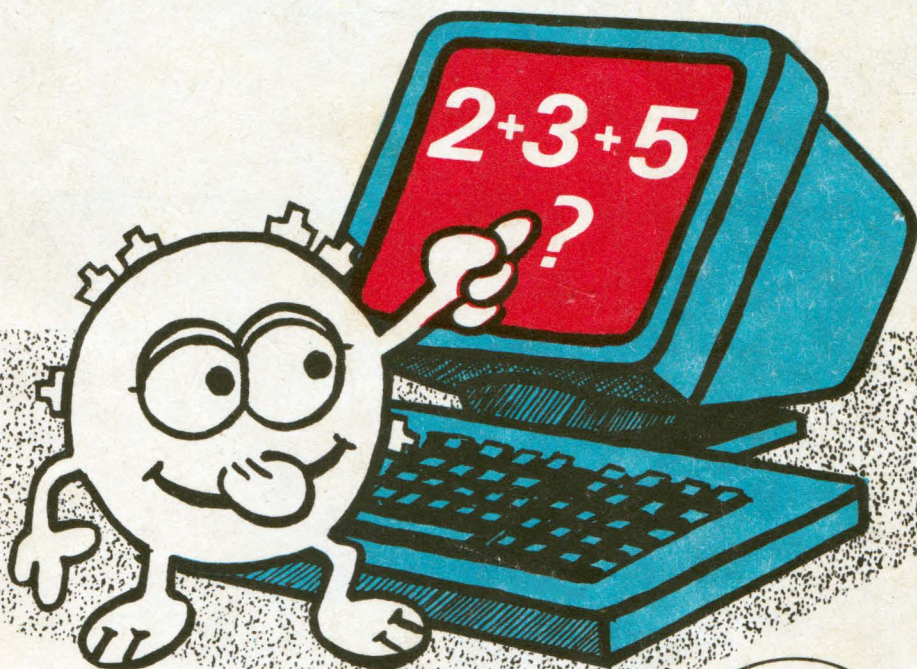


VLAD ATANASIU

MINUNATA
LUMEA
HC-ULUI



Editura Agni

Biblioteca
de
Informatică

Lucrarea

Vlad Atanasiu

Minunata lume a
HC-ului

Programe demonstrative în limbajul de asamblare Z80

Editura Agni

București 1994

ISBN 973-95626-7-1

© Toate drepturile sînt rezervate Editurii AGNI.

Editura AGNI,
CP:30-107, BUCUREȘTI
tel: 615.55.59 fax: 312.93.33

Redactor : *Victor Cososchi*

Tehnoredactare computerizată : *Vlad Atanasiu*

Coperta : *Adina Dumitriu*

Desene : *Mădălin Barnea*

CUPRINS

Cuvânt înainte

VII

1. Introducere

1.1 Scopul cărții

3

1.2 Cum trebuie citită

5

2. Noțiuni de bază

2.1 Diferența BASIC - cod mașină

7

2.2 Din ce este format un calculator

Memoria

10

Microprocesorul

11

3. Introducere în cod mașină

3.1 Regiștrii

15

3.2 Comparație instrucțiuni BASIC-cod mașină

Instrucțiuni pentru acumulator	20
Instrucțiuni pentru regiștri simpli	22
Instrucțiuni pentru registrul HL	25
Instrucțiuni pentru locații de memorie	29
Instrucțiuni pentru cuvinte de memorie	32
Instrucțiuni de salt	34
Alte instrucțiuni	35

