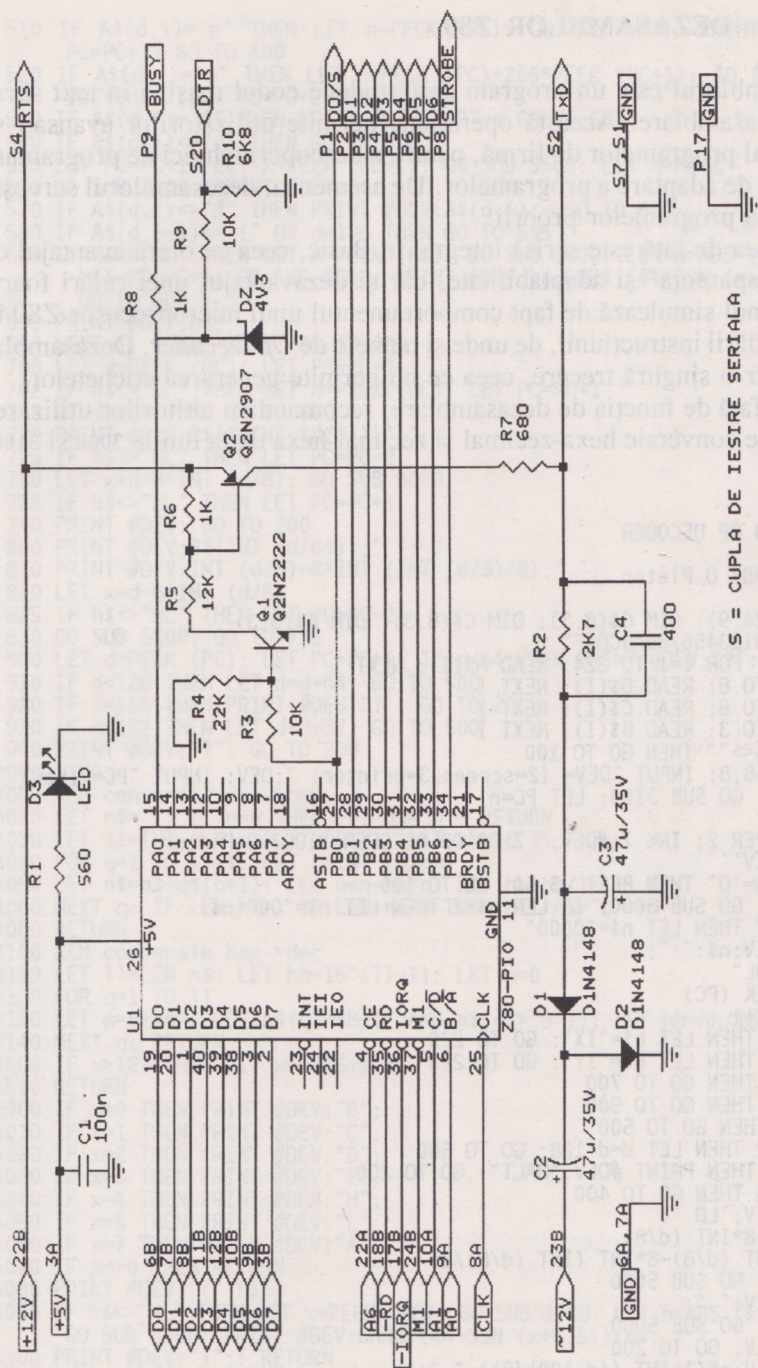
Circuitul central este un Z80A PIO, din care *port-ul B* este utilizat ca ieșire paralelă unidirecțională pe 8 biți, iar din *port-ul A* este utilizat un singur bit de intrare, pentru protocol. Bitul zero al *port-ului B* are și funcția de ieșire de date la comunicația serială. Bitul 7 al *port-ului B* are rol de STROBE în comunicația paralelă, care se face prin urmare numai pe 7 biți de date. *Port-ul B* nu este *buffer*-at la ieșire, ceea ce face schema nu numai economică, ci și mai flexibilă, întrucât comunicația poate deveni bidirecțională. Nici linia serială nu este complicată, trecerea de la nivelurile TTL la $-12\text{ V} \div +12\text{ V}$ făcându-se printr-un circuit cu două tranzistoare, iar trecerea inversă printr-o diodă Zenner. Tensiunea de $+12\text{ V}$ este preluată de la cupla de extensie a calculatorului, iar tensiunea de -12 V este creată intern din tensiunea alternativă de 12 V din cupla de extensie.

Ieșirea serială dispune de o cuplă D25, conectată conform EIA RS232, iar ieșirea paralelă se face tot printr-o cuplă D25, conectată după modelul adoptat la calculatorul Tim-S. Viteza de transmisie serială poate fi selectată prin software din gama: 110, 300, 600, 1200, 2400 și 4800 bauds. Ceilalți parametri sunt: 7 biți de date, 1 bit de stop, fără paritate.

Cablul de legătură pentru comunicația paralelă trebuie să aibă 10 linii: D0÷D6, STROBE, BUSY și GND. În cazul în care la capătul dinspre imprimantă apare intrarea D7 (al optulea bit de date), aceasta se leagă la masă.

Cablul de legătură pentru comunicația serială trebuie să aibă 3 linii: TxD, DTR și GND.

Interfața Tuttiptint utilizează o selecție liniară de *port*, pe adresa A6. Din acest motiv, ea nu este compatibilă cu alte interfețe care utilizează această linie de adrese.



S = CUPLA DE IESIRE SERIALĂ

P = CUPLA DE IESIRE PARALELA

Fig. 4.1. Schema interfeței de imprimantă

4.2. DEZASAMBLOR Z80

Dezasamblorul este un program care traduce codul mașină în text sursă de limbaj de asamblare. Această operațiune permite utilizatorilor avansați să analizeze codul programelor de firmă, pentru a descoperi tehnici de programare sau modalități de adaptare a programelor. De asemenea, dezasamblorul servește și la depanarea programelor proprii.

Versiunea de față este scrisă integral în Basic, ceea ce oferă avantajul de maximă "transparență" și adaptabilitate, dar și dezavantajul unei rulări foarte lente. Programul simulează de fapt comportamentul unui microprocesor Z80 în decursul decodării instrucțiunii, de unde și numele de *Op Decoder*. Dezasamblarea se face într-o singură trecere, ceea ce nu permite generarea etichetelor.

În plus față de funcția de dezasamblare, recomandăm cititorilor utilizarea subrutinelor de conversie hexa-zecimal și zecimal-hexa de la liniile 3000 și 3100.

```
10 REM Z-80 OP DECODER
20 REM
30 REM (C)1987 O Pleter
40 REM
50 DIM A$(224,9): DIM O$(8,3): DIM C$(8,3): DIM B$(3,3)
60 LET q$="0123456789ABCDEF"
70 RESTORE : FOR I=1 TO 224: READ A$(I): NEXT I
80 FOR I=1 TO 8: READ O$(I): NEXT I
90 FOR I=1 TO 8: READ C$(I): NEXT I
95 FOR I=1 TO 3: READ B$(I): NEXT I
100 IF INKEY$<>" " THEN GO TO 100
105 POKE 23658,8: INPUT "DEV= (2=screen,3=printer) ";DEV: INPUT "PC= (hex) ";
    LINE n$: GO SUB 3100: LET PC=n
110 CLS
120 PRINT PAPER 2; INK 7:#DEV;" Z-80 OP DECODER ?1987 O Pleter "
130 PRINT #DEV
200 IF INKEY$="Q" THEN BEEP .3,50: GO TO 100
201 LET n=PC: GO SUB 3000: IF LEN n$=2 THEN LET n$="00"+n$
202 IF n$="0" THEN LET n$="0000"
205 PRINT #DEV;n$:" ";
210 LET h$="HL"
220 LET d=PEEK (PC)
230 LET PC=PC+1
240 IF d=221 THEN LET h$="IX": GO TO 220
250 IF d=253 THEN LET h$="IY": GO TO 220
260 IF d=203 THEN GO TO 700
270 IF d=237 THEN GO TO 900
280 IF d<64 THEN GO TO 500
290 IF d>=192 THEN LET d=d-128: GO TO 500
300 IF d=118 THEN PRINT #DEV:"HALT": GO TO 200
310 IF d>=128 THEN GO TO 400
320 PRINT #DEV:"LD ";
330 LET r2=d-8*INT (d/8)
340 LET r1=INT (d/8)-8*INT (INT (d/8)/8)
350 LET x=r1: GO SUB 5000
360 PRINT #DEV:". ";
370 LET x=r2: GO SUB 5000
380 PRINT #DEV: GO TO 200
400 PRINT #DEV;o$(1+INT ((d-128)/8));" ";
410 LET x=d-8*INT (d/8): GO SUB 5000
420 PRINT #DEV: GO TO 200
500 LET d=d+1: FOR i=1 TO 9
```



```

510 IF A$(d,i)="n" THEN LET n=PEEK (PC): GO SUB 3000: PRINT #DEV;n$:: LET
    PC=PC+1: GO TO 600
520 IF A$(d,i)="m" THEN LET n=PEEK (PC)+256*PEEK (PC+1): GO SUB 3000: PRINT
    #DEV;n$:: LET PC=PC+2: GO TO 600
530 IF A$(d,i)="$" THEN PRINT #DEV:"SP": GO TO 600
540 IF A$(d,i)<>"d" THEN GO TO 570
550 LET PC=PC+1: LET x=PEEK (PC-1): GO SUB 3500: LET n=PC+x
560 GO SUB 3000: PRINT #DEV;n$:: GO TO 600
570 IF A$(d,i)<>"#" THEN PRINT #DEV:A$(d,i):: GO TO 600
580 IF A$(d,i-1)<>"(" OR d=106 THEN GO TO 595
590 IF h$<>"HL" THEN LET x=PEEK (PC): GO SUB 3500: LET n=ABS (x):
    LET PC=PC+1: GO SUB 3000: PRINT #DEV:h$:CHR$ (44-SGN (x+0.5)):n$::
    GO TO 600
595 PRINT #DEV:h$:
600 NEXT i: PRINT #DEV: GO TO 200
700 LET d=PEEK (PC): LET PC=PC+1
702 IF h$<>"HL" THEN LET d=PEEK (PC): LET PC=PC+1
710 IF d>=64 THEN GO TO 800
720 PRINT #DEV:C$(1+INT (d/8)):" ":
725 IF h$<>"HL" THEN LET PC=PC-2
730 LET x=d-8*INT (d/8): GO SUB 5000
735 IF h$<>"HL" THEN LET PC=PC+1
740 PRINT #DEV: GO TO 200
800 PRINT #DEV:B$(INT (d/64)):" ":
810 PRINT #DEV:INT (d/8)-8*INT (INT (d/8)/8):".":
820 LET x=d-8*INT (d/8)
825 IF h$<>"HL" THEN LET PC=PC-2
830 GO SUB 5000: GO TO 735
900 LET d=PEEK (PC): LET PC=PC+1: IF d<64 THEN PRINT #DEV:"?": GO TO 200
910 IF d<128 THEN LET d=d+64: GO TO 500
920 IF d<160 THEN PRINT #DEV:"?": GO TO 200
930 IF d<192 THEN LET d=d+32: GO TO 500
940 PRINT #DEV:"?": GO TO 200
1000 STOP
3000 REM conversie dec->hex
3010 LET n$="": IF n=0 THEN LET n$="0": RETURN
3030 LET l1=INT (LN n/2.7725887): LET hh=16^l1
3040 FOR q=1 TO l1+1: LET o=INT (n/hh)
3050 LET n$=n$+q$(o+1): LET n=n-hh*o: LET hh=hh/16
3060 NEXT q: IF LEN n$=1 OR LEN n$=3 THEN LET n$="0"+n$
3080 RETURN
3100 REM conversie hex->dec
3110 LET l1=LEN n$: LET hh=16^(l1-1): LET n=0
3120 FOR q=1 TO l1
3130 LET n=n+hh*(CODE n$(q)-48-(7 AND n$(q)>"9")): LET hh=hh/16
3140 NEXT q: RETURN
3500 IF x>129 THEN LET x=-256+x
3510 RETURN
5000 IF x=0 THEN PRINT #DEV:"B":
5010 IF x=1 THEN PRINT #DEV:"C":
5020 IF x=2 THEN PRINT #DEV:"D":
5030 IF x=3 THEN PRINT #DEV:"E":
5040 IF x=4 THEN PRINT #DEV:"H":
5050 IF x=5 THEN PRINT #DEV:"L":
5060 IF x=7 THEN PRINT #DEV:"A":
5070 IF x<>6 THEN RETURN
5080 PRINT #DEV:"(":"h$:
5090 IF h$<>"HL" THEN LET x=PEEK (PC): GO SUB 3500: LET n=ABS (x): LET PC=PC+1:
    GO SUB 3000: PRINT #DEV:CHR$ (44-SGN (x+0.5)):n$:
5100 PRINT #DEV:")": RETURN
9000 DATA "NOP","LD BC,m","LD (BC),A","INC BC","INC B","DEC B",
    "LD B,n","RLCA","EX AF,AF',"ADD #.BC","LD A,(BC)","DEC BC","INC C",
    "DEC C","LD C,n","RRCA"

```



```

9010 DATA "DJNZ d","LD DE,m","LD (DE),A","INC DE","INC D","DEC D","LD
D,n","RLA","JR d","ADD #,DE","LD A,(DE)","DEC DE","INC E","DEC E","LD
E,n","RRA"
9020 DATA "JR NZ,d","LD #,m","LD (m),#","INC #","INC H","DEC H","LD
H,n","DAA","JR Z,d","ADD #,#","LD #,(m)","DEC #","INC L","DEC L","LD
L,n","CPL"
9030 DATA "JR NC,d","LD $,m","LD (m),A","INC $","INC (#)","DEC (#)","LD
(#),n","SCF","JR C,d","ADD #,$","LD A,(m)","DEC $","INC A","DEC A","LD
A,n","CCF"
9040 DATA "RET NZ","POP BC","JP NZ,m","JP m","CALL NZ,m","PUSH BC","ADD A,n",
"RST 0","RET Z","RET","JP Z,m","?","CALL Z,m","CALL m","ADC A,n",
"RST 8"
9050 DATA "RET NC","POP DE","JP NC,m","OUT (n),A","CALL NC,m","PUSH DE","SUB
n","RST 10","RET C","EXX","JP C,m","IN A,(n)","CALL C,m","?","SBC A,n","RST
18"
9060 DATA "RET PO","POP #","JP PO,m","EX ($),#","CALL PO,m","PUSH #","AND n",
"RST 20","RET PE","JP (#)","JP PE,m","EX DE,#","CALL PE,m","?","XOR n",
"RST 28"
9070 DATA "RET P","POP AF","JP P,m","DI","CALL P,m","PUSH AF","OR n","RST
30","RET M","LD $,#","JP M,m","EI","CALL M,m","?","CP n","RST 38"
9080 DATA "IN B,(C)","OUT (C),B","SBC #,BC","LD (m),BC","NEG","RETn","IM 0","LD
I,A","IN C,(C)","OUT (C),C","ADC #,BC","LD BC,(m)","?","RETI","?","LD
R,A"
9090 DATA "IN D,(C)","OUT (C),D","SBC #,DE","LD (m),DE","?","?","IM 1","LD
A,I","IN E,(C)","OUT (C),E","ADC #,DE","LD DE,(m)","?","?","IM 2",
"LD A,R"
9100 DATA "IN H,(C)","OUT (C),H","SBC #,#","LD (m),#","?","?","?","RRD","IN
L,(C)","OUT (C),L","ADC #,#","LD #,(m)","?","?","?","RLD"
9110 DATA "IN F,(C)","?","SBC #,$","LD (m),$","?","?","?","?","IN A,(C)","OUT
(C),A","ADC #,$","LD $,(m)","?","?","?","?"
9120 DATA "LDI","CPI","INI","OUTI","?","?","?","?","LDD","CPD","IND",
"OUTD","?","?","?","?"
9130 DATA "LDIR","CPIR","INIR","OTIR","?","?","?","?","LDDR","CPDR",
"INDR","OTDR","?","?","?"
9200 DATA "ADD","ADC","SUB","SBC","AND","XOR","OR","CP"
9210 DATA "RLC","RRC","RL","RR","SLA","SRA","s!s","SRL"
9220 DATA "BIT","RES","SET"
9999 CLEAR : SAVE "OP DECODER" LINE 10

```

4.3. GRAFICE DE FUNCȚII

Versaplot este o subrutină scrisă în Basic pentru reprezentarea grafică cu autoscalare a funcțiilor de o variabilă $y = y(x)$, cunoscându-se domeniul de definiție al funcției, de la $\liminf$ la $\limsup$. Programul realizează două treceri pe intervalul dat. La prima trecere se caută extremele funcției, pentru calcularea factorului de scalare pe ordonată. În a doua trecere se desenează graficul funcției și o grilă ajutătoare cu linii punctate, pentru estimarea valorilor funcției în diverse puncte. Intervalul între două diviziuni ale grilei este ales automat pentru o reprezentare cât mai convenabilă.

Dacă funcția are singularități în intervalul $[\liminf, \limsup]$, ne putem aștepta la erori de tipul Invalid argument, Number too big sau la o reprezentare total deformată.

Pentru a include subrutina în alte programe, se vor face câteva adaptări:

– se șterg liniile 1. 9. 10. 15. 20 și 30;

- se verifică dacă programul principal nu conține linia 0 și linii peste 9499, altfel trebuie mutate;
- se face MERGE cu programul apelant peste Versaplot;
- se schimbă instrucțiunea LET y=VAL f\$ din liniile 9506 și 9522 cu instrucțiunea sau setul de instrucțiuni prin care se poate calcula valoarea funcției y pentru un x dat; de exemplu, dacă în programul apelant există o subrutină eval care realizează acest lucru cu rezultatul în variabila f, se va scrie: GO SUB eval: LET y=f;
- se introduce în programul principal apelul la subrutina grafică prin GO SUB versaplot sau GO SUB 9500.

Exemplu

Vom reprezenta grafic funcția:

$$F(x) = e^{-2\sin(x)}, \quad x = [-\pi, 3\pi]$$

Pentru aceasta, vom rula programul cu următoarele date:

F(x)= EXP (-2*SIN x)

lim inf= -PI

lim sup= 3*PI

Listinul programului

```

1> DEF FN l(i,s)=10^INT (LN ABS (s-i)/2.3025851): LET versaplot=9500: REM
VERSAPLOT 1.0 PLETER 1986
9 INK 0: PAPER 7: CLS : BORDER 7
10 INPUT "F(x)= ": LINE f$:"lim inf= ":li:"lim sup= ":ls
15 IF PEEK 64009=195 THEN RANDOMIZE USR 64000: LPRINT CHR$ 1:"BMS versaplot":
LPRINT "Funcția y(x) = ":f$:"x min= ":li:"x max= ":ls:"
20 GO SUB 9500
30 GO TO 10
9500 REM versaplot IN li,ls,f$ LOCAL x,y,q,j,min,max,ll,xx,yy,q(),r(),dx,dy
9502 CLS : BORDER 4: INPUT :: PRINT #0:" VERSAPLOT":TAB 32:
9503 PLOT 3.80: DRAW 249.0: DRAW 0.24: DRAW -249.0: DRAW 0.-24:
PRINT AT 13.0:"0%":AT 13.29:"50%":AT 10.7:"TIMPUL DE EXECUTIE"
9504 LET max=-5.9e-37: LET min=1e+38: FOR q=4 TO 251:
LET x=(q-4)/248*(ls-li)+li
9506 LET y=VAL f$: PLOT q,81: DRAW OVER 1:0.22
9508 IF y>max THEN LET max=y
9510 IF y<min THEN LET min=y
9512 NEXT q
9514 FOR q=1 TO 13: PLOT INVERSE 1:0,q+79: DRAW INVERSE 1:255.0: PLOT INVERSE
1:0,105-q: DRAW INVERSE 1:255.0: NEXT q
9515 PRINT AT 13.0:" ":AT 13.29:"
9518 IF max=min THEN PLOT 4,86: DRAW 248.0: PRINT AT 15.0:"y=":y: GO TO 9600
9520 FOR q=4 TO 251: LET x=(q-4)/248*(ls-li)+li
9522 LET y=VAL f$
9524 PLOT q,(y-min)/(max-min)*168+4
9526 NEXT q
9527 IF max=min THEN PLOT 4,86: DRAW 248.0: PRINT AT 15.0:"y=":y: GO TO 9600

```



```

9528 LET dx=FN l(li,ls): LET dy=FN l(min,max)
9530 DIM q(11): DIM r(11): LET vpc=1: FOR x=SGN li*INT (ABS li/dx)*dx TO ls STEP
dx
9531 LET xx=(x-li)/(ls-li)*248+4: IF xx<0 OR xx>255 THEN NEXT x
9532 PLOT xx,4: IF NOT x THEN DRAW 0.168: GO TO 9534
9533 FOR q=4 TO 172 STEP 2: PLOT xx,q: NEXT q
9534 LET q(vpc)=INT (xx/8): LET r(vpc)=x/dx*NOT (ABS (x/dx)<1e-4): IF q(vpc)>29
THEN LET q(vpc)=q(vpc)-LEN STR$ r(vpc)+1
9535 PRINT AT 21,q(vpc):r(vpc): LET vpc=vpc+1: NEXT x
9536 FOR y=SGN min*INT (ABS min/dy)*dy TO max STEP dy
9537 LET yy=(y-min)/(max-min)*168+4: IF yy<0 OR yy>175 THEN NEXT y
9538 PLOT 4,yy: IF NOT y THEN DRAW 248.0: GO TO 9540
9539 FOR q=4 TO 252 STEP 2: PLOT q,yy: NEXT q
9540 LET ll=INT ((175-yy)/8): PRINT AT ll-(1 AND ll=21).0:y/dy*NOT (ABS
(y/dy)<1e-4): NEXT y
9541 FOR q=1 TO vpc-1: PRINT AT 21,q(q):r(q): NEXT q
9542 INPUT :: PRINT #0:"dx=":dx,"dy=":dy
9544 IF INKEY$<>" " THEN GO TO 9544
9546 IF INKEY$="" THEN GO TO 9546
9548 IF INKEY$="z" AND PEEK 64009=195 THEN RANDOMIZE USR 64009: GO TO 9544
9600 BORDER 7: INPUT :: RETURN
9999 CLEAR : SAVE "versaplot" LINE 1

```

4.4. PROGRAM DE REPREZENTĂRI GRAFICE 3D

Programul este scris integral în Basic și este compatibil cu Interfața 1 și Microdrive, precum și cu Timext. Cu modificări minore la instrucțiunile LOAD * și SAVE *, poate fi adaptat pentru HC-90 cu unitate floppy. Fundamentul teoretic al programului îl constituie lucrarea [43].

4.4.1. TERMENI SPECIFICI PENTRU GRAFICĂ

2D – bidimensional.

3D – tridimensional.

Axonometrie – reprezentarea imaginii după o anumită direcție a obiectelor tridimensionale în plan.

Bidimensional – spațiu în care fiecare punct este determinat de două coordonate; planul fizic.

Coordonate carteziene – poziția unui punct în spațiu definită într-un sistem de coordonate cu trei axe perpendiculare două câte două: X, Y și Z. Fiecărui punct îi corespunde un triplet de numere reale, distanțele cu semn ale punctului față de origine, pe fiecare axă.

Coordonate cilindrice – poziția unui punct în spațiu definită într-un sistem de coordonate prin trei numere: distanța față de axa verticală care trece prin origine (raza), unghiul de azimut față de o referință de direcție și înălțimea față de planul orizontal care trece prin origine, notate: R, A, H.

Elevare (termen cu sens specific în această lucrare) – procedeu de obținere a unui obiect tridimensional prin translația perpendiculară pe plan a unui contur plan, însoțită eventual de mărire/micșorări, translații în plan sau

rotații în plan ale conturului; de exemplu, un trunchi de piramidă dreaptă cu baza pătrat, determinat de înălțimea H , latura bazei L și latura bazei mici l , se obține prin elevarea unui pătrat (baza) de latură L trasat în plan orizontal, pe distanța H , însoțită de micșorarea conturului cu un factor de l/L .

Generatoare — dreapta care mișcându-se după o anumită lege descrie o suprafață; familie de drepte care descriu o suprafață; (specific) segmente de dreaptă prin care se materializează suprafața laterală a unui cilindru sau con.

Hidden, ascuns — reprezentare derivată din *wireframe* în care corpul devine opac, adică sunt șterse muchiile "ascunse" în spatele unor suprafețe ale obiectului.

Panorama — desfășurata proiecției pe tablou cilindric.

Panorama sferică — desfășurata proiecției pe tablou sferic.

Perspectiva — reprezentarea bidimensională a obiectelor tridimensionale, așa cum se văd dintr-un anumit punct.

Perspectiva ascendentă — perspectivă în care punctul de vedere se găsește dedesubtul subiectului; vedere oblică de jos în sus.

Perspectiva descendentă — perspectivă în care punctul de vedere se găsește deasupra subiectului; vedere oblică de sus.

Perspectiva orizontală — perspectivă în care punctul de vedere se găsește chiar deasupra sau dedesubtul subiectului; vedere de sus sau de jos.

Perspectiva verticală — perspectivă în care punctul de vedere se găsește pe același nivel cu subiectul; vedere laterală; este singurul caz în care, în reprezentare, liniile verticale sunt paralele.

Plan lateral — plan perpendicular pe axa Oy ($y = \text{constant}$).

Plan orizontal — plan perpendicular pe axa Oz ($z = \text{constant}$).

Plan vertical — plan perpendicular pe axa Ox ($x = \text{constant}$).

Plotter — periferic de calculator capabil să deseneze cu ajutorul unui cap de trasare mobil în plan față de suprafața de desen.

Proiecție — metodă de a reprezenta un corp tridimensional pe un plan numit tablou de proiecție. Axonometria este o proiecție paralelă, iar perspectiva, o proiecție centrală.

Prompter — simbol sau succesiune de simboluri sugestive, prin care programul cere utilizatorului o dată, un parametru, o instrucțiune sau o comandă.

Punct de vedere — punct în care se consideră amplasat privitorul la calculul perspectivei.

Scalare — modificarea scării unui desen, adică mărirea sau micșorarea desenului, cu păstrarea proporțiilor.

Subiect — obiect sau grup de obiecte aflate în cadrul imaginii.

Tridimensional — spațiul fizic, în care fiecare punct este determinat de trei coordonate.

Wireframe — reprezentare a obiectelor în plan tipică pentru calculatoare, caracterizată de aproximarea obiectului cu o rețea de sârmă. Obiectul în reprezentare *wireframe* este complet transparent.

4.4.2. INSTRUCȚIUNI GRAFICE

Primul segment al programului (liniile 1000–5000) este dedicat construirii obiectelor. În această scop se folosește un limbaj specializat de descriere a corpurilor și de desen în 3 dimensiuni. Segmentul poate fi considerat un editor de obiecte în spațiu.

Orice corp trebuie descris în unul din următoarele moduri (sau o combinație a lor):

- desen în spațiu din puncte și linii (instrucțiunile PLOT x,y,z și DRAW x,y,z);

- desen într-un plan vertical, orizontal sau lateral (instrucțiunile FIGURE, CIRCLE);

- elevarea unui desen generat de FIGURE sau CIRCLE, eventual însoțită de translații în plan, rotații în plan sau mărimi-micșorări (instrucțiunea ELEVATION);

- elevarea unui desen generat de FIGURE sau CIRCLE într-un punct (instrucțiunea CONE);

- paralelipiped (instrucțiunea PRISM).

După încărcare, programul afișează un *prompter* READY 1 pe linia de jos a ecranului. Numărul din dreapta jos (1) reprezintă numărul de puncte generate în fișierul obiect, majorat cu 1 (deci deocamdată 0 puncte generate). În versiunea curentă, numărul nu poate depăși 700 de puncte generate. Instrucțiunile și comenzile sunt invocate prin apăsarea unor taste (în continuare, tastele apar între paranteze drepte). Înainte de a apăsa o tastă, utilizatorul trebuie să aștepte apariția lui READY în colțul din stânga jos al ecranului. Sintaxa instrucțiunilor nu trebuie memorată, fiecare parametru fiind cerut de program printr-un *prompter* sugestiv. În cele ce urmează, prezentăm lista instrucțiunilor primului segment al programului.

[Q] PLOT $\langle x,y,z \rangle$ — (punct) instrucțiune de generare a unui punct în spațiu, de coordonate x, y, z . Coordonatele sunt cerute de $X=$, $Y=$ și $Z=$.

[W] DRAW $\langle x,y,z \rangle$ — (linie) instrucțiune de generare a unui segment de dreaptă în spațiu, pornind din punctul definit anterior cu PLOT sau DRAW, până în punctul de coordonate x, y, z . Coordonatele sunt cerute de $X=$, $Y=$ și $Z=$. Dacă instrucțiunea precedentă nu este PLOT sau DRAW, apare mesajul de eroare STARTPOINT MISSING (lipsește punctul de plecare).

[F] FIGURE $\langle \text{plan}, \text{coordonate} \rangle$ — (figură, contur) instrucțiune de generare a unui contur închis într-un plan orizontal, vertical sau lateral. La SET se răspunde cu una din literele X, Y sau Z astfel:

SET X — fixează coordonata x — contur în plan vertical

SET Y — fixează coordonata y — contur în plan lateral

SET Z — fixează coordonata z — contur în plan orizontal

La AT se răspunde cu coordonata planului respectiv (de pildă: SET Z AT 0 este un plan orizontal de cotă 0; SET Y AT 10 este un plan lateral cu ecuația $y=10$). În continuare, programul cere coordonatele primului punct al conturului în planul ales. De exemplu, dacă am fixat planul x, atunci se cer coordonatele $Y=$ și $Z=$. După fiecare punct introdus, apare *prompter*-ul Next / Home, adică utilizatorul poate introduce un nou punct apăsând [N], sau poate închide conturul apăsând [H].

[C] CIRCLE <plan.centru.raza.nr.puncte.unghi> — (cerc, poligon) instrucțiune de generare a unui poligon regulat într-un plan orizontal, vertical sau lateral. La SET se răspunde ca la comanda FIGURE. În continuare, se cer coordonatele centrului poligonului în planul ales. De exemplu, dacă am fixat planul y, atunci se cer coordonatele $XC=$ și $ZC=$. Prin $R=$ se introduce raza poligonului (mai corect spus, raza cercului în care este înscris poligonul). Numărul de puncte se introduce prin $N=$ și poate fi:

- $N=3$ — triunghi echilateral
- $N=4$ — pătrat
- $N=5$ — pentagon
- $N=6$ — hexagon
- $N=7$ — heptagon (aproximație de cerc pentru raze mici)
- $N=8$ — octogon (— " —)
- $N>8$ — cerc (aproximativ)

Ultimul parametru, cerut prin $ANGLE=$, este unghiul în grade făcut de direcția formată de centrul poligonului și primul vârf al acestuia cu ordonata planului. Acest parametru permite orientarea în direcția dorită a poligonului în plan. De exemplu, un triunghi echilateral în planul vertical, cu unghiul 0, va avea vârful spre dreapta, iar cu unghiul 90, va avea vârful în sus.

[P] PRISM <coordonate.coordonate> — (prismă, paralelipiped) instrucțiune de generare a unui paralelipiped sau prismă dreptunghiulară dreaptă, cu fețele paralele cu planurile vertical, orizontal și respectiv lateral. Utilizatorul trebuie să specifice coordonatele x, y și z a două colțuri opuse pe o diagonală mare. *Prompter*-ii sunt următorii: FROM $X=$, FROM $Y=$, FROM $Z=$ pentru primul colț și TO $X=$, TO $Y=$, TO $Z=$ pentru celălalt colț. De exemplu, un paralelipiped de coordonate: (0,0,0), (0,1,0), (1,1,0), (1,0,0), (0,0,2), (0,1,2), (1,1,2), (1,0,2) poate fi descris prin oricare din următoarele instrucțiuni:

```
PRISM FROM 0.0.0 TO 1.1.2
PRISM FROM 0.0.2 TO 1.1.0
PRISM FROM 0.1.0 TO 1.0.2
PRISM FROM 1.0.0 TO 0.1.2
```

în funcție de care anume colțuri au fost alese.

[O] CONE <coordonate> — (con, piramidă) instrucțiune de generare a unei piramide având ca bază conturul generat anterior (cu FIGURE sau cu CIRCLE). Utilizatorul trebuie să specifice coordonatele vârfului piramidei (conului) prin TOP X=, TOP Y= și TOP Z=. În fapt, se unesc toate punctele de pe conturul definit anterior, cu vârful piramidei (conului). Dacă instrucțiunea imediat anterioară nu este FIGURE, CIRCLE sau ELEVATION, instrucțiunea CONE generează un mesaj de eroare: BASE NOT DEFINED (baza nu a fost definită).

[E] ELEVATION <înălțime, translație, amplificare, răsucire> — (elevare) translația conturului cel mai recent definit (cu FIGURE, CIRCLE sau ELEVATION), însoțită eventual de amplificare și răsucire. Scopul translației este generarea unui corp în spațiu, cu baza specificată prin FIGURE sau CIRCLE. Translația principală se face de-a lungul axei perpendiculare pe planul bazei, pe distanța h. De aceea, primul parametru cerut de program este H=. Dacă h este negativ, corpul va fi generat *sub* planul bazei, altfel el va fi generat *deasupra*¹ acestuia. În continuare, se cer distanțele pe care se face translația pe celelalte două axe. De exemplu, dacă baza este în plan orizontal, se cer TRANSLATION X= și TRANSLATION Y=. Dacă aceste două translații sunt zero, atunci corpul generat va fi drept, altfel va fi oblic. Dacă baza este generată cu CIRCLE, mai urmează două întrebări: AMPLIFY *, adică factorul de amplificare a poligonului de bază (se va răspunde cu "1" pentru aceeași mărime, cu un număr subunitar pentru micșorare, sau supraunitar pentru mărire), și TWIST ANGLE=, adică unghiul cu care poligonul elevat este rotit față de poligonul de bază (0 înseamnă fără rotire). Dacă baza este generată de FIGURE, atunci nu are sens amplificarea și rotirea, deoarece aceste transformări necesită un centru pe care doar poligoanele regulate îl au în mod natural. În cele ce urmează, vom exemplifica generarea unor corpuri prin instrucțiunea ELEVATION:

a) trunchi de piramidă cu baza hexagon, latura bazei 5, înălțimea 10, latura bazei mici 2, baza mare în planul orizontal de cotă 0.

```
CIRCLE FIX Z, AT 0, XC=0, YC=0, R=5, N=6, ANGLE=0
ELEVATION H=10, TR. X=0, TR. Y=0, AMPLIFY * 2/5, TWIST 0
```

b) cilindru drept cu raza 7 și înălțimea 20, cu baza în planul vertical x = -8.

```
CIRCLE FIX X, AT -8, XC=0, YC=0, R=7, N=12, ANGLE=0
ELEVATION H=20, TR. Y=0, TR. Z=0, AMPLIFY * 1, TWIST 0
```

c) hiperboloid drept cu raza 7 și înălțimea 20, obținut prin rotirea bazei cu 60°, cu baza în planul orizontal z=0.

¹sub și deasupra în cazul unei baze în plan orizontal se vor generaliza prin *invers* și *direct* față de sensul axelor de coordonate, pentru celelalte planuri.

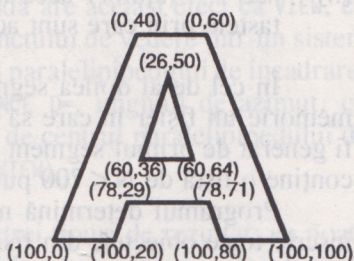
CIRCLE FIX Z, AT 0, XC=0, YC=0, R=7, N=12, ANGLE=0
ELEVATION H=20, TR. X=0, TR. Y=0, AMPLIFY * 1, TWIST 60

d) prismă oblică cu baza triunghi, cu colțurile de coordonate: (1,1), (1,4), (5,4), înclinată cu 30° peste muchia (1,4) – (5,4), baza orizontală, înălțimea 60.

FIGURE FIX Z, AT 0, X=1, Y=1, NEXT, X=1, Y=4, NEXT, X=5, Y=4, HOME
ELEVATION H=60, TR. X=60*TAN(30*PI/180), TR. Y=0

e) litera "A" în relief, în planul vertical:
întâi se proiectează litera și se scriu coordonatele colțurilor (scara este arbitrară, la latitudinea utilizatorului):

FIGURE FIX X, AT 0, Y=0, Z=60, NEXT, Y=0, Z=40, NEXT, Y=100, Z=0, NEXT, Y=100, Z=20, NEXT, Y=78, Z=29, NEXT, Y=78, Z=71, NEXT, Y=100, Z=80, NEXT, Y=100, Z=100, HOME
ELEVATION H=10, TR. Y=0, TR. Z=0



Dacă instrucțiunea anterioară nu a fost CIRCLE, FIGURE sau ELEVATION, instrucțiunea ELEVATION nu are sens și se generează mesajul de eroare ELEVATION OF WHAT? (ce se elevează?).

[CAP-0] DELETE — șterge cel mai recent definit punct, suprafață sau corp (efectul instrucțiunii precedente).

[A] NEW — (nou) începe asamblarea unei noi imagini. Această instrucțiune șterge fișierul existent în memorie, cu o confirmare.

[S] SAVE & VERIFY <fișier> — salvează și verifică fișierul generat sub numele specificat de utilizator, pe bandă sau *microdrive*-ul specificat.

[L] LOAD <fișier> — încarcă un fișier salvat anterior pe casetă sau pe *microdrive*.

[K] LIST — listează coordonatele generate sub formă de tabel:

LIST	X	Y	Z	T
1:	0	0	0	*
2:	162.42	-3.5	6	

Coloana T necesită explicații suplimentare. Aici figurează fie semnul "*", fie semnul "|". Cei care cunosc funcționarea unui *plotter*, vor observa că "*" înseamnă "cu penița sus", iar "|" înseamnă "cu penița jos". Cu alte cuvinte, punctul marcat cu "*" nu va fi unit cu o linie de punctul

precedent, în timp ce punctul marcat cu "I" este unit cu o linie de punctul precedent. Reprezentarea internă a datelor simulează de fapt un *plotter* virtual în 3D. Pentru întreruperea listării, se apasă [BREAK], după care se revine în program cu GO TO cont.

[SS-A] STOP — închide fișierul generat și transferă controlul secțiunii a doua a programului.

[H] HELP — (ajutor) această comandă listează toate comenzile disponibile, tastele prin care sunt accesate și un rezumat al funcțiilor lor.

În cel de-al doilea segment al programului se poate intra dacă există în memorie un fișier în care să fie descris cel puțin un obiect. Acest fișier poate fi generat de primul segment sau încărcat de pe casetă sau *microdrive*. Fișierul conține o listă de $N < 700$ puncte (x,y,z,t).

Programul determină mai întâi cel mai mic paralelipiped în care se pot înscrie toate obiectele din fișier. Apoi se înscriu în vectorul C(3) coordonatele centrului geometric al paralelipipedului. După specificarea unui punct de vedere cu instrucțiunile VIEW sau CYLINDRICAL, programul alege tabloul optim de perspectivă, adică perpendicular pe axa care unește punctul de vedere cu centrul imaginii, la jumătatea distanței dintre cele două puncte. Dacă planul tabloului intersectează paralelipipedul de încadrare, înseamnă că punctul de vedere este prea apropiat de obiect, rezultând un efect exagerat de perspectivă, anunțat printr-un mesaj de eroare ERROR: TOO CLOSE (prea aproape).

Programul anunță tipul perspectivei: verticală, orizontală, descendentă sau ascendentă. Tipul perspectivei este important pentru că programul calculează altfel în fiecare caz.

Imaginea este scalată automat la scara cerută de utilizator (vezi instrucțiunea RESCALE) și încadrată automat pe orizontală și pe verticală, dar păstrând proporția între axe. Imaginea obținută poate fi salvată pe casetă (SAVE), rescalată (RESCALE), iar fișierul care conține lista punctelor de coordonate x, y poate fi listat pe ecran (LIST) sau poate fi salvat pe *microdrive*. Instrucțiunile din segmentul al doilea al programului sunt următoarele:

[V] VIEW <coordonate> — (vedere) prin această instrucțiune se specifică un punct de vedere de coordonate x, y, z și se cere desenarea perspectivei din acel punct. Coordonatele sunt introduse de X=, Y=, Z=. Punctul de vedere trebuie să se găsească la o oarecare distanță de obiect, altfel apare eroarea ERROR: TOO CLOSE. Pentru a obține anumite tipuri particulare de perspectivă (orizontală și verticală), se poate răspunde pentru anumite coordonate, cu cele ale centrului paralelipipedului de încadrare, astfel:

VIEW X=1000, Y=C(Y), Z=C(Z)

VIEW X=C(X), Y=C(Y), Z=1000

VIEW X=C(X), Y=C(Y), Z=-800

VIEW X=C(X), Y=2000, Z=C(Z)

perspectivă verticală frontală

perspectivă orizontală de sus

perspectivă orizontală de jos

perspectivă verticală laterală

Numerele 1000, -800, 2000 sunt arbitrare, ele exprimând distanța până la obiect. Cu cât distanța este mai mare, cu atât efectul de perspectivă este mai redus. La limită, pentru distanțe foarte mari, proiecția devine paralelă, astfel că programul poate face în aceste condiții și reprezentări axonometrice. După terminarea calculelor, care pot dura între 8 secunde și 10 minute, în funcție de numărul de puncte, între 2 și 700, programul execută automat instrucțiunea PICTURE.

[C] CYLINDRICAL <coordonate> — această comandă are același efect ca VIEW, cu excepția faptului că se cer coordonatele punctului de vedere într-un sistem cilindric cu originea în centrul geometric al paralelipipedului de încadrare. Se cer: distanța până la obiect, cu DISTANCE D=, unghiul de azimut, cu AZIMUTH [DEG]= (în grade), și înălțimea față de centrul paralelipipedului de încadrare, cu HEIGHT H= (vezi și comanda VIEW).

[P] PICTURE — (imagine) această comandă are trei tipuri de rezultat: ea poate produce un desen pe ecran, o listă de coordonate pe ecran, sau o listă de coordonate într-un fișier pe *microdrive*. La început, utilizatorul trebuie să aleagă între ecran și *microdrive*, prin *prompter*-ul SCREEN (0) MDRIVE (1-8). Fișierul pe *microdrive* este util pentru desenul la imprimanta grafică, cu programul de la § 4.5. Înainte de a salva fișierul, va trebui să rescalăți imaginea la 0,511/0,767. În cazul ecranului, acesta este șters și este desenată perspectiva calculată în cadrul specificat de comanda RESCALE (în lipsa acestei specificații, pe întreg ecranul). Dacă imaginea este rescalată în afara limitelor ecranului, atunci în loc de desen, pe ecran apare lista coordonatelor punctelor (x,y), precum și semnul "*" sau "|" (vezi comanda LIST).

[R] RESCALE — (rescalare) această comandă permite specificarea cadrului în care se face trasarea perspectivei. În mod normal, acest cadru este întreg ecranul (cu excepția unei borduri de 2 pixeli lățime), adică între coordonatele 2 și 253 pe orizontală și 2 și 173 pe verticală. La *prompter*-ul Default/Rescale, pentru rescalare, apăsați [R] iar pentru revenirea la scara inițială (2,253/2,173) apăsați [D]. În cazul în care s-a optat pentru rescalare, vor trebui specificate coordonatele noului cadru, cu XMIN=, XMAX=, YMIN=, YMAX=. În final, la RESCALE CURRENT IMAGE? (Y/N), se răspunde cu [Y] pentru rescalarea imaginii curente, sau altfel cu [N]. În caz afirmativ, după calcule, programul invocă automat comanda PICTURE. Cadrul specificat poate fi oricât de mare, în afara ecranului, caz semnalizat printr-un avertisment WARNING: OUT OF SCREEN. Din exemplele următoare, rezultă utilitatea comenzii RESCALE, pentru:

a) desenarea la un *plotter* cu suprafața de coordonate 0÷1000 pe orizontală, și 0÷700 pe verticală:

RESCALE R XMIN=0, XMAX=1000, YMIN=0, YMAX=700
WARNING: OUT OF SCREEN

b) desenarea manuală pe o planșetă Faber-Castell A3, având coordonatele extreme: $0 \div 420$ pe orizontală și $-135 \div 135$ pe verticală:

RESCALE R XMIN=0, XMAX=420, YMIN=-135, YMAX=135
WARNING: OUT OF SCREEN

c) desenarea pe ecran, în colțul din dreapta sus:

RESCALE R XMIN=128, XMAX=253, YMIN=88, YMAX=173

[K] LIST — listează pe ecran coordonatele (x,y) ale perspectivei calculate, într-un tabel de forma:

LIST	X	Y	T
1:	0.00	0.00	*
2:	162.43	-3.50	

Pentru semnificațiile coloanei T, vezi comanda LIST din primul segment al programului. Coordonatele sunt tipărite în format fix, cu două zecimale.

[D] DETAILS — (detalii) comanda afișează pe ecran următorii parametri interni ai calculului perspectivei: coordonatele centrului (paralelipipedului de încadrare), parametrii planului pe care se face proiecția (a, b, c) și cadrul curent de scalare (SCALE: xmin,xmax/ymin,ymax).

[Y] RETURN — întoarcere în segmentul de editare pentru adăugarea de noi elemente fișierului imagine, pentru crearea unui nou fișier sau pentru operații de încărcare/salvare.

[S] SAVE <fișier> — salvează fișierul ecran pe casetă sau pe *microdrive*. Se cere dispozitivul și numele fișierului.

4.4.3. APLICAȚII ALE PROGRAMULUI

4.4.3.1. Aplicație în prezentarea datelor

Să presupunem că dorim o histogramă 3D pentru următoarele date statistice:

1987 - 100%

1988 - 78%

1989 - 61%

Vom defini cele trei valori statistice ca prisme cu baza pătrat de latură 0.5, în planul de cotă 0, aliniate la axa $x=0$, cu înălțimea proporțională cu procentul respectiv. Se dau următoarele instrucțiuni de generare:

```
PRISM FROM 0.1987.0: TO 0.5.1987.5.10.0
PRISM FROM 0.1988.0: TO 0.5.1988.5.7.8
PRISM FROM 0.1989.0: TO 0.5.1989.5.6.1
STOP
VIEW X=5, Y=1990, Z=15
```

4.4.3.2. Aplicație în arhitectură

Să presupunem că dorim să vedem din diferite unghiuri un templu cu patru coloane. Coloanele sunt așezate în colțurile unui dreptunghi, de coordonate: (10,10), (20,10), (20,30), (10,30). Coloanele au înălțimea de 29 m (inclusiv capitelul de 4 m). Acoperișul templului are secțiunea de coordonate (5,29), (25,29), (25,32), (15,36), (5,32). Templul se construiește în felul următor:

-- prima coloană -- conturul bazei

```
CIRCLE SET Z=0, XC=10, YC=10, R=2, N=8, ANGLE=0
```

-- coloana propriu-zisă

```
ELEVATE H=25, TR. X=0, TR. Y=0, AMPLIFY * 1, TWIST 0
```

-- capitelul

```
ELEVATE H=4, TR. X=0, TR. Y=0, AMPLIFY * 2, TWIST 0
```

-- a doua coloană

```
CIRCLE SET Z=0, XC=20, YC=10, R=2, N=8, ANGLE=0
```

```
ELEVATE H=25, TR. X=0, TR. Y=0, AMPLIFY * 1, TWIST 0
```

```
ELEVATE H=4, TR. X=0, TR. Y=0, AMPLIFY * 2, TWIST 0
```

-- a treia coloană

```
CIRCLE SET Z=0, XC=20, YC=30, R=2, N=8, ANGLE=0
```

```
ELEVATE H=25, TR. X=0, TR. Y=0, AMPLIFY * 1, TWIST 0
```

```
ELEVATE H=4, TR. X=0, TR. Y=0, AMPLIFY * 2, TWIST 0
```

-- a patra coloană

```
CIRCLE SET Z=0, XC=10, YC=30, R=2, N=8, ANGLE=0
```

```
ELEVATE H=25, TR. X=0, TR. Y=0, AMPLIFY * 1, TWIST 0
```

```
ELEVATE H=4, TR. X=0, TR. Y=0, AMPLIFY * 2, TWIST 0
```

-- acoperiș

```
FIGURE SET Y=5, X=5, Z=29; N X=25, Z=29; N X=25, Z=32; N X=15, Z=35; N X=5, Z=32; H
```

```
ELEVATE H=30, TR. X=0, TR. Z=0
```

-- trecem în a doua secțiune

STOP

-- așteptăm 2 minute apariția *prompter*-ului READY

```
VIEW X=100, Y=50, Z=2
```

-- calculul perspectivei durează cca 6 minute

Punctul de vedere ales este al unei persoane în picioare (de la înălțimea de 2 m). Se poate trasa apoi și perspectiva din elicopter (înălțimea între 50 și 300 m).

4.4.3.3. Aplicație în inginerie

Să trasăm axonometria unui cot al unui arbore cotit. În cele ce urmează, descrierea formei o facem prescurtat, așa cum arătează programul instrucțiunile introduse.

```
CIRCLE Y=0: C0.0: R5: N12:
ELEVATE ^20: T0.0: *1:
CIRCLE Y=20: C0.0: R6: N12:
ELEVATE ^5: T0.0: *1:
CIRCLE Y=20: C0.16: R6: N12:
ELEVATE ^5: T0.0: *1:
PLOT 6.20.0:
DRAW 6.20.16:
PLOT -6.20.0:
DRAW -6.20.16:
PLOT 6.25.0:
DRAW 6.25.16:
PLOT -6.25.0:
DRAW -6.25.16:
CIRCLE Y=25: C0.16: R5: N12:
ELEVATE ^20: T0.0: *1:
STOP
VIEW 10000.10000.10000
```

Punctul de vedere situat la distanță foarte mare determină o proiecție practic paralelă (deci o axonometrie). În plus, axonometria este izometrică, deoarece am ales punctul de vedere cu $x=y=z$. Toate cele 6 proiecții ale piesei se obțin cu următoarele comenzi:

```
VIEW 10000.0.0
VIEW -10000.0.0
VIEW 0.10000.0
VIEW 0.-10000.0
VIEW 0.0.10000
VIEW 0.0.-10000
```

4.4.4. LISTINGUL PROGRAMULUI

```
0>REM Octavian Pleter 1985
1
2 REM ON ERROR GO TO cont !
3 DEF FN a(i,j)=22528+j+32*i
4 GO TO 1000
5 LET NT=0: LET NR=0
10 FOR g=1 TO 3: LET q(g)=p(g)-v(g): NEXT g
15 LET D=SQR (Q(X)*Q(X)+Q(Y)*Q(Y)+Q(Z)*Q(Z))
20 FOR G=X TO Z
25 LET Q(G)=Q(G)/D
30 LET NR=NR+A(G)*V(G): LET NT=NT+A(G)*Q(G): NEXT G
35 LET L=(NR-1)/NT
40 FOR G=X TO Z: LET Q(G)=V(G)-L*Q(G): NEXT G
50 LET E=(Q(X)*A27+(Q(Y)-S)*A28)
55 LET F=(Q(X)*A30+(Q(Y)-S)*A31+(Q(Z)-A29)*A32)
60 RETURN
1000 DIM O(700.3)
1010 DIM C(3): DIM T(3): DIM I$(700)
```



```

1020 LET X=1: LET Y=2: LET Z=3
1030 LET I=1: LET last=0: LET sur=0: LET circle=0: LET cont=1040
1040 BEEP .2.19: INPUT :: PRINT #1:" READY":TAB 29:i: POKE 23264,130: POKE
    23658,8
1050 IF INKEY$="Q" THEN GO TO 1300
1060 IF INKEY$="W" THEN GO TO 1350
1070 IF INKEY$="F" THEN GO TO 1400
1080 IF INKEY$="C" THEN GO TO 1600
1090 IF INKEY$="P" THEN GO TO 1800
1100 IF INKEY$="O" THEN GO TO 2500
1110 IF INKEY$="E" THEN GO TO 3000
1120 IF INKEY$=CHR$ 12 THEN GO TO 4100
1130 IF INKEY$="A" THEN GO TO 4200
1140 IF INKEY$="S" STOP " THEN GO TO 4800
1150 IF INKEY$="S" THEN GO TO 4300
1160 IF INKEY$="J" THEN GO TO 4500
1170 IF INKEY$="K" THEN GO TO 4700
1180 IF INKEY$="H" THEN GO TO 4900
1200 GO TO 1050
1300 INPUT :: PRINT "PLOT " : LET r$="*"
1320 FOR j=x TO z: INPUT "XYZ"(j)=" :O(i,j): PRINT O(i,j):"." : NEXT j: PRINT
    CHR$ 8 :"." :
1330 LET I$(i)=r$: LET i=i+1
1340 LET circle=0: PRINT : LET last=1: LET sur=0: GO TO 1040
1350 IF last<>1 THEN INPUT :: PRINT #1:" ERROR: STARTPOINT MISSING": POKE
    23264,130: BEEP .2-.7: PAUSE 250: GO TO 1040
1360 INPUT :: PRINT "DRAW " : LET r$="|"
1370 GO TO 1320
1380 INPUT "SET ":f:" AT ":O(i,f): PRINT "XYZ"(f)=" :O(i,f):: RETURN
1400 INPUT :: PRINT "FIGURE ":
1410 LET I$(i)="*": LET sur=i
1420 GO SUB 1380
1430 PRINT " ":
1440 FOR j=x TO z: IF j=f THEN GO TO 1460
1450 INPUT "XYZ"(j)=" :O(i,j): PRINT O(i,j):"." :
1460 NEXT j: PRINT CHR$ 8 :"." :
1470 BEEP .2.19: INPUT :: PRINT #1: FLASH 1:" " : FLASH 0:"READY (Next/Home)":TAB
    29:i+1
1480 IF I$(i)=" " THEN LET O(i,f)=O(i-1,f): LET I$(i)="|"
1490 LET i=i+1
1500 IF INKEY$="" THEN GO TO 1500
1510 IF INKEY$<>"H" THEN GO TO 1440
1520 FOR j=x TO z: LET O(i,j)=O(sur,j): PRINT ((STR$ O(i,j)+".") AND j<>f):: NEXT
    j: PRINT CHR$ 8 :"." :
1530 LET I$(i)="|": LET i=i+1: LET last=i-sur
1540 LET circle=0: PRINT : GO TO 1040
1600 INPUT :: PRINT "CIRCLE ":
1610 GO SUB 1380
1620 PRINT " : C":
1630 FOR j=x TO z: IF j=f THEN GO TO 1650
1640 INPUT "XYZ"(j):C=" :C(j): PRINT C(j):"." :
1650 NEXT j: PRINT CHR$ 8 :"." :
1660 INPUT "R=" :r: IF r<=0 THEN PRINT "R<=0!": PRINT : GO TO 1040
1661 PRINT "R":r:" :
1670 INPUT "N=" :n: IF n<=1 THEN PRINT "N<=1!": PRINT : GO TO 1040
1671 PRINT "N":n:" :
1680 INPUT "ANGLE=" :a: IF a1 THEN PRINT " A":a:" :
1685 LET I$(i)="*": LET sur=i
1690 LET circle=1: PRINT CHR$ 8 :"." :
1700 FOR a=a1 TO a1+360 STEP 360/n
1710 LET ar=a*PI/180: LET pr=1
1720 FOR j=x TO z: IF j=f THEN GO TO 1750
1730 LET arg=SIN ar: IF pr THEN LET arg=COS ar: LET pr=0
1740 LET O(i,j)=C(j)+r*arg
1750 NEXT j

```



```

1760 IF I$(i)="" THEN LET I$(i)="|": LET O(i,f)=O(i-1,f)
1770 LET i=i+1
1780 NEXT a
1790 PRINT : LET last=i-sur: GO TO 1040
1800 INPUT :: PRINT "PRISM ":
1810 FOR j=x TO z: INPUT "FROM ":XYZ"(j)!=" ":O(i,j): PRINT O(i,j):".": NEXT
j: PRINT CHR$ 8:".":
1820 FOR j=x TO z: INPUT "TO ":XYZ"(j)!=" ":C(j): PRINT C(j):".": NEXT j: PRINT
CHR$ 8:".":
1830 LET I$(i)="*": LET sur=i
1840 GO SUB 2400
1850 GO SUB 2300: LET O(i,z)=C(z)
1860 GO SUB 2400
1870 GO SUB 2300: LET O(i,x)=C(x): LET I$(i)="*"
1880 GO SUB 2300: LET O(i,z)=O(sur,z)
1890 GO SUB 2300: LET O(i,y)=C(y): LET I$(i)="*"
1900 GO SUB 2300: LET O(i,z)=C(z)
1910 GO SUB 2300: LET O(i,x)=O(sur,x): LET I$(i)="*"
1920 GO SUB 2300: LET O(i,z)=O(sur,z)
1930 LET circle=0: PRINT : LET i=i+1: LET last=i-sur: LET sur=0: GO TO 1040
2300 LET i=i+1: FOR j=x TO z
2310 LET O(i,j)=O(i-1,j): NEXT j
2320 LET I$(i)="|"
2330 RETURN
2400 GO SUB 2300: LET O(i,x)=C(x)
2410 GO SUB 2300: LET O(i,y)=C(y)
2420 GO SUB 2300: LET O(i,x)=O(sur,x)
2430 GO SUB 2300: LET O(i,y)=O(sur,y)
2440 RETURN
2500 IF NOT sur THEN INPUT :: PRINT #1:" ERROR: BASE NOT DEFINED": POKE FN
a(23,0).130: BEEP .2,-7: PAUSE 250: GO TO 1040
2510 INPUT :: PRINT "CONE ": LET il=i-1
2520 FOR j=x TO z: INPUT "TOP ":XYZ"(j)!=" ":C(j): PRINT C(j):".": NEXT j:
PRINT CHR$ 8:".":
2530 FOR k=il TO 1 STEP -1
2540 LET I$(i)="*"
2550 FOR j=x TO z: LET O(i,j)=O(k,j): NEXT j
2560 LET i=i+1
2570 LET I$(i)="|"
2580 FOR j=x TO z: LET O(i,j)=C(j): NEXT j
2590 LET i=i+1
2600 IF k=sur THEN PRINT : LET circle=0: LET last=i-sur: LET sur=0: GO TO 1040
2610 NEXT k: STOP
3000 IF NOT sur THEN INPUT :: PRINT #1:" ERROR: ELEVATION OF WHAT?": POKE FN
a(23,0).130: BEEP .2,-7: PAUSE 250: GO TO 1040
3010 INPUT :: PRINT "ELEVATE ":
3020 INPUT "H=":T(f): PRINT "^.":T(f):".":
3030 PRINT "T": FOR j=x TO z: IF j=f THEN GO TO 3050
3040 INPUT "TRANSLATION ":XYZ"(j)!=" ":T(j): PRINT T(j):".":
3050 NEXT j: PRINT CHR$ 8:".":
3055 LET k=sur
3060 IF NOT circle THEN GO TO 3400
3070 INPUT "AMPLIFY *":m
3080 PRINT " *":m:".":
3090 FOR j=x TO z: LET C(j)=C(j)+T(j): NEXT j
3100 LET r=r*m: LET l=i+2*n+2: LET o=1
3110 INPUT "TWIST ANGLE=":m: IF m THEN PRINT " A":m:".":
3120 PRINT CHR$ 8:".":
3130 LET al=a+l*m: LET I$(l)="*"
3140 FOR a=al TO al+360 STEP 360/n
3150 LET ar=a*PI/180: LET pr=1
3160 FOR j=x TO z: IF j=f THEN GO TO 3190
3170 LET arg=SIN ar: IF pr THEN LET arg=COS ar: LET pr=0
3180 LET O(i,j)=C(j)+r*arg
3190 NEXT j

```



```

3200 LET I$(i)="*"
3210 LET O(i,f)=O(i-1,f)+T(f)
3220 FOR j=x TO z
3230 LET O(1,j)=O(i,j)
3240 NEXT j: IF I$(1)=" " THEN LET I$(1)="|"
3250 LET i=i+1: LET I$(i)="|"
3260 FOR j=x TO z
3270 LET O(i,j)=O(k,j)
3280 NEXT j
3290 LET l=l+1: LET k=k+1: LET i=i+1
3300 NEXT a
3310 LET i=1: GO TO 4000
3400 PRINT CHR$ 8:":": PRINT : LET n=i-k-1: LET l=i+2*n+2: LET o=1: LET I$(1)="*"
3410 FOR a=1 TO n+1
3420 LET I$(a)="*"
3430 FOR j=x TO z: LET O(i,j)=O(k,j): NEXT j
3440 LET i=i+1
3450 FOR j=x TO z
3460 LET O(i,j)=O(i-1,j)+T(j): LET O(1,j)=O(i,j): NEXT j: LET I$(i)="|"
3470 IF I$(1)=" " THEN LET I$(1)="|"
3480 GO TO 3290
4000 PRINT : LET last=i-sur: LET sur=o: GO TO 1040
4100 INPUT : PRINT "DELETE": LET sur=0: LET circle=0
4105 LET i=i-last: IF i<0 THEN LET i=1: GO TO 4120
4110 FOR j=i TO i+last: LET I$(j)=" ": NEXT j
4120 PRINT : GO TO 1040
4200 INPUT : PRINT #1:" NEW (Y TO CONFIRM)": POKE 23264,130: BEEP .1,27: BEEP
      .1,27: BEEP .1,27
4210 FOR j=1 TO 200
4220 IF INKEY$="Y" THEN RUN
4225 IF INKEY$<>" " AND INKEY$<>"Y" THEN GO TO 1040
4230 NEXT j
4240 GO TO 1040
4300 IF i<2 THEN GO TO 1040
4301 INPUT : DIM I(i,4): PRINT "SAVE ":
4310 LET I(1,1)=i
4320 FOR j=1 TO i-1
4330 FOR k=x TO z
4340 LET I(j+1,k)=O(j,k): NEXT k
4350 IF I$(j)="*" THEN LET I(j+1,4)=-1
4351 IF I$(j)="|" THEN LET I(j+1,4)=+1
4360 NEXT j
4370 POKE 23658,0: INPUT "NAME: ": LINE n$: INPUT "TAPE (0) MDRIVE (1-8) ":MD:
      POKE 23658,8
4380 IF LEN n$>10 OR NOT LEN n$ THEN GO TO 4370
4390 PRINT n$: IF NOT MD THEN SAVE n$ DATA I(): GO TO 4400
4395 SAVE *M":MD:N$ DATA I()
4400 INPUT : PRINT #1:" VERIFY? (Y/N)": POKE 23264,130
4410 IF INKEY$=" " THEN GO TO 4410
4411 INPUT :
4420 IF INKEY$<>"Y" THEN GO TO 4450
4430 INPUT : PRINT "VERIFY on error GO TO cont "
4440 IF NOT MD THEN VERIFY n$ DATA I(): GO TO 4450
4445 VERIFY *M":MD:N$ DATA I()
4450 PRINT : DIM I(1): GO TO 1040
4500 INPUT : PRINT "LOAD ":
4510 POKE 23658,0: INPUT "NAME: ": LINE n$: INPUT "TAPE (0) MDRIVE (1-8) ":MD:
      POKE 23658,8: IF NOT LEN n$ AND MD THEN GO TO 4510
4520 PRINT n$: IF NOT MD THEN LOAD n$ DATA I(): GO TO 4530
4525 LOAD *M":MD:N$ DATA I()
4530 LET i=I(1,1)
4535 DIM O(700,3): DIM I$(700)
4540 FOR j=1 TO i-1
4550 FOR k=x TO z
4560 LET O(j,k)=I(j+1,k): NEXT k

```



```

4570 LET I$(j) = ("|" AND I(j+1.4)=1)+("*" AND I(j+1.4)=-1)
4580 NEXT j
4590 LET last=0: LET sur=0: LET circle=0
4600 DIM I(1): PRINT "
4610 GO TO 1040
4700 IF i<2 THEN GO TO 1040
4701 INPUT :: PRINT "LIST":TAB 8:"X":TAB 18:"Y":TAB 28:"Z":TAB 31:"T":TAB 0:"

```

```

4705 DIM n$(10)
4710 FOR j=1 TO i-1: LET j$=STR$ j: LET a$=STR$ O(j,x): LET b$=STR$ O(j,y): LET
  c$=STR$ O(j,z)
4711 IF LEN a$>8 THEN LET a$=a$( TO 8)
4712 IF LEN b$>8 THEN LET b$=b$( TO 8)
4713 IF LEN c$>8 THEN LET c$=c$( TO 8)
4714 IF LEN j$=1 THEN LET j$=" "+j$
4715 IF LEN j$=2 THEN LET j$=" "+j$
4720 PRINT j$:"":n$( TO 8-LEN a$):a$:TAB 13:n$( TO 8-LEN b$):b$:TAB 22:n$( TO
  8-LEN c$):c$:TAB 31:I$(j)
4730 NEXT j
4740 GO TO 6700
4800 IF i<3 THEN GO TO 1040
4810 INPUT :: PRINT "STOP": GO TO 5000
4900 CLS : INPUT :: PRINT #1:" HELP":TAB 26:"PAGE 1": PRINT "[Q]PLOT x,y,z""
  [W]DRAW x,y,z (LINE)""[F]FIGURE plane,coords (CONTOUR)""[C]CIRCLE
  plane,center,radius. nb. of points,angle between first point and
  ordinate of plane (POLYGON)"
4905 POKE FN a(23.0),130
4910 PRINT "[P]PRISM from.to: coords of two diagonal opposite
  corners""[O]CONE top coords (PYRAMID)""[E]ELEVATION: translation of the
  latest defined surface (ampli fied or twisted), link of the corre-
  sponding points"
4920 PAUSE 0: CLS
4930 INPUT :: PRINT #1:" HELP":TAB 26:"PAGE 2": POKE 23264,130
4940 PRINT "[CAPS]+[O]DELETE latest defined point, surface or body""
4950 PRINT "[A]NEW start again: y to confirm"
4960 PRINT "[S]SAVE & VERIFYthe file created"
4970 PRINT "[J]LOAD an old file"
4980 PRINT "[K]LIST the coords generated""[SYMB]+[A]STOP close the file"
4990 PRINT "[H]HELP list the commands": PAUSE 0: CLS : GO TO 1040
5000 DIM I(700.2): LET np=i-1
5010 DIM V(3): DIM A(3): DIM C(3): DIM P(3): DIM E(6): DIM Q(3)
5020 LET U=100000: FOR I=1 TO 6
5030 LET E(I)=U: LET U=-U: NEXT I
5040 FOR I=1 TO np
5050 FOR J=X TO Z
5060 IF O(I,J)<E(2*J-1) THEN LET E(2*J-1)=O(I,J)
5070 IF O(I,J)>E(2*J) THEN LET E(2*J)=O(I,J)
5080 IF O(I,J)>E(2*J) THEN LET E(2*J)=O(I,J)
5090 NEXT J: NEXT I
5100 FOR I=X TO Z: LET C(I)=(E(2*I-1)+E(2*I))/2: NEXT I
5110 LET cont=6000: LET out=0: LET x1=2: LET x2=253: LET y1=2: LET y2=173: CLS
  : GO TO 6000
5120 LET CYL=0: INPUT "VIEWPOINT X= ":V(X)
5130 INPUT "VIEWPOINT Y= ":V(Y): IF V(Y)=0 THEN LET V(Y)=1e-3
5140 INPUT "VIEWPOINT Z= ":V(Z)
5150 LET T=0: LET U=0: FOR I=X TO Z: LET T=(C(I)-E(2*I))*(C(I)-E(2*I)): LET
  U=U+(C(I)-V(I))*(C(I)-V(I)): NEXT I
5160 IF U<4*T THEN INPUT :: PRINT #1:" ERROR: TOO CLOSE": POKE 23264,130: BEEP
  .2-.7: PAUSE 250: GO TO (5120 AND NOT CYL)+(6110 AND CYL)
5170 LET D=0: LET S=0: LET A27=1: LET A28=0: LET A29=0: LET A30=0: LET A31=1
5180 FOR I=X TO Z
5190 LET T=(V(I)+C(I))/2
5200 LET D=D+(C(I)-V(I))*T

```



```

5210 NEXT I
5220 FOR I=X TO Z: LET A(I)=(C(I)-V(I))/D: NEXT I
5230 LET T=PI/2: IF ABS A(Z)>1E-8 THEN LET T=ATN (SQR (A(X)*A(X)+
A(Y)*A(Y))/A(Z)): GO TO 5250
5240 INPUT :: PRINT #1:" VERTICAL PERSPECTIVE"
5250 LET A32=SIN ABS T
5260 LET D=SQR ((A(X)*A(X)+A(Y)*A(Y))*(A(X)*A(X)+A(Y)*A(Y)+ A(Z)*A(Z))): IF
D<1E-9 THEN GO TO 5350
5270 LET A30=SGN (C(X)-V(X))*SGN T*ABS (A(X)*A(Z))/D
5280 LET A31=SGN (C(Y)-V(Y))*SGN T*ABS (A(Y)*A(Z))/D
5290 LET A27=SGN (C(Y)-V(Y))*ABS A(Y)/SQR (A(X)*A(X)+A(Y)*A(Y))
5300 LET A28=SGN (V(X)-C(X))*ABS A(X)/SQR (A(X)*A(X)+A(Y)*A(Y))
5310 IF A32=1 THEN GO TO 5340
5320 IF T<0 THEN INPUT :: PRINT #1:" ASCENDENT PERSPECTIVE"
5330 IF T>0 THEN INPUT :: PRINT #1:" DESCENDENT PERSPECTIVE"
5340 IF A(Y)=0 THEN BEEP .2,-7: INPUT :: PRINT #1:" ERROR: DIV BY ZERO": POKE
23264,130: PAUSE 250: GO TO 6000
5341 LET S=1/A(Y): GO TO 5360
5350 LET A29=V(Z)/2: INPUT :: PRINT #1:" HORIZONTAL PERSPECTIVE"
5360 FOR I=X TO Z: LET P(I)=C(I): NEXT I
5370 GO SUB 5: LET M=E-100: LET N=F-80
5380 LET M=E-100: LET N=F-80
5390 FOR K=1 TO np
5400 FOR I=X TO Z: LET P(I)=O(K,I): NEXT I
5410 GO SUB 5
5420 LET I(K,X)=E-M
5430 LET I(K,Y)=F-N
5440 NEXT K
5450 LET Q(X)=-1e5: LET Q(Y)=-1e5: LET P(X)=1e5: LET P(Y)=1e5
5460 FOR I=1 TO np
5470 FOR J=X TO Y
5480 IF I(I,J)>Q(J) THEN LET Q(J)=I(I,J)
5490 IF I(I,J)<P(J) THEN LET P(J)=I(I,J)
5500 NEXT J: NEXT I
5510 LET xmax=Q(X): LET xmin=P(X): LET ymax=Q(Y): LET ymin=P(Y)
5520 IF (xmax-xmin)/(ymax-ymin)>(x2-x1)/(y2-y1) THEN LET a=(x2-x1)/(xmax-xmin):
LET bx=x1-(x2-x1)*xmin/(xmax-xmin): LET by=(y2+y1)/2-((ymax+ ymin)/2)*a:
GO TO 5570
5530 LET a=(y2-y1)/(ymax-ymin)
5540 LET by=y1-(y2-y1)*xmin/(ymax-ymin)
5550 LET bx=(x2+x1)/2-((xmax+xmin)/2)*a
5570 FOR I=1 TO np
5580 LET I(I,X)=a*I(I,X)+bx
5590 LET I(I,Y)=a*I(I,Y)+by
5600 NEXT I
5730 CLS : BEEP .2,19: INPUT "SCREEN (0) MDRIVE (1-8) ":MD: IF MD THEN GO TO 6730
5732 IF out THEN GO TO 6510
5735 GO SUB 5740: GO TO 6000
5740 FOR I=1 TO np
5750 IF I$(I)="*" THEN GO TO 5780
5760 DRAW INT (I(I,X)+.5)-INT (I(I-1,X)+.5),INT (I(I,Y)+.5)-INT (I(I-1,Y)+.5)
5770 GO TO 5790
5780 PLOT INT (I(I,X)+.5),INT (I(I,Y)+.5)
5790 NEXT I: RETURN
6000 BEEP .2,19: INPUT :: PRINT #1:" READY":TAB 25:"CENTRAL": POKE 23264,130:
POKE 23658,8
6010 IF INKEY$="H" THEN GO TO 6200
6020 IF INKEY$="V" THEN GO TO 5120
6040 IF INKEY$="Y" THEN LET i=np+1: CLS : LET cont=1040: GO TO 1040
6050 IF INKEY$="D" THEN GO TO 6220
6060 IF INKEY$="P" THEN GO TO 5730
6070 IF INKEY$="C" THEN GO TO 6110
6080 IF INKEY$="R" THEN GO TO 6400
6090 IF INKEY$="K" THEN GO TO 6510
6091 IF INKEY$="S" THEN GO TO 6101

```



```

6100 GO TO 6010
6101 POKE 23658,0: INPUT "NAME: ": LINE n$: INPUT "TAPE (0) MDRIVE (1-8) ":MD:
    POKE 23658,8: DIM m$(10): LET m$=n$: IF NOT MD THEN SAVE m$SCREEN$: GO TO
    6000
6102 SAVE *JM:MD:N$SCREEN$: GO TO 6000
6110 LET CYL=1: INPUT "DISTANCE D= ":d
6120 INPUT "AZIMUTH [DEG]= ":a: IF a=0 THEN LET a=0.1
6130 INPUT "HEIGHT H= ":h
6140 LET a=a*PI/180
6150 LET v(x)=c(x)+d*COS a
6160 LET v(y)=c(y)+d*SIN a
6170 LET v(z)=c(z)+h
6180 GO TO 5150
6200 CLS : PRINT "[V]VIEW draw perspective of the object viewed from a given
    point (X,Y,Z)""[Y]RETURN to the first stage"
6201 PRINT "[D]DETAILS display the viewpoint coords, aprox. center coords
    and the scale factors"
6202 PRINT "[P]PICTURE display the image"
6203 PRINT "[R]RESCALE change the draw scale"
6204 PRINT "[K]LIST list the coords X,Y of the calculated perspective"
6205 PRINT "[C]CYLINDRICAL enter viewpoint in cylindrical coords:
    distance, azimuth, height"
6206 PRINT "[S]SAVE dump the screen to tape"
6210 GO TO 6000
6220 CLS : PRINT "DETAILS""VIEWPOINT: ":V(X):".":V(Y):".":V(Z)
6230 LET D=SQR ((V(X)-C(X))*(V(X)-C(X))+(V(Y)-C(Y))*(V(Y)-C(Y))): LET A=ASN
    ((V(Y)-C(Y))/D): LET H=V(Z)-C(Z)
6240 PRINT "CYLINDRICAL COORDINATES:":TAB 7:"distance= ":D:TAB 7:"azimuth=
    ":A*180/PI:" [DEG]:TAB 7:"height= ":H
6250 PRINT "CENTER:":C(X):".":C(Y):".":C(Z)
6260 PRINT "PERSPECTIVE PLANE PARAMETERS:":TAB 7:"a= ":A(X):TAB 7:"b= ":A(Y):TAB
    7:"c= ":A(Z)
6280 PRINT "SCALE: ":X1:".":X2:"/":Y1:".":Y2
6290 IF out THEN PRINT "OUT OF SCREEN"
6300 GO TO 6000
6400 INPUT :: BEEP .2,19: PRINT #1:" SCALE (Default/Rescale)": POKE 23264,130
6401 IF INKEY$="" THEN GO TO 6401
6402 IF INKEY$="D" THEN LET out=0: LET x1=2: LET x2=253: LET y1=2: LET y2=173:
    GO TO 6450
6403 IF INKEY$<>"R" THEN GO TO 6400
6405 INPUT "XMIN= ":x1,"XMAX= ":x2,"YMIN= ":y1,"YMAX= ":y2
6410 IF x2<=x1 OR y2<=y1 THEN BEEP .2,-7: PRINT #1:" ERROR: MAX LESS THAN MIN":
    POKE 23264,130: PAUSE 250: GO TO 6400
6420 LET out=0
6430 IF x1<2 OR x1>245 OR x2>253 OR x2<10 OR y1<2 OR y1>165 OR y2>173 OR y2<10
    THEN LET out=1: BEEP .2,-7: PRINT #1:" WARNING: OUT OF SCREEN": POKE
    23264,130: PAUSE 250
6450 BEEP .2,19: INPUT :: PRINT #1:" RESCALE CURRENT IMAGE? (Y/N)": POKE
    23264,130
6460 IF INKEY$="Y" THEN INPUT :: GO TO 5450
6470 IF INKEY$="N" THEN GO TO 6000
6480 GO TO 6460
6510 INPUT :: PRINT "LIST":TAB 13:"X":TAB 25:"Y":TAB 31:"T":TAB 0:"
    "
    "
6520 DIM n$(10)
6530 FOR j=1 TO np: LET j$=STR$ j: LET a$=STR$ I(j,x): LET b$=STR$ I(j,y)
6540 FOR k=1 TO LEN a$: IF a$(k)=". " AND k+2<=LEN a$ THEN LET a$=a$( TO k+2): GO
    TO 6570
6550 IF a$(k)=". " AND k+1<=LEN a$ THEN LET a$=a$( TO k+1)+"0": GO TO 6570
6555 IF a$(k)=". " THEN LET a$=a$( TO k)+"00": GO TO 6570
6560 NEXT k: LET a$=a$+".00"
6570 FOR k=1 TO LEN b$: IF b$(k)=". " AND k+2<=LEN b$ THEN LET b$=b$( TO k+2): GO
    TO 6600

```



```

6580 IF b$(k)=". " AND k+1<=LEN b$ THEN LET b$=b$( TO k+1)+"0": GO TO 6600
6585 IF b$(k)=". " THEN LET b$=b$( TO k)+"00": GO TO 6600
6590 NEXT k: LET b$=b$+".00"
6600 PRINT " "(1 TO 3-LEN j$):j$:".":TAB 8:n$( TO 8-LEN a$):a$:TAB 20:n$( TO
      8-LEN b$):b$:TAB 31:I$(j)
6610 NEXT j
6700 PRINT "
: PRINT : GO TO cont
6730 INPUT "NAME: ": LINE N$: IF NOT LEN N$ OR LEN N$>10 THEN GO TO 6730
6750 OPEN #5:"M":MD:N$
6760 FOR J=1 TO NP: PRINT #5:I$(J):INT (I(J.X)+.5):".":768-INT (I(J.Y)+.5): NEXT
      J
6770 PRINT #5:"█": CLOSE #5: GO TO 6000
7000 PAPER 1: INK 7: BORDER 1
7010 CLS
7040 PRINT AT 7.8: PAPER 0:"CENTRAL CAD V3.4":AT 12.3:"Octavian Pleter
      ©1982.1990"
7500 PAUSE 0: RUN
9999 SAVE "CENTRAL" LINE 7000

```

4.5. GRAFICĂ DE ÎNALTĂ REZOLUȚIE LA IMPRIMANTA MATRICEALĂ

Acest program este un *overlay*, adică un segment de cod care trebuie încărcat peste un alt program, în scopul extinderii sau modificării funcționării acestuia. În cazul de față, *overlay*-ul se încarcă peste programul Tuttiprint, prezentat în § 4.1. El permite încărcarea a 8 fișiere ecran de pe bandă sau de pe disc și tipărirea lor în modul grafic, astfel încât să formeze prin juxtapunere o imagine de 512×768 pixeli, conform figurii 4.2. La rezoluția unei imprimante matriceale cu 9 ace, cele 8 ecrane juxtapuse acoperă aproximativ întreaga suprafață activă a unei pagini normale A4.

HISOFT GENS3M2 ASSEMBLER
ZX SPECTRUM

Copyright (C) HISOFT 1983.4
All rights reserved

Pass 1 errors: 00

```

F870      10      ORG      63600
FA0C      20 NR SPA EQU      64012
FDA7      30 SEND EQU      #FDA7
FD81      40 POINT EQU      #FD81
FEC9      50 y cd EQU      #FEC9
          60 HCOPY2
          70 :formatter grafic pentru 512X768
F870 3A1FFE      80      LD      A,(65055)
F873 FE3E      90      CP      62

```

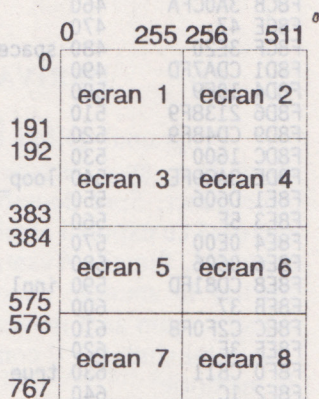


Fig. 4.2. Organizarea spațiului grafic de înaltă rezoluție (512×768)

F875	C0	100	RET	NZ
F876	3AF9FE	110	LD	A, (65273)
F879	FE00	120	CP	0
F87B	CA56F9	130	JP	Z, SCAMP
F87E	FE8D	140	CP	141
F880	CAB5F8	150	JP	Z, ROMOM
F883	FE1B	160	CP	27
F885	2801	170	JR	Z, K6313
F887	C9	180	RET	
F888	3E1B	190	K6313	LD A, 27
F88A	3246F9	200	LD	(final), A
F88D	3E0A	210	LD	A, 10
F88F	323DF9	220	LD	(halfn1), A
F892	3E8D	230	LD	A, 141
F894	323EF9	240	LD	(halfn1+1), A
F897	3E41	250	LD	A, 65
F899	3242F9	260	LD	(norm+1), A
F89C	3E06	270	LD	A, 6
F89E	3243F9	280	LD	(norm+2), A
F8A1	3E1B	290	LD	A, 27
F8A3	3244F9	300	LD	(norm+3), A
F8A6	3E2A	310	LD	A, 42
F8A8	3239F9	320	LD	(size+1), A
F8AB	3E05	330	LD	A, 5
F8AD	323AF9	340	LD	(size+2), A
F8B0	3E00	350	LD	A, 0
F8B2	323BF9	360	LD	(size+3), A
F8B5	2141F9	370	ROMOM	LD HL, norm
F8B8	CD48F9	380	CALL	SENTXT
F8BB	AF	390	XOR	A
F8BC	32C9FE	400	LD	(y cd), A
F8BF	3AC9FE	410	loop_y	LD A, (y_cd)
F8C2	C606	420	ADD	A, 6
F8C4	32C9FE	430	LD	(y cd), A
F8C7	FEC6	440	CP	#C6
F8C9	2866	450	JR	Z, exit
F8CB	3A0CFA	460	LD	A, (NR_SPA)
F8CE	47	470	LD	B, A
F8CF	3E20	480	spaces	LD A, " "
F8D1	CDA7FD	490	CALL	SEND
F8D4	10F9	500	DJNZ	spaces
F8D6	2138F9	510	LD	HL, size
F8D9	CD48F9	520	CALL	SENTXT
F8DC	1600	530	LD	D, 0
F8DE	3AC9FE	540	loop_x	LD A, (y_cd)
F8E1	D606	550	SUB	6
F8E3	5F	560	LD	E, A
F8E4	0E00	570	LD	C, 0
F8E6	0606	580	LD	B, 6
F8E8	CD81FD	590	inn1	CALL POINT
F8EB	37	600	SCF	
F8EC	C2F0F8	610	JP	NZ, true
F8EF	3F	620	CCF	
F8F0	CB11	630	true	RL C
F8F2	1C	640	INC	E
F8F3	10F3	650	DJNZ	inn1
F8F5	79	660	LD	A, C
F8F6	CDA7FD	670	CALL	SEND
F8F9	14	680	INC	D
F8FA	AF	690	XOR	A
F8FB	BA	700	CP	D
F8FC	C2DEF8	710	JP	NZ, loop_x
F8FF	3A86FD	720	LD	A, (#FD86)
F902	FE37	730	CP	#37
F904	2811	740	JR	Z, ELSE
F906	3E37	750	LD	A, #37

F908	3286FD	760	LD	(#FD86).A
F90B	3E1F	770	LD	A.#1F
F90D	3287FD	780	LD	(#FD87).A
F910	3E0F	790	LD	A.#0F
F912	3288FD	800	LD	(#FD88).A
F915	1812	810	JR	CONT
F917	3E0F	820	ELSE	LD A.#0F
F919	3286FD	830	LD	(#FD86).A
F91C	3E37	840	LD	A.#37
F91E	3287FD	850	LD	(#FD87).A
F921	3E1F	860	LD	A.#1F
F923	3288FD	870	LD	(#FD88).A
F926	C3DEF8	880	JP	loop_x
F929	213DF9	890	CONT	LD HL,halfn1
F92C	CD48F9	900	CALL	SENTXT
F92F	188E	910	JR	loop_y
F931	2146F9	920	exit	LD HL,final
F934	CD48F9	930	CALL	SENTXT
F937	C9	940	RET	
F938	1B4B0082	950	size	DEFB #1B.#4B.0.2+128.2+128
F93D	1B5B3165	960	halfn1	DEFB #1B.#5B.#31.#65
F941	1B5B30E0	970	norm	DEFB #1B.#5B.#30.#60+128."2"+128
F946	8DB0	980	final	DEFB 13+128."0"+128
		990		
		1000	SENTXT	
		1010	:print	text de la HL până când
		1020	:bitul	7 este găsit setat

F948	7E	1030	LD	A.(HL)
F949	F5	1040	PUSH	AF
F94A	E67F	1050	AND	#7F
F94C	CDA7FD	1060	CALL	SEND
F94F	23	1070	INC	HL
F950	F1	1080	POP	AF
F951	E680	1090	AND	#80
F953	28F3	1100	JR	Z.SENTXT
F955	C9	1110	RET	

		1120	SCAMP	
		1130	:formatter	grafic pentru SCAMP
F956	3E1B	1140	LD	A.27
F958	CDA7FD	1150	CALL	SEND
F95B	3E47	1160	LD	A."G"
F95D	CDA7FD	1170	CALL	SEND
F960	AF	1180	XOR	A
F961	32C9FE	1190	LD	(y_cd).A
F964	3AC9FE	1200	soop_y	LD A.(y_cd)
F967	C606	1210	ADD	A.6
F969	32C9FE	1220	LD	(y_cd).A
F96C	FEC6	1230	CP	#C6
F96E	2853	1240	JR	Z.sexit
F970	1600	1250	LD	D.0
F972	3AC9FE	1260	soop_x	LD A.(y_cd)
F975	3D	1270	DEC	A
F976	5F	1280	LD	E.A
F977	0E01	1290	LD	C.1
F979	0606	1300	LD	B.6
F97B	CD81FD	1310	sinn1	CALL POINT
F97E	37	1320	SCF	
F97F	C283F9	1330	JP	NZ.strue
F982	3F	1340	CCF	
F983	CB11	1350	strue	RL C
F985	1D	1360	DEC	E
F986	10F3	1370	DJNZ	sinn1
F988	79	1380	LD	A.C
F989	CDA7FD	1390	CALL	SEND
F98C	14	1400	INC	D
F98D	AF	1410	XOR	A

F98E	BA	1420	CP	D
F98F	C272F9	1430	JP	NZ, soop x
F992	3A86FD	1440	LD	A, (#FD86)
F995	FE37	1450	CP	#37
F997	2811	1460	JR	Z, SELSE
F999	3E37	1470	LD	A, #37
F99B	3286FD	1480	LD	(#FD86).A
F99E	3E1F	1490	LD	A, #1F
F9A0	3287FD	1500	LD	(#FD87).A
F9A3	3E0F	1510	LD	A, #0F
F9A5	3288FD	1520	LD	(#FD88).A
F9A8	1812	1530	JR	SCONT
F9AA	3E0F	1540	LD	A, #0F
F9AC	3286FD	1550	LD	(#FD86).A
F9AF	3E37	1560	LD	A, #37
F9B1	3287FD	1570	LD	(#FD87).A
F9B4	3E1F	1580	LD	A, #1F
F9B6	3288FD	1590	LD	(#FD88).A
F9B9	C372F9	1600	JP	soop x
F9BC	3E2F	1610	LD	A, "/"
F9BE	CDA7FD	1620	CALL	SEND
F9C1	18A1	1630	JR	soop_y
F9C3	3E2B	1640	LD	A, "+"
F9C5	CDA7FD	1650	CALL	SEND
F9C8	C9	1660	RET	
F9C9	00	1670	zz	NOP

Pass 2 errors: 00

CONT	F929	ELSE	F917	HCOPY2	F870	K6313	F888	NR SPA	FA0C
POINT	FD81	ROMOM	F8B5	SCAMP	F956	SCONT	F9BC	SELSE	F9AA
SEND	FDA7	SENTXT	F948	exit	F931	final	F946	halfnl	F93D
innl	F8E8	loop_x	F8DE	loop_y	F8BF	norm	F941	sexit	F9C3
sinnl	F97B	size_	F938	soop_x	F972	soop_y	F964	spaces	F8FC
strue	F983	true	F8F0	y_cd_	FEC9	zz	F9C9		

Table used: 358 from 389

4.6. DERIVATOR SIMBOLIC DE FUNCȚII

Programul Difun derivează simbolic orice expresie dată în raport cu variabila x , apoi permite evaluarea numerică directă a funcției și derivatei în orice punct. Spre deosebire de calculul numeric tradițional, calculul simbolic operează cu expresii algebrice văzute ca liste de simboluri: variabile, constante, operatori, funcții. Avantajele calculului simbolic sunt: precizia, neacumularea erorilor în anumite metode și posibilitatea abordării unor calcule imposibile numeric, cum ar fi inversarea unei matrice de funcții. Dezavantajul principal al calculului simbolic îl constituie consumul mare de resurse de timp și de spațiu de memorie.

Expresia cerută de program prin *prompter*-ul $f(x)=$ se compune din:

- constante numerice fără semn, fie întregi (3, 10000), fie reale cu punct zecimal, *nu* cu exponent (1.25, 0.0066, dar nu $3e-4$);
- constante și variabile algebrice (a , ϵ , u , y);
- variabila în raport cu care se derivează (x);
- operatori aritmetici binari: $+$, $-$, $*$, $/$, $^$; (nu se admit operatorii unari $+$ și $-$, ca în exemplele: -5 , $+b$);
- paranteze: ($;$);

- funcții standard, scrise literă cu literă, cu argumentul între paranteze:
 - funcțiile trigonometrice sin, cos, tan, cot;
 - funcțiile hiperbolice sinh, cosh, tanh, coth;
 - funcțiile invers trigonometrice arcsin, arccos, arctan, arccot;
 - funcțiile invers hiperbolice arcsinh, arccosh, arctanh, arccoth;
 - exponențiala exp și logaritmul natural ln;
 - rădăcina pătrată sqr;

Pentru introducerea unei expresii care conține - unar, de exemplu $-\sin(-x)$, operatorul va fi prefixat de 0, astfel: $0-\sin(0-x)$. În rest, sintaxa expresiilor o respectă pe cea a Basic-ului. Scrierea cu litere mici este obligatorie.

Programul nu este capabil să prelucreză expresia obținută prin simplificarea, reducere de termeni sau calcule numerice parțiale. Evaluarea numerică este permisă numai dacă expresia nu conține alte variabile în afară de x .

Exemplu

Să se deriveze funcția $f(x)$ și să se calculeze derivata în punctele $x=0.3$, $x=\pi/2$ și $x=0$:

$$f(x) = \frac{x^2}{\sin x}$$

Se introduce funcția: $f(x)=x^2/\sin(x)$ și se obține derivata:

$f'(x)=(2*x*\sin(x)-x^2*\cos(x))/\sin(x)^2$, ceea ce în notația matematică reprezintă:

$$f'(x) = \frac{2x \sin x - x^2 \cos x}{\sin^2 x}$$

În continuare se utilizează calculatorul, apăsând [C]:

$x=0.3$

$$f(0.3)=0.3045477$$

$$f'(0.3)=1.0457981$$

$x=PI/2$

$$f(1.5707963)=2.4674011$$

$$f'(1.5707963)=3.1415927 \text{ (observați precizia lui } \pi \text{)}$$

$x=0$

$$f(0)= \text{6 Number too big. 5030:2}$$

Se continuă cu GO TO g, deoarece $x=0$ este un punct singular, atât al funcției, cât și al derivatei.

4.6.1. LISTINGUL PROGRAMULUI

```

2 GO SUB 8000
3 DEF FN c(x)=1/TAN x: DEF FN a(x)=ATN (1/x): DEF FN s(x)=0.5*(EXP x-EXP -x):
  DEF FN b(x)=0.5*(EXP x+EXP -x): DEF FN t(x)=(EXP x-EXP -x)/(EXP x+EXP -x):
  DEF FN d(x)=(EXP x+EXP -x)/(EXP x-EXP -x): DEF FN e(x)=LN (x+SQR (x*x+1)):
  DEF FN f(x)=LN (x+SGN x*SQR (x*x-1)): DEF FN g(x)=0.5*LN ((1+x)/(1-x)): DEF
  FN h(x)=0.5*LN ((x+1)/(x-1))
4 GO TO 40
5 LET e=1: DEF FN b$(h$)="("+h$+")"
6 IF r$(e)<>" " THEN LET e=e+1: GO TO 6
7 IF e>1 THEN LET r$=r$( TO e-1): RETURN
8 LET r$="": RETURN
10 LET sd=0: FOR e=1 TO LEN d$
11 IF d$(e)="+ " OR d$(e)="- " THEN LET sd=1: RETURN
12 NEXT e: RETURN
15 FOR e=1 TO LEN b$
16 IF b$(e)<"(" OR b$(e)>"9" OR b$(e)=", " THEN LET d$=b$+"-1": RETURN
17 NEXT e
18 LET d$=" "+STR$ (VAL b$-1): RETURN
40 LET m=16: REM adincimea max a expresiilor
45 LET w=350: REM lungimea max a expresiilor
50 DIM f$(m,w): DIM s$(m,w): DIM u$(m,w): DIM v$(m,w): DIM o$(m,w): DIM o(m):
  LET z$="+-*/^"
51 RESTORE 4000: READ fz: DIM x$(fz,20): DIM y$(fz,20): DIM l(fz)
52 FOR i=1 TO fz: READ w$,y$(i)
53 LET l(i)=LEN w$: LET x$(i)=w$
54 NEXT i
55 BORDER 7: PAPER 7: INK 0: CLS
60 LET z=1
70 INPUT "f(x)=": LINE w$: IF LEN w$=0 THEN GO TO 70
74 CLS : PRINT "derivarea simbolica a functiei:"
75 LET r$=""
76 FOR e=1 TO LEN w$
77 IF w$(e)<>" " THEN LET r$=r$+w$(e)
78 NEXT e: LET w$=r$: LET f$(1)=w$
79 PRINT "f(x)="+w$
80 GO SUB 170
90 LET r$=f$(1): GO SUB 5: PRINT "f(x)="+r$: LET h$=r$
95 LET r$=s$(1): GO SUB 5: PRINT "f'(x)="+r$: LET d$=r$
100 GO TO 5000
170 LET r$=f$(z): GO SUB 5: LET w$=r$
171 POKE 23692,255
175 PRINT " ? "":w$: ""
180 LET q$=w$: GO SUB 2010
200 IF p=1 OR p=1 THEN GO TO 1000: REM secventa ilegala de operatori
260 IF p=0 THEN GO TO 700
280 REM separare f=u#v => u', v'
290 LET o(z)=o: LET u$(z)=w$( TO p-1): LET v$(z)=w$(p+1 TO )
300 LET z=z+1: LET f$(z)=u$(z-1): GO SUB 170: LET o$(z-1)=s$(z)
310 LET f$(z)=v$(z-1): GO SUB 170: LET z=z-1
320 LET r$=u$(z): GO SUB 5: LET a$=r$: LET r$=v$(z): GO SUB 5: LET b$=r$: LET
  r$=o$(z): GO SUB 5: LET c$=r$: LET r$=s$(z+1): GO SUB 5: LET d$=r$: LET
  o=o(z)
325 IF o=2 AND LEN d$>1 THEN GO SUB 10: IF sd THEN LET d$=FN b$(d$)
331 IF o=1 THEN GO TO 350
332 IF o=2 THEN GO TO 350
333 IF o=3 THEN GO TO 400
334 IF o=4 THEN GO TO 400
335 IF o=5 THEN GO TO 540
336 GO TO 1000
340 REM f=u+v => f'=u'+v'
341 REM f=u-v => f'=u'-v'
350 IF c$<>"0" THEN IF d$<>"0" THEN LET s$(z)=c$+z$(o)+d$: RETURN

```



```

360 IF c$<>"0" THEN LET s$(z)=c$: RETURN
370 IF d$<>"0" THEN LET s$(z)=z$(2 TO o)+d$: RETURN
380 LET s$(z)="0": RETURN
390 REM f=u*v => f'=u'*v+u*v'
391 REM f=u/v => f'=(u'*v-u*v')/v^2
400 IF c$="0" THEN LET t=0: LET e$=c$: GO TO 440
410 LET t=1
420 IF c$="1" THEN LET e$=b$: GO TO 440
425 LET q$=c$: GO SUB 2000: IF o<3 THEN LET c$=FN b$(c$)
428 LET g$=b$: LET q$=g$: GO SUB 2000: IF o<3 THEN LET g$=FN b$(g$)
430 LET e$=c$+"*"+g$: GO TO 440
440 IF d$="0" THEN GO TO 480
445 LET g$=c$: LET q$=c$: GO SUB 2000: IF o<3 THEN LET g$=FN b$(g$)
450 LET t=t+1: LET e$=e$(1+(t=1) TO )+z$(o(z)-2 TO o(z)-2+(1-((t=1) AND
(o(z)=3)))-1)
460 IF d$="1" THEN LET e$=e$+a$: GO TO 480
465 LET q$=a$: GO SUB 2000: IF o<3 THEN LET a$=FN b$(a$)
470 LET q$=d$: GO SUB 2000: IF o<3 THEN LET d$=FN b$(d$)
475 LET e$=e$+a$+"*"+d$
480 IF e$="0" OR o(z)=3 THEN GO TO 510
490 IF t=2 THEN LET e$=FN b$(e$)
495 LET q$=b$: GO SUB 2000: IF o<3 THEN LET b$=FN b$(b$)
500 LET e$=e$+"/" + b$+"^2"
510 LET s$(z)=e$
520 RETURN
530 REM f=u^v => f'=v*u'*u^(v-1)
531 REM f'=u^v*(u'/u+v*ln(u)*v')
540 IF d$<>"0" THEN GO TO 610
550 IF c$="0" THEN LET s$(z)=c$: RETURN
560 IF c$="1" THEN LET e$="": GO TO 590
570 LET q$=c$: GO SUB 2000: IF o<3 THEN LET c$=FN b$(c$)
580 LET e$=c$+"*"
590 IF b$<>"1" THEN LET e$=b$+"*"+e$
595 IF b$(1)="" THEN LET b$=b$(2 TO LEN b$-1)
596 GO SUB 15
597 IF d$(1)="" THEN LET d$=d$(2 TO ): GO TO 599
598 LET d$=FN b$(d$)
599 IF d$="1" THEN LET s$(z)=e$+a$: RETURN
600 IF d$="0" THEN LET s$(z)=e$+"1": RETURN
605 LET s$(z)=e$+a$+"^"+d$: RETURN
610 LET g$=a$
620 LET q$=g$: GO SUB 2000: IF o<3 THEN LET g$=FN b$(g$)
630 LET q$=c$: GO SUB 2000: IF o<3 THEN LET c$=FN b$(c$)
640 LET q$=b$: GO SUB 2000: IF o<3 THEN LET b$=FN b$(b$)
650 LET q$=d$: GO SUB 2000: IF o<3 THEN LET d$=FN b$(d$)
660 LET r$=f$(z): GO SUB 5: LET e$=r$+"*(": IF c$<>"0" THEN LET e$=e$+c$+"/" +
g$+"*"+b$+"*"
665 LET e$=e$+"ln"+FN b$(a$)
670 IF d$="1" THEN LET s$(z)=e$+"": RETURN
680 LET s$(z)=e$+"*"+d$+"": RETURN
690 REM cautarea functiilor
700 FOR i=1 TO fz
710 DIM i$(20): LET i$=w$: LET i$=i$( TO 1(i)): IF i$<>x$(i) THEN NEXT i: GO TO
910
720 LET ii=i: LET i=fz: NEXT i: IF w$(LEN w$)<>"") THEN GO TO 1000
730 REM calculeaza u'
735 LET o(z)=11
740 LET z=z+1: LET f$(z)=w$(1(ii)+1 TO 1-1): GO SUB 170: LET z=z-1
750 LET r$=f$(z): GO SUB 5: LET w$=r$: LET r$=f$(z+1): GO SUB 5: LET a$=r$: LET
r$=s$(z+1): GO SUB 5: LET c$=r$
760 IF c$="0" THEN LET s$(z)="0": RETURN
770 LET q$=c$: GO SUB 2000: IF o<3 THEN LET c$=FN b$(c$)
790 LET o=o(z): IF o>10 THEN GO TO 850
800 REM f'=u'*g(u)
810 LET r$=y$(o): GO SUB 5: LET e$=r$+a$+"": IF c$<>"1" THEN LET e$=c$+"*"+e$

```



```

820 IF o>6 THEN LET e$=e$+"^2)"
830 LET s$(z)=e$: RETURN
840 REM f'=u'/g(u)
850 LET r$=v$(o): GO SUB 5: LET e$=r$+a$+")": LET e$=c$+/"'+e$
860 IF o<19 THEN LET e$=e$+"^2)"
870 LET s$(z)=e$: RETURN
900 REM derivateza parantezele, x si constantele
910 IF w$(1)<>(" OR w$(LEN w$)<>") THEN GO TO 930
915 LET q$=w$(2 TO (LEN w$-1)): GO SUB 3000: IF n THEN LET s$(z)="0": RETURN
920 LET f$(z)=w$(2 TO (LEN w$-1)): GO TO 170
930 IF w$="x" THEN LET s$(z)="1": RETURN
940 LET s$(z)="0": RETURN
1000 PRINT " ? eroare de sintaxa": GO TO 60
2000 REM gaseste operatorul cu precedenta cea mai joasa
2001 REM proc loprec(q$) returneaza:
2002 REM p indexul operatorului
2003 REM o tipul (1=+,2=-,3=*,4=/,5=^,6=nici unul)
2004 REM l lungimea stringului
2100 LET l=LEN (q$): LET p=0: LET o=6: LET k1=0
2030 FOR i=1 TO l
2040 LET k$=q$(i)
2050 LET k1=k1-(k$="")+ (k$="(")
2060 IF k1<0 THEN GO TO 2100: REM ignora parantezele
2070 IF k$="+" OR k$="-" THEN LET p=i: LET o=1+(k$="-")
2080 IF k$="*" OR k$="/" THEN IF o>2 THEN LET p=i: LET o=3+(k$="/" )
2090 IF k$="^" THEN IF o>4 THEN LET p=i: LET o=5
2100 NEXT i
2110 IF k1 THEN GO TO 1000: REM paranteze dezechilibrate
2130 RETURN
3000 REM testeaza daca termenul s$ este un numar valid (n=1 adevarat; n=0 fals)
3030 FOR i=1 TO LEN q$
3040 LET k$=q$(i)
3050 IF k$>"/" THEN IF k$<":" THEN NEXT i: LET n=1: RETURN
3060 IF k$="-" THEN NEXT i: LET n=1: RETURN
3070 LET n=0: RETURN
4000 REM functiile standard
4010 DATA 19
4020 DATA "sin(","cos("
4030 DATA "cos(","(-1)*sin("
4040 DATA "sinh(","cosh("
4050 DATA "cosh(","sinh("
4060 DATA "exp(","exp("
4070 DATA "sqr(","0.5/sqr("
4090 DATA "tan(","(1+tan("
4100 DATA "cot(","(-1)*(1+cot("
4110 DATA "tanh(","(1-tanh("
4120 DATA "coth(","(-1)*(1-coth("
4130 DATA "arcsin(","sqr(1-("
4140 DATA "arccos(","(-1)/sqr(1-("
4150 DATA "arctan(","(1+("
4160 DATA "arccot(","(-1)/(1+("
4170 DATA "arcsinh(","sqr(1+("
4180 DATA "arccosh(","(-1)/sqr(-1+("
4190 DATA "arctanh(","(1-("
4200 DATA "arccoth(","(1-("
4210 DATA "ln(","("
5000 INPUT :: BEEP .1,40: POKE 23692,255: LET r=5090: LET resume=r: LET g=r:
PRINT INVERSE 1;"daca doriti evaluarea numerica,asteptati, altfel apasati
g.
5010 LET r$=h$: GO SUB 6000: LET h$=r$
5020 LET r$=d$: GO SUB 6000: LET d$=r$
5021 PRINT ' INVERSE 1:" dupa eroare reveniti cu GO TO g": BEEP .1,45: GO TO
5050
5025 INPUT "x=";x
5030 POKE 23692,255: PRINT "'f('x:)'=";VAL h$

```



```

5040 PRINT "f('":x;")=":VAL d$
5050 INPUT :: PRINT #1: INVERSE 1:" apasati c pentru calculator apasati q
    pentru intoarcere
5060 IF INKEY$="c" THEN GO TO 5025
5070 IF INKEY$="q" THEN GO TO 5110
5080 GO TO 5060
5090 PRINT "eroare": GO TO 5050
5100 INPUT :: PRINT #0:TAB 10: FLASH 1:"intrerupt": BEEP 1.-8
5110 INPUT :: CLS : GO TO 60
6000 RESTORE 7000: FOR i=1 TO fz
6010 READ k$,l$
6020 LET j=1
6030 LET k=j+LEN k$-1: IF k<=LEN r$ THEN IF r$(j TO k)=k$ THEN GO SUB 6100
6035 IF INKEY$="q" THEN GO TO 5100
6040 IF j<LEN r$-LEN k$+1 THEN LET j=j+1: GO TO 6030
6050 NEXT i: RETURN
6100 IF LEN k$=LEN r$ THEN LET r$=l$: RETURN
6105 IF j=1 THEN LET r$=l$+r$(j+LEN k$ TO ): RETURN
6120 IF j=LEN r$-LEN k$+1 THEN LET r$=r$( TO j-1)+l$: RETURN
6130 LET r$=r$( TO j-1)+l$+r$(j+LEN k$ TO ): RETURN
7000 DATA "arcsin(","ASN ("
7010 DATA "arccos(","ACS ("
7020 DATA "arctan(","ATN ("
7030 DATA "arccot(","FN a("
7040 DATA "sin(","SIN ("
7050 DATA "cos(","COS ("
7060 DATA "tan(","TAN ("
7070 DATA "cot(","FN c("
7080 DATA "arcsinh(","FN e("
7090 DATA "arccosh(","FN f("
7100 DATA "arctanh(","FN g("
7110 DATA "arccoth(","FN h("
7120 DATA "sinh(","FN s("
7130 DATA "cosh(","FN b("
7140 DATA "tanh(","FN t("
7150 DATA "coth(","FN d("
7160 DATA "sqr(","SQR ("
7170 DATA "exp(","EXP ("
7180 DATA "ln(","LN ("
8000 BORDER 7: PAPER 7: INK NOT PI: CLS : PRINT PAPER 6:AT 4,0:"derivarea
    simbolica a functiilor": PAPER 2: INK 7:AT 8,7:"de Octavian Pleter"
8005 PRINT "TAB 6:"prima versiune: 1982""
8010 PRINT TAB 5:"versiunea curenta: 1985""
8020 PRINT "acest program permite evaluarea numerica a functiei si derivatei"
8030 PAUSE 300: RETURN
9999 CLEAR : SAVE "difun" LINE 1

```

BIBLIOGRAFIE

1. ARENTZ, R.; GÖRLITZ, M., *Das Sinclair Spectrum ROM*, Hueber Software, München, 1984.
2. BARDEN, W., Jr., *The Z-80 Microcomputer Handbook*, Howard Sams, Indianapolis, 1978.
3. BÉNARD, J., *50 Programmes ZX Spectrum*, Editions Radio, Paris, 1983.
4. BISHOP, G., *Spectrum Interfacing and Projects*, McGraw-Hill, UK, 1983.
5. BURTON, D.P.; Dexter, A.L., *Microprocessor Systems Handbook*, Analog Devices, Norwood Mass., 1977.
6. CALLENDER, C., *Putting Your Spectrum to Work*, Interface Publications, London, 1983.

7. CĂPĂȚÎNĂ, O.: CORNEA-HASEGAN, M.: PUȘCĂ, M., *Proiectarea cu microprocesoare*, Editura Dacia, Cluj-Napoca, 1983.
8. CARRI, G., *Spectrum Shadow ROM Disassembly*, Melbourne House, 1985.
9. DANCEA, I., *Microprocesoare - Arhitectura internă, programare, aplicații*, Editura Dacia, Cluj-Napoca, 1979.
10. DICKENS, A., *Spectrum Hardware Manual*, Melbourne House, 1983.
11. DOGARU, D., *Elemente de grafică 3D*, Editura Științifică și Enciclopedică, București, 1988.
12. DOWNING, D. : M. COVINGTON, *Dictionary of Computer Terms*, Barron's, 1988.
13. EGELER, R., *ZX Spectrum Hardware*, Markt & Technik Verlag, Haar bei München, 1985.
14. HAMPSHIRE, N., *Spectrum Graphics*, Duckworth, London, 1982.
15. HARTNELL, T., *Exploring Artificial Intelligence on your Spectrum+ and Spectrum*, Interface Publications, London, 1984.
16. HEWSON, A., *20 Best Programs for the ZX Spectrum*, Hewson Consultants, Oxon, 1984.
17. JAMES, M., *An Expert Guide to the Spectrum*, Granada, London, 1984.
18. JONES, D., *Delving Deeper into your ZX Spectrum*, Interface Publications, London, 1985.
19. KRAMER, S., *The Spectrum Operating System*, MicroPress, London, 1984.
20. LAINE, D., *Machine Code Applications for the ZX Spectrum*, Interface Publications, London, 1985.
21. LANCASTER, D., *TV Typewriter Cookbook*, Howard Sams, Indianapolis, 1976.
22. LAWRENCE, D., *The Working Spectrum - A Library of Practical Subroutines and Programs*, Sunshine Books, London, 1982.
23. LOGAN, I., *Il libro del Microdrive Spectrum*, Jacopo Castelfranchi Editore, 1984.
24. LOGAN, I., *Understanding Your Spectrum*, Melbourne House, 1983.
25. LOGAN, I.; O'Hara, F., *The Complete Spectrum ROM Disassembly*, Melbourne House, 1983.
26. LORD, M., *Exploring Spectrum Basic*, Timedata, Basildon, 1982.
27. LUPU, C., *Microprocesoare. 2/4/8 biți*, Editura Militară, București, 1995.
28. LUPU, C.; TEPELEA, V.; PURICE, E., *Microprocesoare. Aplicații*, Editura Militară, București, 1982.
29. LUPU, C.; STĂNCESCU, S., *Microprocesoare - Circuite, proiectare*, Editura Militară, București, 1986.
30. MARGOLIS, A., *Troubleshooting & Repairing Personal Computers*, TAB Books, Blue Ridge Summit, PA, 1981.
31. McLEAN, I.; GORDON, J., *100 Programs for the ZX Spectrum*, Prentice Hall International, 1983.
32. MERTZ, J., *Mikrodrive-Handbuch für den ZX Spectrum*, Profisoft, Osnabrück, 1985.
33. MOORE, L., *Mastering the ZX Spectrum*, Ellis Horwood Ltd., Chichester, 1983.
34. NEWMAN, W. P.; SPROULL, R., *Principles of Interactive Computer Graphics*, 2nd Edition, McGraw Hill, 1979.
35. NORTON, P., *Inside the IBM-PC*, Brady, New York, 1988.
36. PELLIER, P., *Langage machine - Trucs et astuces sur ZX Spectrum*, Eyrolles, Paris, 1984.
37. PATRUBĂNI, M., *Totul despre... microprocesorul Z80*, 2 vol., Editura Tehnică, București, 1989.
38. PETRESCU, A.; TĂPUȘ, N.; MOISA, T.; RIZESCU, G.; HĂRĂBOR, V.; MÂRȘANU, M.; ZAMFIRESCU, P.; COSOSCHI, V.; DOBROVIE, E.; BADEA, N.; HĂRĂBOR, C., *abc de calculatoare personale și... nu doar atât*, vol. 1-2, Editura Tehnică, București, 1990.
39. ROGERS, D.; ADAMS, J.A., *Mathematical Elements for Computer Graphics*, McGraw Hill, 1976.

40. SÉHAN, J.-F., *Clefs pour le ZX Spectrum et Timex 2000*, Editions du P. S. I., Torcy Marne-la-Vallée, 1984.
41. SINCLAIR, I., *The ZX Spectrum*, Granada, 1982.
42. STEWART, I.; JONES, R., *Spectrum Machine Code*, Shiva Publishing Ltd, Nantwich, 1983.
43. TÂNĂSESCU, A.; SAVA, I., *Utilizarea calculatoarelor electronice în arhitectură și perspectiva arhitecturală*, Editura Didactică și Pedagogică, București, 1972.
44. VICKERS, S.; BRADBEER, R., *Sinclair ZX Spectrum - Introduction - Basic Programming*, Sinclair Research Ltd., Cambridge, 1982.
45. WEBB, D., *Advanced Spectrum Machine Language*, Melbourne House, 1984.
46. WILLIAMS, P., *Over the Spectrum*, Melbourne House, 1983.
47. WOODS, T., *Learn and Use Assembly Language on the ZX Spectrum*, McGraw-Hill UK, 1983.
48. ZAKS, R.; LESEA, A., *Microprocessor Interfacing Techniques*, 3rd Edition, Sybex, 1979.
49. ***, Amstrad Plc., Sinclair Computers Division, *Sinclair ZX Spectrum +3*, Cambridge, 1987.
50. ***, Întreprinderea de calculatoare electronice, *HC85 - Manual tehnic*, București, 1986.
51. ***, ITCI-FMECTC, *TIM-S - Microcalculator personal - Instrucțiuni de instalare și scheme logice*, Timisoara, 1986.
52. ***, MBL Electronics Signetics, *Integrated Circuits*, 1982.
53. ***, SGS-ATES, *Z80 CPU Instruction Set*.
54. ***, SGS-ATES, *Z80 Microprocessor Family Databook*, 3rd Edition, 1982.
55. ***, Sinclair Research Ltd., *Introduction - Spectrum 128*, Cambridge, 1986.
56. ***, Sinclair Research Ltd., *Sinclair ZX Spectrum Microdrive and Interface 1 Manual*, Cambridge, 1983.
57. ***, Sinclair Research Ltd., *Servicing Manual for ZX Spectrum*, Cambridge, 1984.
58. ***, *Computer Kontakt*, Verlag Rätz-Eberle, Bretten, colecția 1985-1987.
59. ***, *Sinclair User*, EMAP Business & Computer Publications, London, colecția 1984-1988.
60. ***, *ZX Computing*, Argus Specialist Publications, London, colecția 1983-1986.

Anexa A - TABELUL MATRICEAL PRINCIPAL AL INSTRUCȚIUNILOR Z80

#	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NOP LD BC,nn	LD (BC),A	INC BC	INC B	DEC B	LD B,n	RLCA	EX AF,AF'	ADD HL,BC	LD A,(BC)	DEC BC	INC C	DEC C	LD C,n	RRCA	
1	DJNZ d	LD DE,nn	LD (DE),A	INC DE	INC D	DEC D	LD D,n	RLA	JR d	ADD HL,DE	LD A,(DE)	DEC DE	INC E	DEC E	LD E,n	RRA
2	JR NZ,d	LD HL,nn	LD (nn),HL	INC HL	INC H	DEC H	LD H,n	DAA	JR Z,d	ADD HL,HL	LD HL,(nn)	DEC HL	INC L	DEC L	LD L,n	CPL
3	JR NC,d	LD SP,nn	LD (nn),A	INC SP	INC (HL)	DEC (HL)	LD (HL),n	SCF	JR C,d	ADD HL,SP	LD A,(nn)	DEC SP	INC A	DEC A	LD A,n	CCF
4	LD B,B	LD B,C	LD B,D	LD B,E	LD B,H	LD B,L	LD B,(HL)	LD B,A	LD C,B	LD C,C	LD C,D	LD C,E	LD C,H	LD C,L	LD C,(HL)	LD C,A
5	LD D,B	LD D,C	LD D,D	LD D,E	LD D,H	LD D,L	LD D,(HL)	LD D,A	LD E,B	LD E,C	LD E,D	LD E,E	LD E,H	LD E,L	LD E,(HL)	LD E,A
6	LD H,B	LD H,C	LD H,D	LD H,E	LD H,H	LD H,L	LD H,(HL)	LD H,A	LD L,B	LD L,C	LD L,D	LD L,E	LD L,H	LD L,L	LD L,(HL)	LD L,A
7	LD (HL),B	LD (HL),C	LD (HL),D	LD (HL),E	LD (HL),H	LD (HL),L	HALT	LD (HL),A	LD A,B	LD A,C	LD A,D	LD A,E	LD A,H	LD A,L	LD A,(HL)	LD A,A
8	ADD A,B	ADD A,C	ADD A,D	ADD A,E	ADD A,H	ADD A,L	ADD A,(HL)	ADD A,A	ADC A,B	ADC A,C	ADC A,D	ADC A,E	ADC A,H	ADC A,L	ADC A,(HL)	ADC A,A
9	SUB B	SUB C	SUB D	SUB E	SUB H	SUB L	SUB (HL)	SUB A	SBC A,B	SBC A,C	SBC A,D	SBC A,E	SBC A,H	SBC A,L	SBC A,(HL)	SBC A,A
A	AND B	AND C	AND D	AND E	AND H	AND L	AND (HL)	AND A	XOR B	XOR C	XOR D	XOR E	XOR H	XOR L	XOR (HL)	XOR A
B	OR B	OR C	OR D	OR E	OR H	OR L	OR (HL)	OR A	CP B	CP C	CP D	CP E	CP H	CP L	CP (HL)	CP A
C	RET NZ	POP BC	JP NZ,nn	JP nn	CALL NZ,nn	PUSH BC	ADD A,n	RST 0	RET Z	RET	JP Z,nn	PREFIX ▲	CALL Z,nn	CALL nn	ADC A,n	RST 8
D	RET NC	POP DE	JP NC,nn	OUT (n),A	CALL NC,nn	PUSH DE	SUB n	RST 16	RET C	EXX	JP C,nn	IN A,(n)	CALL C,nn	PREFIX IX	SBC A,n	RST 24
E	RET PO	POP HL	JP PO,nn	EX (SP),HL	CALL PO,nn	PUSH HL	AND n	RST 32	RET PE	JP (HL)	JP PE,nn	EX DE,HL	CALL PE,nn	PREFIX ▼	XOR n	RST 40
F	RET P	POP AF	JP P,nn	DI	CALL P,nn	PUSH AF	OR n	RST 48	RET M	LD SP,HL	JP M,nn	EI	CALL M,nn	PREFIX IY	CP n	RST 56

LEGENDA:

instrucțiunea suportă adresarea indexată, prin prefixul DDh sau FDh; în aceste cazuri, în loc de HL vom avea IX, respectiv IY, iar în loc de (HL) vom avea (IX+d), respectiv (IY+d)

Anexa B - TABELUL MATRICEAL AL INSTRUCȚIUNILOR EXTINSE CU PREFIXUL #CB

 0 1 2 3 4 5 6 7 8 9 A B C D E F																
0	RLC B	RLC C	RLC D	RLC E	RLC H	RLC L	RLC (HL)	RLC A	RRC B	RRC C	RRC D	RRC E	RRC H	RRC L	RRC (HL)	RRC A
1	RL B	RL C	RL D	RL E	RL H	RL L	RL (HL)	RL A	RR B	RR C	RR D	RR E	RR H	RR L	RR (HL)	RR A
2	SLA B	SLA C	SLA D	SLA E	SLA H	SLA L	SLA (HL)	SLA A	SRA B	SRA C	SRA D	SRA E	SRA H	SRA L	SRA (HL)	SRA A
3	SLL B	SLL C	SLL D	SLL E	SLL H	SLL L	SLL (HL)	SLL A	SRL B	SRL C	SRL D	SRL E	SRL H	SRL L	SRL (HL)	SRL A
4	BIT 0,B	BIT 0,C	BIT 0,D	BIT 0,E	BIT 0,H	BIT 0,L	BIT 0,(HL)	BIT 0,A	BIT 1,B	BIT 1,C	BIT 1,D	BIT 1,E	BIT 1,H	BIT 1,L	BIT 1,(HL)	BIT 1,A
5	BIT 2,B	BIT 2,C	BIT 2,D	BIT 2,E	BIT 2,H	BIT 2,L	BIT 2,(HL)	BIT 2,A	BIT 3,B	BIT 3,C	BIT 3,D	BIT 3,E	BIT 3,H	BIT 3,L	BIT 3,(HL)	BIT 3,A
6	BIT 4,B	BIT 4,C	BIT 4,D	BIT 4,E	BIT 4,H	BIT 4,L	BIT 4,(HL)	BIT 4,A	BIT 5,B	BIT 5,C	BIT 5,D	BIT 5,E	BIT 5,H	BIT 5,L	BIT 5,(HL)	BIT 5,A
7	BIT 6,B	BIT 6,C	BIT 6,D	BIT 6,E	BIT 6,H	BIT 6,L	BIT 6,(HL)	BIT 6,A	BIT 7,B	BIT 7,C	BIT 7,D	BIT 7,E	BIT 7,H	BIT 7,L	BIT 7,(HL)	BIT 7,A
8	RES 0,B	RES 0,C	RES 0,D	RES 0,E	RES 0,H	RES 0,L	RES 0,(HL)	RES 0,A	RES 1,B	RES 1,C	RES 1,D	RES 1,E	RES 1,H	RES 1,L	RES 1,(HL)	RES 1,A
9	RES 2,B	RES 2,C	RES 2,D	RES 2,E	RES 2,H	RES 2,L	RES 2,(HL)	RES 2,A	RES 3,B	RES 3,C	RES 3,D	RES 3,E	RES 3,H	RES 3,L	RES 3,(HL)	RES 3,A
A	RES 4,B	RES 4,C	RES 4,D	RES 4,E	RES 4,H	RES 4,L	RES 4,(HL)	RES 4,A	RES 5,B	RES 5,C	RES 5,D	RES 5,E	RES 5,H	RES 5,L	RES 5,(HL)	RES 5,A
B	RES 6,B	RES 6,C	RES 6,D	RES 6,E	RES 6,H	RES 6,L	RES 6,(HL)	RES 6,A	RES 7,B	RES 7,C	RES 7,D	RES 7,E	RES 7,H	RES 7,L	RES 7,(HL)	RES 7,A
C	SET 0,B	SET 0,C	SET 0,D	SET 0,E	SET 0,H	SET 0,L	SET 0,(HL)	SET 0,A	SET 1,B	SET 1,C	SET 1,D	SET 1,E	SET 1,H	SET 1,L	SET 1,(HL)	SET 1,A
D	SET 2,B	SET 2,C	SET 2,D	SET 2,E	SET 2,H	SET 2,L	SET 2,(HL)	SET 2,A	SET 3,B	SET 3,C	SET 3,D	SET 3,E	SET 3,H	SET 3,L	SET 3,(HL)	SET 3,A
E	SET 4,B	SET 4,C	SET 4,D	SET 4,E	SET 4,H	SET 4,L	SET 4,(HL)	SET 4,A	SET 5,B	SET 5,C	SET 5,D	SET 5,E	SET 5,H	SET 5,L	SET 5,(HL)	SET 5,A
F	SET 6,B	SET 6,C	SET 6,D	SET 6,E	SET 6,H	SET 6,L	SET 6,(HL)	SET 6,A	SET 7,B	SET 7,C	SET 7,D	SET 7,E	SET 7,H	SET 7,L	SET 7,(HL)	SET 7,A

LEGENDA:

instrucțiunea suportă adresarea indexată, prin prefixul DDh sau FDh; în aceste cazuri, în loc de HL vom avea IX, respectiv IY, iar în loc de (HL) vom avea (IX+d), respectiv (IY+d) instrucțiuni "ascunse", omise din documentația Zilog datorită executării lor defectuoase; utilizarea lor trebuie evitată

**Anexa C - TABELUL MATRICEAL AL INSTRUCȚIUNILOR
EXTINSE CU PREFIXUL #ED**

PREFIX ▼	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0																
1																
2																
3																
4	IN B,(C)	OUT (C),B	SBC HL,BC	LD (nn),BC	NEG	RETN	IM 0	LD I,A	IN C,(C)	OUT (C),C	ADC HL,BC	LD BC,(nn)		RETI		LD R,A
5	IN D,(C)	OUT (C),D	SBC HL,DE	LD (nn),DE			IM 1	LD A,I	IN E,(C)	OUT (C),E	ADC HL,DE	LD DE,(nn)			IM 2	LD A,R
6	IN H,(C)	OUT (C),H	SBC HL,HL	LD (nn),HL				RRD	IN L,(C)	OUT (C),L	ADC HL,HL	LD HL,(nn)				RLD
7	IN F,(C)	OUT (C),F	SBC HL,SP	LD (nn),SP					IN A,(C)	OUT (C),A	ADC HL,SP	LD SP,(nn)				
8																
9																
A	LDI	CPI	INI	OUTI					LDD	CPD	IND	OUTD				
B	LDIR	CPIR	INIR	OTIR					LDDRC	CPDR	INDR	OTDR				
C																
D																
E																
F																

LEGENDA: instrucțiuni "ascunse", omise din documentația Zilog datorită executării lor defectuoase;
utilizarea lor trebuie evitată

Anexa D - TABELUL INSTRUCȚIUNILOR Z80
ÎN ORDINEA VITEZEI DE EXECUȚIE

4	LD r,r EX DE,HL ADD A,r SUB r AND r XOR r INC r DAA CCF NOP EI RR(C)A JP (HL) EXX EX AF,AF' ADC A,r SBC A,r OR r CP r DEC r CPL SCF HALT DI RL(C)A
5	RET cc *
6	LD SP,HL INC BC INC DE INC HL INC SP DEC BC DEC DE DEC HL DEC SP
7	LD r,n LD (HL),r LD (BC),A LD (DE),A ADD ADC SBC A,n SUB AND OR XOR CP n JR cc,d * LD r,(HL) LD A,(BC) LD A,(DE) ADD ADC SBC A,(HL) SUB AND OR XOR CP (HL)
8	NEG RRC r RR r SRA r SRL r SET b,r JP (IX) * IM 0 1 2 RLC r RL r SLA r BIT b,r RES b,r DJNZ d *
9	LD A,I LD A,R LD I,A LD R,A
10	LD (HL),n LD BC,nn LD HL,nn POP BC POP HL INC IX JP nn CALL cc,nn * LD SP,IX LD DE,nn LD SP,nn POP DE POP AF DEC IX JP cc,nn RET
11	PUSH BC PUSH HL INC (HL) ADD HL,BC ADD HL,HL IN A,(n) RST z PUSH DE PUSH AF DEC (HL) ADD HL,DE ADD HL,SP OUT (n),A RET cc
12	BIT b,(HL) JR d IN r,(C) JR cc,d JR cc,d OUT (C),r
13	LD A,(nn) DJNZ d LD (nn),A
14	LD IX,nn POP IX RETI RETN
15	PUSH IX ADC SBC HL,BC ADC SBC HL,HL ADD IX,BC ADD IX,IX RR(C) (HL) SRA SLA SRL (HL) ADC SBC HL,DE ADC SBC HL,SP ADD IX,DE ADD IX,SP RL(C) (HL) SET RES b,(HL)
16	LD HL,(nn) LDI CPI LDIR * CPIR * INI OUTI INIR * OTIR * LD (nn),HL LDD CPD LDDR * CPDR * IND OUTD INDR * OTDR *
17	CALL nn CALL cc,nn
18	RRD RLD
19	LD r,(IX+d) EX (SP),HL ADD A,(IX+d) SUB (IX+d) AND (IX+d) XOR (IX+d) LD (IX+d),r LD (IX+d),n ADC A,(IX+d) SBC A,(IX+d) OR (IX+d) CP (IX+d)
20	LD BC,(nn) LD DE,(nn) LD HL,(nn) ** LD SP,(nn) LD IX,(nn) BIT b,(IX+d) LD (nn),BC LD (nn),DE LD (nn),HL ** LD (nn),SP LD (nn),IX
21	LDIR CPIR INIR OTIR LDDR CPDR INDR OTDR
23	EX (SP),IX INC (IX+d) RR(C) (IX+d) SRA (IX+d) SRL (IX+d) SET b,(IX+d) RL(C) (IX+d) SLA (IX+d) RES b,(IX+d)

NOTE: * dacă nu se execută salt ** instrucțiuni cu prefix (există omonime fără prefix)
oriunde apare IX se subînțelege că poate apărea și IY

Anexa E - TABELUL CARACTERELOR ASCII SPECTRUM

Zec	Hexa	ASCII	ASCII Spectrum	Z80	Z80 după CBh	Z80 după EDh
0	00h	NUL		NOP	RLC B	
1	01h	SOH		LD BC,n	RLC C	
2	02h	STX		LD (BC),A	RLC D	
3	03h	ETX		INC BC	RLC E	
4	04h	EOT		INC B	RLC H	
5	05h	ENQ		DEC B	RLC L	
6	06h	ACK	print ,	LD B,n	RLC (HL)	
7	07h	BEL	edit	RLCA	RLC A	
8	08h	BS	left	EX AF,AF'	RRC B	
9	09h	HT	right	ADD HL,BC	RRC C	
10	0Ah	LF	down	LD A,(BC)	RRC D	
11	0Bh	VT	up	DEC BC	RRC E	
12	0Ch	FF	delete	INC C	RRC H	
13	0Dh	CR	enter	DEC C	RRC L	
14	0Eh	SO	numeric	LD C,n	RRC (HL)	
15	0Fh	SI		RRCA	RRC A	
16	10h	DLE	ink	DJNZ d	RL B	
17	11h	DC1	paper	LD DE,n	RL C	
18	12h	DC2	flash	LD (DE),A	RL D	
19	13h	DC3	bright	INC DE	RL E	
20	14h	DC4	inverse	INC D	RL H	
21	15h	NAK	over	DEC D	RL L	
22	16h	SYN	at	LD D,n	RL (HL)	
23	17h	ETB	tab	RLA	RL A	
24	18h	CAN		JR d	RR B	
25	19h	EM		ADD HL,DE	RR C	
26	1Ah	SUB		LD A,(DE)	RR D	
27	1Bh	ESC		DEC DE	RR E	
28	1Ch	FS		INC E	RR H	

Tabelul caracterelor ASCII Spectrum (continuare)

Zec	Hexa	ASCII	ASCII Spectrum	Z80	Z80 după CBh	Z80 după EDh
29	1Dh	GS		DEC E	RR L	
30	1Eh	RS		LD E,n	RR (HL)	
31	1Fh	US		RR A	RR A	
32	20h	SP	SPACE	JR NZ,d	SLA B	
33	21h	!	!	LD HL,nn	SLA C	
34	22h	"	"	LD (nn),HL	SLA D	
35	23h	#	#	INC HL	SLA E	
36	24h	\$	\$	INC H	SLA H	
37	25h	%	%	DEC H	SLA L	
38	26h	&	&	LD H,n	SLA (HL)	
39	27h	'	'	DAA	SLA A	
40	28h	(	(	JR Z,d	SRA B	
41	29h	)	)	ADD HL,HL	SRA C	
42	2Ah	*	*	LD HL,(nn)	SRA D	
43	2Bh	+	+	DEC HL	SRA E	
44	2Ch	,	,	INC L	SRA H	
45	2Dh	-	-	DEC L	SRA L	
46	2Eh	.	.	LD L,n	SRA (HL)	
47	2Fh	/	/	CPL	SRA A	
48	30h	0	0	JR NC,d		
49	31h	1	1	LD SP,nn		
50	32h	2	2	LD (nn),A		
51	33h	3	3	INC SP		
52	34h	4	4	INC (HL)		
53	35h	5	5	DEC (HL)		
54	36h	6	6	LD (HL),n		
55	37h	7	7	SCF		
56	38h	8	8	JR C,d	SRL B	
57	39h	9	9	ADD HL,SP	SRL C	

Tabelul caracterelor ASCII Spectrum (continuare)

Zec	Hexa	ASCII	ASCII Spectrum	Z80	Z80 după CBh	Z80 după EDh
58	3Ah	:	:	LD A. (nn)	SRL D	
59	3Bh	;	;	DEC SP	SRL E	
60	3Ch	<	<	INC A	SRL H	
61	3Dh	=	=	DEC A	SRL L	
62	3Eh	>	>	LD A.n	SRL (HL)	
63	3Fh	?	?	CCF	SRL A	
64	40h	@	@	LD B.B	BIT 0.B	IN B.(C)
65	41h	A	A	LD B.C	BIT 0.C	OUT (C).B
66	42h	B	B	LD B.D	BIT 0.D	SBC HL.BC
67	43h	C	C	LD B.E	BIT 0.E	LD (nn).BC
68	44h	D	D	LD B.H	BIT 0.H	NEG
69	45h	E	E	LD B.L	BIT 0.L	RETN
70	46h	F	F	LD B.(HL)	BIT 0.(HL)	IM 0
71	47h	G	G	LD B.A	BIT 0.A	LD I.A
72	48h	H	H	LD C.B	BIT 1.B	IN C.(C)
73	49h	I	I	LD C.C	BIT 1.C	OUT (C).C
74	4Ah	J	J	LD C.D	BIT 1.D	ADC HL.BC
75	4Bh	K	K	LD C.E	BIT 1.E	LD BC.(nn)
76	4Ch	L	L	LD C.H	BIT 1.H	
77	4Dh	M	M	LD C.L	BIT 1.L	RETI
78	4Eh	N	N	LD C.(HL)	BIT 1.(HL)	
79	4Fh	O	O	LD C.A	BIT 1.A	LD R.A
80	50h	P	P	LD D.B	BIT 2.B	IN D.(C)
81	51h	Q	Q	LD D.C	BIT 2.C	OUT (C).D
82	52h	R	R	LD D.D	BIT 2.D	SBC HL.DE
83	53h	S	S	LD D.E	BIT 2.E	LD (nn).DE
84	54h	T	T	LD D.H	BIT 2.H	
85	55h	U	U	LD D.L	BIT 2.L	
86	56h	V	V	LD D.(HL)	BIT 2.(HL)	IM 1

Tabelul caracterelor ASCII Spectrum (continuare)

Zec	Hexa	ASCII	ASCII Spectrum	Z80	Z80 după CBh	Z80 după EDh
87	57h	W	W	LD D,A	BIT 2,A	LD A,I
88	58h	X	X	LD E,B	BIT 3,B	IN E,(C)
89	59h	Y	Y	LD E,C	BIT 3,C	OUT (C),E
90	5Ah	Z	Z	LD E,D	BIT 3,D	ADC HL,DE
91	5Bh	[	[	LD E,E	BIT 3,E	LD DE,(nn)
92	5Ch	\	\	LD E,H	BIT 3,H	
93	5Dh	]	]	LD E,L	BIT 3,L	
94	5Eh	^	^	LD E,(HL)	BIT 3,(HL)	IM 2
95	5Fh	_	_	LD E,A	BIT 3,A	LD A,R
96	60h	`	£	LD H,B	BIT 4,B	IN H,(C)
97	61h	a	a	LD H,C	BIT 4,C	OUT (C),H
98	62h	b	b	LD H,D	BIT 4,D	SBC HL,HL
99	63h	c	c	LD H,E	BIT 4,E	LD (nn),HL
100	64h	d	d	LD H,H	BIT 4,H	
101	65h	e	e	LD H,L	BIT 4,L	
102	66h	f	f	LD H,(HL)	BIT 4,(HL)	
103	67h	g	g	LD H,A	BIT 4,A	RRD
104	68h	h	h	LD L,B	BIT 5,B	IN L,(C)
105	69h	i	i	LD L,C	BIT 5,C	OUT (C),L
106	6Ah	j	j	LD L,D	BIT 5,D	ADC HL,HL
107	6Bh	k	k	LD L,E	BIT 5,E	LD HL,(nn)
108	6Ch	l	l	LD L,H	BIT 5,H	
109	6Dh	m	m	LD L,L	BIT 5,L	
110	6Eh	n	n	LD L,(HL)	BIT 5,(HL)	
111	6Fh	o	o	LD L,A*	BIT 5,A	RLD
112	70h	p	p	LD (HL),B	BIT 6,B	IN F,(C)
113	71h	q	q	LD (HL),C	BIT 6,C	
114	72h	r	r	LD (HL),D	BIT 6,D	SBC HL,SP
115	73h	s	s	LD (HL),E	BIT 6,E	LD (nn),SP

Tabelul caracterelor ASCII *Spectrum* (continuare)

Zec	Hexa	ASCII	ASCII Spectrum	Z80	Z80 după CBh	Z80 după EDh
116	74h	t	t	LD (HL),H	BIT 6,H	
117	75h	u	u	LD (HL),L	BIT 6,L	
118	76h	v	v	HALT	BIT 6,(HL)	
119	77h	w	w	LD (HL),A	BIT 6,A	
120	78h	x	x	LD A,B	BIT 7,B	IN A,(C)
121	79h	y	y	LD A,C	BIT 7,C	OUT (C),A
122	7Ah	z	z	LD A,D	BIT 7,D	ADC HL,SP
123	7Bh	{	{	LD A,E	BIT 7,E	LD SP,(nn)
124	7Ch			LD A,H	BIT 7,H	
125	7Dh	}	}	LD A,L	BIT 7,L	
126	7Eh	~	~	LD A,(HL)	BIT 7,(HL)	
127	7Fh	DEL	(C)	LD A,A	BIT 7,A	
128	80h			ADD A,B	RES 0,B	
129	81h			ADD A,C	RES 0,C	
130	82h			ADD A,D	RES 0,D	
131	83h			ADD A,E	RES 0,E	
132	84h			ADD A,H	RES 0,H	
133	85h			ADD A,L	RES 0,L	
134	86h			ADD A,(HL)	RES 0,(HL)	
135	87h			ADD A,A	RES 0,A	
136	88h			ADC A,B	RES 1,B	
137	89h			ADC A,C	RES 1,C	
138	8Ah			ADC A,D	RES 1,D	
139	8Bh			ADC A,E	RES 1,E	
140	8Ch			ADC A,H	RES 1,H	
141	8Dh			ADC A,L	RES 1,L	
142	8Eh			ADC A,(HL)	RES 1,(HL)	
143	8Fh			ADC A,A	RES 1,A	

Tabelul caracterelor ASCII *Spectrum* (continuare)

Zec	Hexa	ASCII	ASCII Spectrum	Z80	Z80 după CBh	Z80 după EDh
144	90h		UDG A	SUB B	RES 2,B	
145	91h		UDG B	SUB C	RES 2,C	
146	92h		UDG C	SUB D	RES 2,D	
147	93h		UDG D	SUB E	RES 2,E	
148	94h		UDG E	SUB H	RES 2,H	
149	95h		UDG F	SUB L	RES 2,L	
150	96h		UDG G	SUB (HL)	RES 2,(HL)	
151	97h		UDG H	SUB A	RES 2,A	
152	98h		UDG I	SBC A,B	RES 3,B	
153	99h		UDG J	SBC A,C	RES 3,C	
154	9Ah		UDG K	SBC A,D	RES 3,D	
155	9Bh		UDG L	SBC A,E	RES 3,E	
156	9Ch		UDG M	SBC A,H	RES 3,H	
157	9Dh		UDG N	SBC A,L	RES 3,L	
158	9Eh		UDG O	SBC A,(HL)	RES 3,(HL)	
159	9Fh		UDG P	SBC A,A	RES 3,A	
160	A0h		UDG Q	AND B	RES 4,B	LDI
161	A1h		UDG R	AND C	RES 4,C	CPI
162	A2h		UDG S	AND D	RES 4,D	INI
163	A3h		UDG T	AND E	RES 4,E	OUTI
164	A4h		UDG U	AND H	RES 4,H	
165	A5h		RND	AND L	RES 4,L	
166	A6h		INKEY\$	AND (HL)	RES 4,(HL)	
167	A7h		PI	AND A	RES 4,A	
168	A8h		FN	XOR B	RES 5,B	LDD
169	A9h		POINT	XOR C	RES 5,C	CPD
170	AAh		SCREEN\$	XOR D	RES 5,D	IND
171	ABh		ATTR	XOR E	RES 5,E	OUTD
172	ACh		AT	XOR H	RES 5,H	
173	ADh		TAB	XOR L	RES 5,L	
174	A Eh		VAL\$	XOR (HL)	RES 5,(HL)	

Tabelul caracterelor ASCII Spectrum (continuare)

Zec	Hexa	ASCII	ASCII Spectrum	Z80	Z80 după CBh	Z80 după EDh
175	AFh		CODE	XOR A	RES 5,A	
176	B0h		VAL	OR B	RES 6,B	LDIR
177	B1h		LEN	OR C	RES 6,C	CPIR
178	B2h		SIN	OR D	RES 6,D	INIR
179	B3h		COS	OR E	RES 6,E	OTIR
180	B4h		TAN	OR H	RES 6,H	
181	B5h		ASN	OR L	RES 6,L	
182	B6h		ACS	OR (HL)	RES 6,(HL)	
183	B7h		ATN	OR A	RES 6,A	
184	B8h		LN	CP B	RES 7,B	LDDR
185	B9h		EXP	CP C	RES 7,C	CPDR
186	BAh		INT	CP D	RES 7,D	INDR
187	BBh		SQR	CP E	RES 7,E	OTDR
188	BCh		SGN	CP H	RES 7,H	
189	BDh		ABS	CP L	RES 7,L	
190	BEh		PEEK	CP (HL)	RES 7,(HL)	
191	BFh		IN	CP A	RES 7,A	
192	C0h		USR	RET NZ	SET 0,B	
193	C1h		STR\$	POP BC	SET 0,C	
194	C2h		CHR\$	JP NZ,nn	SET 0,D	
195	C3h		NOT	JP nn	SET 0,E	
196	C4h		BIN	CALL NZ,nn	SET 0,H	
197	C5h		OR	PUSH BC	SET 0,L	
198	C6h		AND	ADD A,n	SET 0,(HL)	
199	C7h		<=	RST 0	SET 0,A	
200	C8h		>=	RET Z	SET 1,B	
201	C9h		<>	RET	SET 1,C	
202	CAh		LINE	JP Z,nn	SET 1,D	
203	CBh		THEN	prefix	SET 1,E	
204	CCh		TO	CALL Z,nn	SET 1,H	
205	CDh		STEP	CALL nn	SET 1,L	

Tabelul caracterelor ASCII Spectrum (continuare)

Zec	Hexa	ASCII	ASCII Spectrum	Z80	Z80 după CBh	Z80 după EDh
206	CEh		DEF FN	ADC A,n	SET 1,(HL)	
207	CFh		CAT	RST 8	SET 1,A	
208	D0h		FORMAT	RET NC	SET 2,B	
209	D1h		MOVE	POP DE	SET 2,C	
210	D2h		ERASE	JP NC,nn	SET 2,D	
211	D3h		OPEN #	OUT (n),A	SET 2,E	
212	D4h		CLOSE #	CALL NC,nn	SET 2,H	
213	D5h		MERGE	PUSH DE	SET 2,L	
214	D6h		VERIFY	SUB n	SET 2,(HL)	
215	D7h		BEEP	RST 16	SET 2,A	
216	D8h		CIRCLE	RET C	SET 3,B	
217	D9h		INK	EXX	SET 3,C	
218	DAh		PAPER	JP C,nn	SET 3,D	
219	DBh		FLASH	IN A,(n)	SET 3,E	
220	DCh		BRIGHT	CALL C,nn	SET 3,H	
221	DDh		INVERSE	prefix IX	SET 3,L	
222	DEh		OVER	SBC A,n	SET 3,(HL)	
223	DFh		OUT	RST 24	SET 3,A	
224	E0h		LPRINT	RET PO	SET 4,B	
225	E1h		LLIST	POP HL	SET 4,C	
226	E2h		STOP	JP PO,nn	SET 4,D	
227	E3h		READ	EX (SP),HL	SET 4,E	
228	E4h		DATA	CALL PO,nn	SET 4,H	
229	E5h		RESTORE	PUSH HL	SET 4,L	
230	E6h		NEW	AND n	SET 4,(HL)	
231	E7h		BORDER	RST 32	SET 4,A	
232	E8h		CONT	RET PE	SET 5,B	
233	E9h		DIM	JP (HL)	SET 5,C	
234	EAh		REM	JP PE,nn	SET 5,D	
235	EBh		FOR	EX DE,HL	SET 5,E	
236	ECh		GO TO	CALL PE,nn	SET 5,H	

Tabelul caracterelor ASCII Spectrum (continuare)

Zec	Hexa	ASCII	ASCII Spectrum	Z80	Z80 după CBh	Z80 după EDh
237	EDh		GO SUB	prefix	SET 5,L	
238	EEh		INPUT	XOR n	SET 5,(HL)	
239	EFh		LOAD	RST 40	SET 5,A	
240	F0h		LIST	RET P	SET 6,B	
241	F1h		LET	POP AF	SET 6,C	
242	F2h		PAUSE	JP P,nn	SET 6,D	
243	F3h		NEXT	DI	SET 6,E	
244	F4h		POKE	CALL P,nn	SET 6,H	
245	F5h		PRINT	PUSH AF	SET 6,L	
246	F6h		PLOT	OR n	SET 6,(HL)	
247	F7h		RUN	RST 48	SET 6,A	
248	F8h		SAVE	RET M	SET 7,B	
249	F9h		RANDOMZ	LD SP,HL	SET 7,C	
250	FAh		IF	JP M,nn	SET 7,D	
251	FBh		CLS	EI	SET 7,E	
252	FCh		DRAW	CALL M,nn	SET 7,H	
253	FDh		CLEAR	prefix IY	SET 7,L	
254	FEh		RETURN	CP n	SET 7,(HL)	
255	FFh		COPY	RST 56	SET 7,A	

Anexa F - TABELUL VARIABILELOR DE SISTEM

Adresa în zecimal	Adresa în hexa	Numele	Lungimea	Semnificația (în paranteze valoarea implicită) < în paranteze unghiulare adresa subrutinei din ROM unde se folosește variabila respec- tivă >																																
23552	#5C00	K_STATE	8	Variabile folosite în citirea tastaturii: <#02BF, #1187> K_STATE _{0/4} = numărul tastei apăsată; K_STATE _{1/5} = contor 5->0 pentru durată; K_STATE _{2/6} = (REPPER) sau (REPDEL); K_STATE _{3/7} = codul caracterului.																																
23560	#5C08	LAST_K	1	Codul ultimei taste apăsată. <#02BF >																																
23561	#5C09	REPDEL	1	Durata cât trebuie ținută apăsată o tastă pentru a se repeta automat. (0,7 sec) <#02BF, #11B7>																																
23562	#5C0A	REPPER	1	Perioada de repetare automată a unei taste. (0,1 sec) <#0310>																																
23563	#5C0B	DEFADD	2	Adresa argumentelor din paranteze ale lui DEF FN. <#27BD>																																
23565	#5C0D	K_DATA	1	Al doilea parametru la caracterele de control al culorii introduse de la tastatura. <#10A8>																																
23566	#5C0E	TVDATA	2	TVDATA ₁ = caracterul de control; TVDATA ₂ = primul operand după caracterul de control (dacă sunt 2 operanți, cel de-al doilea este stocat în registrul A <#0A60>																																
23568	#5C10	STRMS	38	Deplasamentul față de (CHANS) al adresei informației de canal. Valorile inițiale pentru căile de la #FD la #03: <table><tr><td>canalul</td><td>adresa</td><td>conținutul</td><td>tipul</td></tr><tr><td>#FD</td><td>#5C10</td><td>#0100</td><td>K</td></tr><tr><td>#FE</td><td>#5C12</td><td>#0600</td><td>S</td></tr><tr><td>#FF</td><td>#5C14</td><td>#0B00</td><td>R</td></tr><tr><td>#00</td><td>#5C16</td><td>#0100</td><td>K</td></tr><tr><td>#01</td><td>#5C18</td><td>#0100</td><td>K</td></tr><tr><td>#02</td><td>#5C20</td><td>#0600</td><td>S</td></tr><tr><td>#03</td><td>#5C22</td><td>#1000</td><td>P</td></tr></table> #04÷#0F #5C24÷#5C34#0000 -	canalul	adresa	conținutul	tipul	#FD	#5C10	#0100	K	#FE	#5C12	#0600	S	#FF	#5C14	#0B00	R	#00	#5C16	#0100	K	#01	#5C18	#0100	K	#02	#5C20	#0600	S	#03	#5C22	#1000	P
canalul	adresa	conținutul	tipul																																	
#FD	#5C10	#0100	K																																	
#FE	#5C12	#0600	S																																	
#FF	#5C14	#0B00	R																																	
#00	#5C16	#0100	K																																	
#01	#5C18	#0100	K																																	
#02	#5C20	#0600	S																																	
#03	#5C22	#1000	P																																	
23606	#5C36	CHARS	2	Adresa-256 a setului de caractere																																

Tabelul variabilelor de sistem (continuare)

Adresa în zecimal	Adresa în hexa	Numele	Lungimea	Semnificația (în paranteze valoarea implicită) < în paranteze unghiulare adresa subrutinei din ROM unde se folosește variabila respec- tivă >
				(#3C00=15360). Adresa implicită a primu- lui caracter (blanc) este 15616, în ROM.
23608	#5C38	RASP	1	Durata semnalului sonor de avertizare. (64) <#11B7>
23609	#5C39	PIP	1	Lungimea semnalului sonor "clic" la apăsarea unei taste. (00 = minimum)
23610	#5C3A	ERR_NR	1	Codul erorii - 1. (255) IY este încărcat cu această adresă.
23611	#5C3B	FLAGS	1	Grup de 8 variabile de 1 bit: bit 0=0: se tipărește un blank înaintea unui token <#187D> bit 1=0: nu se folosește imprimanta <#1646> bit 2=0: editorul este în modul K <#1937> bit 3=0: modul K al cursorului este selectat <#1326> bit 5=0: nu s-a apăsă o nouă tastă <#1303> bit 6=0: rezultatul este un șir <#1C8C> bit 7=1: se execută o linie (syntax flag) <#12CF>
23612	#5C3C	TV_FLAGS	1	Grup de 8 variabile de 1 bit: bit 0=1: se folosește canalul tastaturii <#1634> bit 3=1: consideră că modul s-a schimbat <#15D4> bit 4=1: listing automat <#1835> bit 5=1: partea de jos a ecranului va fi ștersă <#1219>
23613	#5C3D	ERR_SP	2	Adresa din stivă (SP) care va fi folosită ca punct de întoarcere în caz de eroare. <#0F2C, #11B7, #1EAC, #1EED, #1F23, #2089>
23615	#5C3F	LIST_SP	2	Adresa de revenire după un listing automat. <#1795>

Tabelul variabilelor de sistem (continuare)

Adresa în zecimal	Adresa în hexa	Numele	Lungimea	Semnificația (în paranteze valoarea implicită) < în paranteze unghiulare adresa subrutinei din ROM unde se folosește variabila respec- tivă >
23617	#5C41	MODE	1	Specifică tipul de cursor: = 1: modul E; > 1: modul G; < 1: modulurile K/I/C (+FLAGS ₁ +FLAGS ₂) <#02BF. #0F2C. #0F81. #18E1>
23618	#5C42	NEWPPC	2	Numărul liniei la care se face saltul GO TO. <#0808. #1D03. #1E67>
23620	#5C44	NSPPC	1	Numărul instrucțiunii din linie la care se face saltul GO TO. <#0808. #1D03. #1E67>
23621	#5C45	PPC	2	Numărul liniei în curs de execuție. <#1D03. #1EED>
23623	#5C47	SUBPPC	1	Numărul instrucțiunii din linie în curs de execuție. <#1D03. #1EED>
23624	#5C48	BORDCR	1	Atributele din partea de jos a ecranului (aceiași border și paper) <#053F. #0D4D. #0E44>
23625	#5C49	E_PPC	2	Numărul liniei curente, care conține cursorul de editare >. <#0FA9. #1795>
23627	#5C4B	VARS	2	Adresa variabilelor Basic. <#0605. #0808. #08B6. #11B7. #1EAC>
23629	#5C4D	DEST	2	Adresa variabilei în curs de atribuire, dacă există. <#2AEE>
23631	#5C4F	CHANS	2	Adresa datelor despre canale. <#11B7>
23633	#5C51	CURCHL	2	Adresa informației curente utilizată pentru intrări-ieșiri. <#0A6D. #0D6B. #0DAF, #0FA9. #361F>
23635	#5C53	PROG	2	Adresa programului Basic. (23755) <#0605. #08B6. #092C. #11B7. #196E>

Tabelul variabilelor de sistem (continuare)

Adresa în zecimal	Adresa în hexa	Numele	Lungimea	Semnificația (în paranteze valoarea implicită) <în paranteze unghiulare adresa subrutinei din ROM unde se folosește variabila respec- tivă>
23637	#5C55	NXTLIN	2	Adresa următoarei linii din program. <#1BB2, #1D03>
23639	#5C57	DATADD	2	Adresa terminatorului ultimului obiect din lista DATA (, : sau CR). <#11B7, #1DEC, #1E42>
23641	#5C59	E_LINE	2	Adresa zonei de lucru a editorului. <#0605, #0808, #11B7, #19FB, #2AEE, #2C02>
23643	#5C5B	K_CUR	2	Adresa cursorului din ecran. <#0F2C, #0F81, #18E1, #2089, #361F>
23645	#5C5D	CH_ADD	2	Adresa următorului caracter de interpretat <#0008, #0018, #0074, #19FB, #1D03, #2089, #35DE>.
23647	#5C5F	X_PTR	2	Adresa - 1 a caracterului la care a survenit o eroare de sintaxă marcată cu ? <#0808, #092C, #1DEC, #2089>
23649	#5C61	WORKSP	2	Adresa spațiului temporar de lucru. <#0030, #11B7, #169E, #2089>
23651	#5C63	STKBOT	2	Adresa vârfului stivei calculatorului aritme- tic. <#11B7, #169E, #2089>
23653	#5C65	STKEND	2	Adresa sfârșitului stivei calculatorului arit- metic. <#11B7, #19FB, #1EAC, #1F05, #335B>
23655	#5C67	B_REG	1	Stocarea temporară a registrului B al calcu- latorului aritmetic în timpul execuției unui ciclu. <#335B, #367A>
23656	#5C68	MEM	2	Adresa memoriilor calculatorului aritmetic (MEMBOT = 23698). <#1D03>
23658	#5C6A	FLAGS2	1	Grup de 8 variabile de 1 bit: bit 0=1: urmează ștergerea ecranului <#0DAF> bit 1=0: buffer-ul imprimantei este gol <#0EDF> bit 2=0: caracterul curent nu este între ghilimele <#18E1>

Tabelul variabilelor de sistem (continuare)

Adresa în zecimal	Adresa în hexa	Numele	Lungimea	Semnificația (în paranteze valoarea implicită) <în paranteze unghiulare adresa subrutinei din ROM unde se folosește variabila respectivă>
				bit 3=1: modul C al cursorului <#18E1> bit 4=0: nu se folosește canalul K <#107F>
23659	#5C6B	DF_SZ	1	Numărul de linii din partea de jos a ecranului (inclusiv o linie albă). (2) <#0C55, #0D6B, #11B7, #1795, #2089>
23660	#5C6C	S_TOP	2	Numărul liniei din partea de sus a ecranului la listing automat. <#1795>
23662	#5C6E	OLDPPC	2	Linia la care sare CONTINUE. <#15EF>
23664	#5C70	OSPPC	1	Numărul instrucțiunii din linie, la care sare CONTINUE. <#15EF>
23665	#5C71	FLAGX	1	Grup de 8 variabile de 1 bit: bit 0=1: se va șterge vechea valoare a variabilei șir curente <#2AEE>. bit 1=1: o nouă variabilă <#2089, #2AEE> bit 5=1: modul INPUT <#0F2C, #0FA9, #2089> bit 5=0: modul editare <#1313> bit 7=1: se folosește INPUT LINE <#2089>
23666	#5C72	STRLEN	2	Lungimea șirului de caractere destinat interpretării. <#1D03, #2AEE>
23668	#5C74	T_ADDR	2	Adresa următorului element din tabela de sintaxă. Alternativ, T_ADDR _L indică tipul header <#0605, #1B55>
23670	#5C76	SEED	2	Baza seriei numerelor pseudoaleatoare generate de funcția RND. <#1E4F>
23672	#5C78	FRAMES	3	Numărătorul semicadrelor TV generate de Spectrum, folosit drept ceas de timp real. Primul octet este cel mai semnificativ. 1 unitate = 20 ms. Se reinițializează după 93.2 ore de funcționare neîntreruptă a calculatorului. <#0038, #1E4F>

Tabelul variabilelor de sistem (continuare)

Adresa în zecimal	Adresa în hexa	Numele	Lungimea	Semnificația (în paranteze valoarea implicită) < în paranteze unghiulare adresa subrutinei din ROM unde se folosește variabila respec- tivă >
23675	#5C7B	UDG	2	Adresa primului caracter din zona de UDG-uri. <#11B7>
23677	#5C7D	COORDS	2	Coordonatele x și y ale ultimului pixel adresat. <#0DAF, #22DC, #2320, #2382>
23679	#5C7F	P_POSN	1	33 - numărul coloanei la imprimantă. <#0ADC>
23680	#5C80	PR_CC	2	Pointer către următorul caracter din buffer-ul de imprimantă. <#0ADC, #0EDF>
23681	#5C81		1	(nefolosit)
23682	#5C82	ECHO_E	2	octet 1: 33 - coloana în partea de jos a ecranului; octet 2: 24 - linia în partea de jos a ecranului ale ultimului caracter în buffer-ul de intrare. <#0ADC, #2089>
23684	#5C84	DF_CC	2	Adresa în ecran a caracterului care urmează să fie scris cu PRINT pe calea S. <#0ADC>
23686	#5C86	DF_CCL	2	Adresa în ecran a caracterului care urmează să fie scris cu PRINT pe calea K. <#0ADC>
23688	#5C88	S_POSN	2	octet 1: 33 - coloana; octet 2: 24 - linia; ale poziției PRINT <#2089>
23690	#5C8A	SPOSNL	2	octet 1: 33 - coloana; octet 2: 24 - linia; ale poziției PRINT în partea inferioară a ecranului <#2089>
23692	#5C8C	SCR_CT	1	Numărul de linii + 1 care se afișează înainte de a întreba 'scroll?'. <#0C55, #0DAF, #2089>
23693	#5C8D	ATTR_P	1	Atributele permanente. <#0DAD>
23694	#5C8E	MASK_P	1	Mașca pentru atributele permanente transparente (biții de 1 corespund atributelor care

Tabelul variabilelor de sistem (continuare)

Adresa în zecimal	Adresa în hexa	Numele	Lungimea	Semnificația (în paranteze valoarea implicită) <în paranteze unghiulare adresa subrutinei din ROM unde se folosește variabila respec- tivă>
				se propagă prin transparență).
23695	#5C8F	ATTR_T	1	Atributele temporare. <#0BDB>
23696	#5C90	MASK_T	1	Masca pentru atributele temporare trans- parente (bitii de 1 corespund atributelor care se propagă prin transparență).
23697	#5C91	P_FLAG	1	grup de 8 variabile de 1 bit: bit 0 = 1: OVER 1 <#22DC> bit 2 = 1: INVERSE 1 <#22DC> bit 4 = 1: INK 9 <#0BDB> bit 6 = 1: PAPER 9 <#0BDB>
23698	#5C92	MEMBOT	30	Zonă de memorie rezervată calculatorului aritmetic.
23728	#5CB0		2	(nefolosit)
23730	#5CB2	RAMTOP	2	Adresa celui mai de sus octet accesibil sistemului Basic. (65368) <#11B7, #1EAC>
23732	#5CB4	P_RAMT	2	Adresa celui mai de sus octet din RAM (65535) <#11B7, #1EAC>

Anexa G - MODURILE DE LUCRU ȘI FUNCȚIILE TASTATURII

Organizarea tastaturii este cea mai particulară trăsătură a *Spectrum*-ului. Pentru a evita introducerea greșită a cuvintelor-cheie, precum și pentru a simplifica interpretorul Basic, echipa condusă de Steven Vickers a conceput o tastatură originală cu 40 de taste multifuncționale. Funcția fiecărei taste depinde de modul de lucru și de context. Utilizarea tastaturii *Spectrum* este destul de comodă pentru cineva cu experiență, dar este mai anevoioasă pentru un începător. Planșele următoare ilustrează efectele tastelor în toate împrejurările posibile. Recomandăm începătorilor să facă exerciții cu planșele

K

Modul KEYWORDS (cuvinte cheie)

L

Modul LETTERS (litere)

C

Modul CAPITALS (majuscule)

E

Modul EXTENDED (extins)

G

Modul GRAPHICS (caractere semigrafice)

X

Revenirea la modul anterior

CS

CAPS SHIFT

delete

caracter de control

SS

SYMBOL SHIFT

POINT

cuvânt-cheie

CS

tasta trebuie ținută
apăsată concomitent
cu o altă tastă

UDG
A

caracter grafic definit
de utilizator (UDG)



apăsarea tastei
nu produce nici un efect util



caracter semigrafic

K

1	2	3	4	5	6	7	8	9	0
PLOT	DRAW	REM	RUN	RANDOMIZE	RETURN	IF	INPUT	POKE	PRINT
NEW	SAVE	DIM	FOR	GO TO	GO SUB	LOAD	LIST	LET	Enter
CS	COPY	CLEAR	CONT	CLS	BORDER	NEXT	PAUSE	CS	Space

L

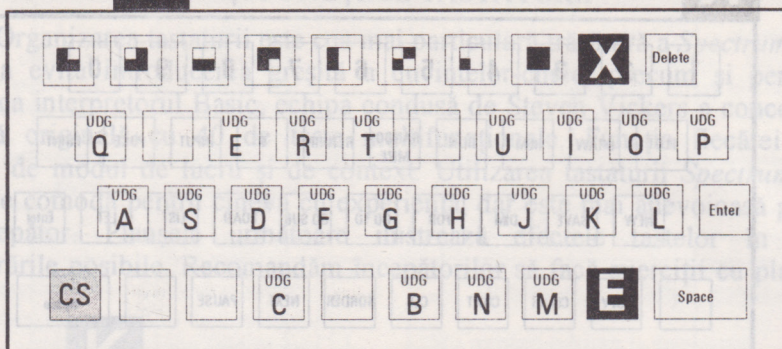
1	2	3	4	5	6	7	8	9	0
q	w	e	r	t	y	u	i	o	p
a	s	d	f	g	h	j	k	L	Enter
CS	z	x	c	v	b	n	m	CS	Space

G

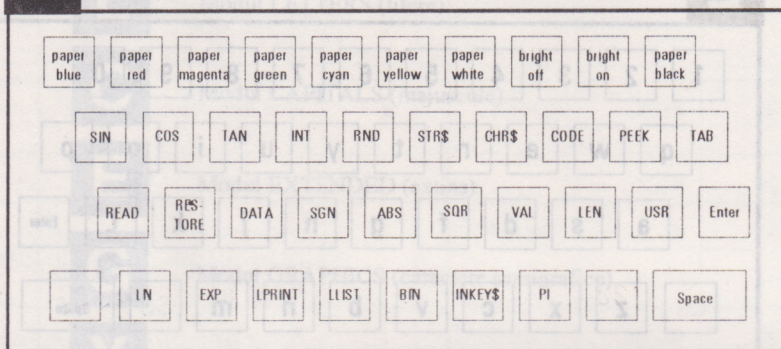
--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

+

G



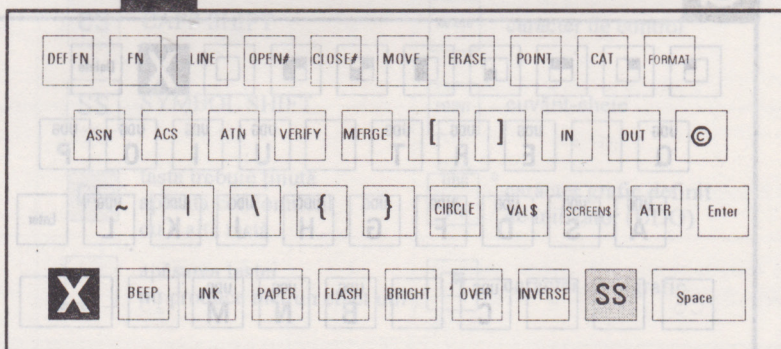
E



SS

+

E



CS + **L**

Edit	C	True video	Inv video	←	↓	↑	→	G	Delete
Q	W	E	R	T	Y	U	I	O	P
A	S	D	F	G	H	J	K	L	Enter
CS	Z	X	C	V	B	N	M	E	Break

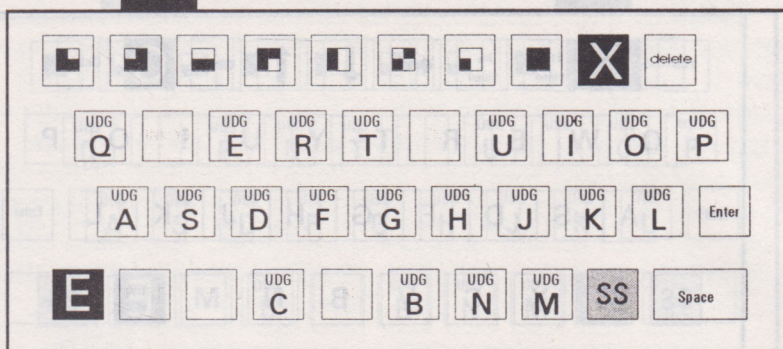
C

1	2	3	4	5	6	7	8	9	0
Q	W	E	R	T	Y	U	I	O	P
A	S	D	F	G	H	J	K	L	Enter
	Z	X	C	V	B	N	M		Space

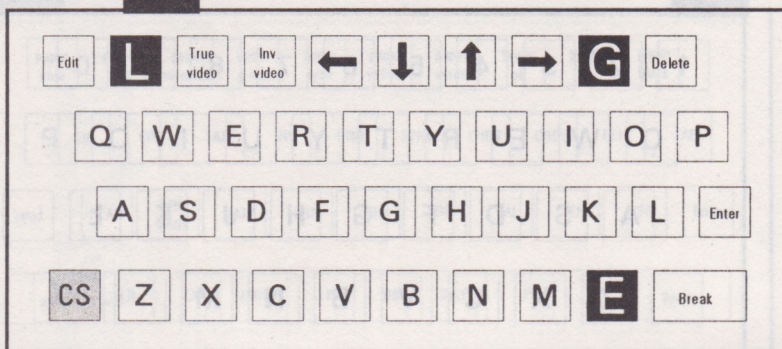
CS + **E**

ink blue	ink red	ink magenta	ink green	ink cyan	ink yellow	ink white	flash off	flash on	ink black
ASN	ACS	ATN	VERIFY	MERGE	[	]	IN	OUT	©
-		\	{	}	CIRCLE	VAL\$	SCREENS	ATTR	Enter
CS	BEEP	INK	PAPER	FLASH	BRIGHT	OVER	INVERSE	X	Space

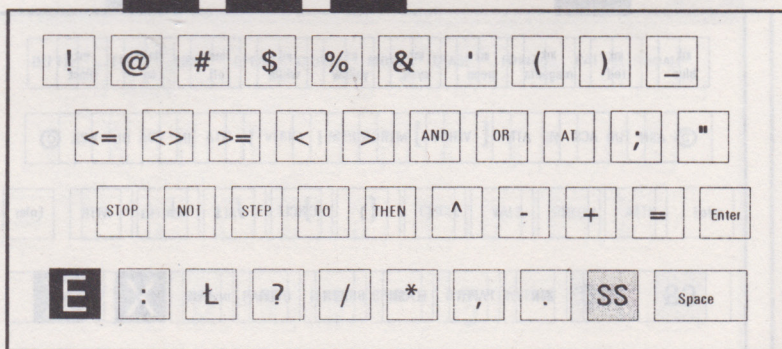
SS + **G**



CS + **C**



SS + **K L C**



Anexa H - FUNCȚIILE INTERPRETORULUI BASIC

Reguli generale privind funcțiile

1. Sintaxa funcțiilor Basic la *Spectrum* care au un singur argument nu obligă la introducerea argumentelor între paranteze. De exemplu, $\text{SIN } x$ este la fel de corect ca $\text{SIN } (x)$. Dacă însă argumentul este o expresie compusă din mai mulți factori sau termeni, parantezele trebuie puse, pentru că altfel funcția se aplică numai primului factor sau termen al expresiei. De exemplu, $\text{SIN } (2*x)$ nu este echivalent cu $\text{SIN } 2*x$, ultima formă fiind evaluată ca $(\text{SIN } 2)*x$.

De asemenea, funcțiile fără nici un argument nu trebuie scrise cu paranteze, ca de exemplu $\text{PI } ()$, ci PI .

Sintaxa funcțiilor definite de utilizator impune însă parantezele la orice număr de argumente, de exemplu: $\text{FN } a()$, $\text{FN } b(x)$, $\text{FN } c(x,y,z)$.

2. În descrierea sintaxei, oriunde apare un număr, poate apărea și o expresie cu rezultat numeric, iar unde apare un șir de caractere, poate apărea și o expresie cu un asemenea rezultat.

3. Toate funcțiile care întorc un caracter sau un șir de caractere (string) au numele terminat în simbolul \$ (dolar), pentru a le deosebi de funcțiile care întorc o valoare numerică.

4. Utilizatorul poate defini până la 26 de funcții cu rezultat numeric și până la 26 de funcții cu rezultat de tip șir de caractere. Definirea se face cu instrucțiunea DEF FN , iar numele funcției constă dintr-o singură literă, urmată eventual de semnul \$. DEF FN nu poate fi dat în linie de comandă, ca majoritatea celorlalte instrucțiuni. Este obligatorie stocarea ei în program, sub un număr de linie. Poziția unei linii DEF FN în program nu este obiect al unor restricții, dar se recomandă plasarea tuturor liniilor care conțin DEF FN cât mai la începutul programului, pentru accelerarea execuției.

Funcțiile Basic

ABS (Absolute value = valoare absolută)

Semnificație: funcția matematică *modul* sau *valoare absolută*.

Argument: un număr.

Rezultat: numărul identic ca valoare, dar cu semnul +.

Observații: asupra numerelor pozitive și asupra numărului zero, funcția nu are practic nici un efect; numerele negative sunt transformate în opusele lor.

Exemplu: PRINT ABS -5, ABS 5, 5 5

ACS (Arc cosine = arccosinus)

Semnificație: funcția invers trigonometrică *arccosinus*.

Argument: un număr real cuprins în intervalul $[-1, 1]$.

Rezultat: unghiul în radiani din intervalul $[0, \pi]$, al cărui cosinus este numărul dat ca argument.

Observații: $-1 \leq x \leq 1$, altfel apare o eroare cu mesajul: A Invalid argument; este inversa funcției COS.

Exemplu: PRINT 180*ACS .5/PI 60

ASN (Arcsine = arcsinus)

Semnificație: funcția invers trigonometrică *arcsinus*.

Argument: un număr real cuprins în intervalul $[-1, 1]$.

Rezultat: unghiul în radiani din intervalul $[-\pi/2, \pi/2]$, al cărui sinus este numărul dat ca argument.

Observații: $-1 \leq x \leq 1$, altfel apare o eroare cu mesajul: A Invalid argument; este inversa funcției SIN.

Exemplu: PRINT 180*ASN .5/PI 30

ATN (Arctangent = arctangenta)

Semnificație: funcția invers trigonometrică *arctangentă*.

Argument: un număr real.

Rezultat: unghiul în radiani din intervalul $(-\pi/2, \pi/2)$, a cărui tangentă este numărul dat ca argument.

Observații: este inversa funcției TAN.

Exemplu: PRINT 180*ATN 1/PI 45

ATTR (Attributes = atribute)

Semnificație: atributele unui caracter de pe ecran, de la poziția dată ca argument.

Argument: două numere separate prin virgulă: linia în intervalul $0 \div 23$ (în mod normal $0 \div 21$) și coloana în intervalul $0 \div 31$.

Rezultat: Atributele caracterului codificate ca întreg de 1 octet ($0 \div 255$).

Exemplu: CLS

PRINT AT 10,31: PAPER 1: INK 7: FLASH 1: BRIGHT 0: "***"
PRINT ATTR(10,31)

87

BIN (Binary number = număr binar)

Semnificație: conversia unei constante numerice din baza 2 în baza 10.

Argument: un număr în baza 2 (format numai din 0 și 1).

Rezultat: același număr convertit în baza 10.

Observații: este singura funcție care nu acceptă o expresie ca argument, ci doar o constantă numerică. De exemplu, este incorect LET a=BIN b.

Exemplu: PRINT BIN 10110000110110 11318

CHR\$ (Character = caracter)

Semnificație: întoarce caracterul al cărui cod ASCII este dat ca argument.

Argument: un număr întreg între 0 și 255.

Rezultat: caracterul cu codul ASCII dat ca argument.

Observații: funcția este foarte utilă pentru scrierea caracterelor de control în programe; este inversa funcției CODE.

Exemplu: PRINT "Sp":CHR\$ 8:"ect":CHR\$ 8:CHR\$ 8:"rum"

Serum

CODE (= cod)

Semnificație: întoarce codul ASCII al primului caracter din șirul de caractere dat ca argument.

Argument: un șir de caractere.

Rezultat: numărul zecimal între 0 și 255, reprezentând codul ASCII al primului caracter din șir; dacă șirul este vid, atunci rezultatul este 0.

Observații: CODE "" = 0; este inversa funcției CHR\$.

Exemplu: PRINT CODE "Spectrum",CODE "" 83 0

COS (Cosine = cosinus)

Semnificație: funcția *cosinus* a unui unghi exprimat în radiani.

Argument: un număr real reprezentând un unghi în radiani.

Rezultat: valoarea funcției cosinus ca număr real între -1 și 1.

Observații: inversa sa este ACS.

Exemplu: PRINT COS(45*PI/180) .70710678

EXP (Exponential = exponențiala)

Semnificație: funcția exponențială, e^x .

Argument: un număr real.

Rezultat: numărul $e = 2,7182818$ ridicat la puterea dată ca argument.

Observații: EXP(0)=1, EXP(1)=2.7182818, EXP(-1)=0.36787944; este inversa funcției LN.

Exemplu: PRINT EXP 2 7.3890561

FN ? (Function = funcția)

Semnificație: o funcție definită de utilizator; în locul semnului de întrebare apare numele funcției de o literă, eventual urmată de semnul \$.

Argumente: între 0 și 26 de argumente numerice sau șir de caractere; numărul de argumente și tipul lor trebuie să concorde între definiția funcției prin DEF FN și apelurile ei.

Rezultat: șir de caractere sau numeric, după cum numele funcției este sau nu urmat de semnul \$. Rezultatul provine din evaluarea expresiei definite în instrucțiunea DEF FN corespunzătoare.

Exemple: Funcția r(d) convertește grade hexagesimale în radiani, iar funcția d(r) convertește radiani în grade hexagesimale.

1 DEF FN r(d)=d*PI/180: DEF FN d(r)=r/180*PI

PRINT FN r(180).FN d(2*PI) 3.1415927 360

IN (Input = intrare)

Semnificație: Permite citirea unui octet dintr-un port.

Argument: Adresa port-ului (pe 8 sau 16 biți).

Rezultat: Valoarea citită din *port* ca întreg reprezentat pe un octet.

Observații: este utilă pentru citirea tastaturii direct de la *port* și nu prin intermediul sistemului de operare, ca `INKEY$`; funcția inversă o realizează instrucțiunea `OUT`.

Exemplu: exemplul următor sesizează apăsarea simultană a tastelor A, S și D:

```
PRINT IN 254
```

INKEY\$ (Input from Keyboard = intrare de la tastatură)

Semnificație: Întoarce tasta apăsată în momentul apelului funcției.

Argument: nu are.

Rezultat: un caracter care corespunde tastei apăsate în momentul apelului funcției.

Observații: funcția nu așteaptă utilizatorul să apese o tastă și nici nu permite citirea combinațiilor de taste; dacă în momentul apelului nu este apăsată nici o tastă, atunci funcția întoarce șirul de caractere vid.

Exemplu: exemplul care urmează permite manevrarea unui obiect pe ecran din tastele cursorului:

```
10 CLS : LET x=16: LET y=11: GO TO 90
20 LET d$=INKEY$
30 IF NOT LEN d$ THEN GO TO 20
40 IF d$="5" AND x>0 THEN LET x=x-1
50 IF d$="8" AND x<31 THEN LET x=x+1
60 IF d$="7" AND y>0 THEN LET y=y-1
70 IF d$="6" AND y<21 THEN LET y=y+1
80 PRINT AT yv,xv;" ";AT y,x;"*";
90 LET xv=x: LET yv=y: GO TO 20
```

INT (Integer = întreg)

Semnificație: convertește un număr real la întregul imediat inferior.

Argument: un număr real.

Rezultat: numărul întreg imediat inferior argumentului.

Observații: în alte limbaje de programare există și funcții de întreg imediat superior, întregul cel mai apropiat (rotunjire) și întregul imediat inferior în valoare absolută. Dintre acestea, cea mai utilă este rotunjirea, care se poate defini ca:

```
1 DEF FN r(x)=SGN x * INT(ABS x + .5)
```

```
PRINT FN r(2.7),FN r(-2.3)      3      -2
```

Exemplu: `PRINT INT 2.7,INT -2,3 2 -3`

LEN (Length = lungime)

Semnificație: întoarce lungimea unui șir de caractere.

Argument: un șir de caractere.

Rezultat: un număr întreg pozitiv care exprimă numărul de caractere conținut de șirul dat ca argument.

Observații: `LEN "" = 0`

Exemplu: linia de mai jos scrie un titlu centrat pe orizontală:

```
LET t$="Spectrum": PRINT TAB((32-LEN t$)/2):t$
```


LN (Logarithm, Natural = logaritm natural)

Semnificație: logaritmul natural al unui număr.

Argument: un număr strict pozitiv.

Rezultat: un număr real la care trebuie ridicat la putere numărul $e=2.7182818$, pentru a obține numărul dat ca argument.

Observații: LN 1 = 0; $x > 0$, altfel apare o eroare cu mesajul: A Invalid argument; este inversa funcției EXP.

Exemplu: PRINT LN 2, LN 1 0.69314718 0

PEEK

Semnificație: întoarce un octet citit direct dintr-o locație de memorie.

Argument: adresa locației de memorie ca întreg fără semn reprezentat pe 16 biți, între 0 și 65535.

Rezultat: valoarea conținută de locația respectivă ca număr reprezentat pe un octet, între 0 și 255.

Observații: $0 \leq x \leq 65535$; funcția inversă o realizează instrucțiunea POKE.

Exemplu: linia următoare citește o locație de la adresa de început a șirului de caractere "scroll?" din ROM.

```
PRINT PEEK 3321 115
PRINT CHR$ 115 s
```

PI

Semnificație: întoarce constanta $\pi = 3.1415927$.

Argument: nu are.

Rezultat: numărul $\pi = 3.1415927$.

Exemplu: fragmentul următor de program calculează aria cercului de rază R:

```
LET R=5: LET A=PI*R*R: PRINT R,A 5 78.539816
```

POINT (= punct)

Semnificație: permite detectarea stării unui pixel din ecran.

Argumente: coordonatele x și y ale pixelului, ca numere întregi fără semn.

Rezultat: 0 dacă pixelul are culoarea hârtiei, 1 dacă are culoarea cernelii.

Observații: $0 \leq x \leq 255$, $0 \leq y \leq 175$; funcția inversă o realizează instrucțiunea PLOT.

Exemplu: instrucțiunile care urmează pun un pixel din centrul ecranului pe 1 și apoi pe 0, citindu-i starea de fiecare dată:

```
CLS
PLOT 128,88: PRINT POINT (128,88) 1
PLOT INVERSE 1:128,88: PRINT POINT (128,88) 0
```

RND (Random number = număr aleator)

Semnificație: furnizează un număr pseudoaleator între 0 și 1.

Argumente: nu are.

Rezultat: un număr între 0 și 1 generat la întâmplare.

Observații: funcția RND va întoarce exact același număr după 65536 apeluri, cu alte cuvinte seria pseudoaleatoare are perioada de 65536; instrucțiunea RANDOMIZE n permite poziționarea în cadrul seriei.

Exemplu: RANDOMIZE 65535: PRINT RND.RND 0.99885559 0.91416931

SCREEN\$ (= ecran)

Semnificație: întoarce caracterul care apare pe ecran în poziția dată.

Argumente: două numere întregi: linia, între 0 și 21, și coloana, între 0 și 31.

Rezultat: întoarce codul ASCII al caracterului scris pe ecran în poziția dată; recunoaște caracterele scrise în video invers.

Exemplu: 10 PRINT AT 5.3:"Sinclair"
 20 PRINT AT 5.5:SCREEN\$ (5.7)
 Silclair

SGN (Signum = signum)

Semnificație: funcția matematică *signum*.

Argument: un număr.

Rezultat: 0 dacă argumentul este 0, 1 dacă argumentul este pozitiv sau -1 dacă argumentul este negativ.

Exemplu: PRINT SGN -2.5:" ";SGN 0:" ";SGN 1000 -1 0 1

SIN (Sine = sinus)

Semnificație: funcția trigonometrică *sinus* a unui unghi exprimat în radiani.

Argument: un număr real reprezentând un unghi în radiani.

Rezultat: valoarea funcției sinus ca număr real între -1 și 1.

Observații: inversa sa este ASN.

Exemplu: PRINT SIN(30*PI/180) .5

SQR (Square Root = rădăcina pătrată)

Semnificație: funcția *radical* sau *rădăcină pătrată* a unui număr pozitiv.

Argument: un număr real pozitiv.

Rezultat: un număr reprezentând rădăcina pătrată a argumentului.

Observații: $x \geq 0$, altfel apare o eroare cu mesajul: A Invalid argument.

Exemplu: 10 PRINT SQR 2 1.4142136

STR\$ (String = șir de caractere)

Semnificație: convertește un număr în șirul de caractere corespunzător scrierii sale pe ecran, format din cifre, punct zecimal, semnul, litera E în cazul notației cu exponent.

Argument: un număr.

Rezultat: șirul de caractere corespunzător scrierii argumentului pe ecran cu instrucțiunea PRINT.

Exemplu: 10 LET a\$="pi="+STR\$ PI
 20 PRINT a\$
 pi=3.1415927

TAN (Tangent = tangenta)

Semnificație: funcția trigonometrică *tangentă* a unui unghi exprimat în radiani.

Argument: un număr real reprezentând un unghi în radiani.

Rezultat: valoarea funcției tangenta ca număr real.

Observații: inversa sa este ATN.

Exemplu: PRINT SIN(30*PI/180) .5

USR (User Subroutine = subrutină a utilizatorului)

Semnificație: întoarce conținutul registrului BC după execuția unei subrutine în cod mașină de la adresa dată ca argument.

Argument: un număr întreg între 0 și 65535, reprezentând adresa de început a subrutinei în cod mașină.

Rezultat: un număr întreg între 0 și 65535, reprezentând conținutul registrului BC la întoarcerea din subrutină.

Observații: cel mai adesea nu rezultatul este util, ci efectul lateral al funcției constând din execuția subrutinei în cod mașină; de aceea, adeseori această funcție este apelată ca RANDOMIZE USR n, unde RANDOMIZE este un simplu pretext de a invoca funcția.

Exemplu: în secvența următoare se apelează direct o subrutină din ROM care întoarce numărul de locații de memorie ocupate de sistem și de programele utilizatorului; scăzând acest număr din 65536, vom obține numărul de locații de memorie rămase disponibile la un moment dat.

PRINT 65536-USR 7962 41398

USR (User defined graphics = caractere grafice definite de utilizator)

Semnificație: întoarce adresa zonei de definiție a unui caracter definit de utilizator (UDG).

Argument: un șir de caractere care conține caracterul căruia îi corespunde UDG-ul respectiv, între "A" și "U" sau între "a" și "u".

Rezultat: un număr întreg între 0 și 65535, reprezentând adresa de început a zonei unde este definit UDG-ul dat ca argument.

Observații: practic există două funcții omonime USR distincte, decelarea lor făcându-se numai prin natura argumentului.

Exemplu: PRINT USR "b" 65376

VAL (Value = valoare)

Semnificație: evaluează o expresie dată ca șir de caractere și întoarce rezultatul numeric al expresiei.

Argument: un șir de caractere care conține o expresie numerică validă, după sintaxa Basic Spectrum.

Rezultat: un număr reprezentând rezultatul evaluării expresiei în momentul execuției.

Observații: este funcția cea mai puternică a limbajului Basic Spectrum, permițând utilizatorilor invocarea evaluatorului de expresii.

Exemplu:
10 LET a\$="2+3"
20 PRINT VAL (a\$+a\$+"2")
20

VAL\$ (Value = valoare)

Semnificație: evaluează o expresie dată ca șir de caractere și întoarce rezultatul de tip șir de caractere al expresiei.

Argument: un șir de caractere care conține o expresie de tip șir de caractere validă, conform sintaxei Basic Spectrum.

Rezultat: un șir de caractere reprezentând rezultatul evaluării expresiei în momentul execuției.

Observații: este o funcție aproape inutilă în practică, introdusă din considerente de simetrie față de funcția VAL.

Exemplu:

```
10 LET a$="b+c$"  
20 LET b$="Expresia:": LET c$=a$  
30 PRINT VAL$ a$  
Expresia:b+c$
```

SIN (Sine = sinus)

Semnificație: funcția trigonometrică sinus

Argument: un număr real care reprezintă unghiul în grade

Rezultat: valoarea sinusului unghiului dat

Observații: inversa sa este ASH

Exemplu:

SQR (Square root = rădăcină pătrată)

Semnificație: funcția radical

Argument: un număr real pozitiv

Rezultat: un număr reprezentând rădăcina pătrată a numărului dat

Observații: inversa sa este X^2

Exemplu:

SIN (Sine = sinus)

Semnificație: funcția trigonometrică sinus

Argument: un număr real care reprezintă unghiul în grade

Rezultat: valoarea sinusului unghiului dat

Observații: inversa sa este ASH

Exemplu:

SQR (Square root = rădăcină pătrată)

Semnificație: funcția radical

Argument: un număr real pozitiv

Rezultat: un număr reprezentând rădăcina pătrată a numărului dat

Observații: inversa sa este X^2

Exemplu:

SIN (Sine = sinus)

Semnificație: funcția trigonometrică sinus

Argument: un număr real care reprezintă unghiul în grade

Rezultat: valoarea sinusului unghiului dat

Observații: inversa sa este ASH

Exemplu:

micro procesoare

Editura



Militară

ISBN 973-32-0431-51 Lei 6 200