3.3 Stiva

Cum se folosește	38
Cum este construită	42

3.4 Cum se folosește asamblorul	43
---------------------------------	----

3.5 Primul program în cod mașină	48
----------------------------------	----

3.6 Rutine din memoria ROM	52
----------------------------	----

4. *Efecte de acoperire a ecranului*

4.1 Presentare	57
----------------	----

4.2 Acoperirea de sus în jos	61
------------------------------	----

4.3 Acoperirea de jos în sus	62
------------------------------	----

4.4 Acoperirea de la stînga la dreapta	64
--	----

4.5	Stingere cu FLASH	65
-----	-------------------	----

5. Efecte de așteptare

5.1	Prezentare	69
5.2	FLASH colorat	69
5.3	Efecte pe BORDER	71

6. Lucrul cu imagini

6.1	Prezentare	75
6.2	Compactarea ecranului	77
6.3	Refacerea unui ecran compactat	80
6.4	Memorarea unei ferestre	81
6.5	Refacerea unei ferestre memorate	85
6.6	Negarea (inversarea) unei ferestre	86
6.7	Ștergerea unei ferestre	89

7. Mișcare pe ecran

7.1	Prezentare	
	Cum se realizează mișcarea ?	91
	Structura ecranului	93
	Setul de caractere	96

7.2	Deplasarea orizontală	99
7.3	Rotirea unei benzi orizontale pe ecran	106
7.4	Defilarea orizontală a unui text pe ecran	114
7.5	Deplasarea unei benzi verticale pe ecran	122
7.6	Defilarea unui text pe verticală	128
7.7	Mișcarea unui cursor pe ecran	138

Anexa A

	Instrucțiunile microprocesorului Z80	147
--	--------------------------------------	-----

Anexa B

	Lista programelor	157
--	-------------------	-----

	<i>Bibliografie</i>	161
--	----------------------------	-----

Cuvânt înainte

Nu puțini dintre noi am fost, mai devreme sau mai târziu, fermecați mai întâi de magia unei prezentări ale cărei litere se răsucesc ca la cinematograful și, mai apoi, de naturalețea unei mâini care, puțin tremurată și, poate de aceea, incredibil de uman lasă eleganta semnătură a autorului pe ecranul calculatorului.

Mulți au recunoscut, desigur, jocul Video Pool. Putem spune deja că am intrat pe tărâmul lumii vrăjite a HC-ului, iar fie numai și acest prim contact stârnește o fascinație - izvor al primelor întrebări: cum se realizează astfel de lucruri ? Este într-adevăr ceva deosebit de dificil, numai la îndemâna specialistului care a pierdut poate ani pentru a descoperi algoritmi și rutina care dă viață lucrurilor pe ecran sau este ceva care poate fi și la îndemâna noastră ?

Sunt întrebări pe care, probabil și le-a pus marea majoritate a copiilor atunci când au fost prima dată în fața unui calculator personal pentru că, evident, cel mai natural prim contact al copilului cu calculatorul este jocul. Apoi copilul a înțeles repede că instrumentul poate fi programat și, fie sub îndrumarea părinților, fie la un cerc de informatică, fie la școală a început cu BASIC-ul cel comod. Dar să nu uităm că a dorit să învețe programarea în primul rând pentru a-și satisface prima dorință, și anume, aceea de a realiza și el ceva deosebit de frumos și spectaculos care să semene, de exemplu, cu jocul amintit. După un timp însă a simțit că BASIC-ul a devenit o haină prea strâmtă pentru el, un instrument cu care nu a putut să-și realizeze visul.

Când s-a întâmplat acest lucru, la ce vârstă ? Este greu de răspuns iar răspunsul nu este același pentru fiecare copil. Oricum, această vârstă scade spectaculos odată cu progresele tehnologice. O încercare de răspuns o putem regăsi în celebrul articol de umor informatic. Adevărații programatori: "Adevărații programatori nu scriu în BASIC după venerabila vârstă de 12 ani... Ei scriu în cod mașină și nu pierd timpul cu declarații și comentarii."

Lăsând gluma la o parte, indiferent de vârstă, este un moment de cumpănă care marchează trecerea de la joacă sau de la învățarea prin joc la învățarea pentru joc, la învățarea meșteșugului necesar creației. Din acest punct de vedere, lucrarea de față nu este pentru oricine. Pentru a intra în lumea minunată a HC-ului sau, cu alte cuvinte, pentru a cunoaște și a "simți mașina", sunt necesare multe calități: spirit de cercetare, meticulozitate și conștiinciozitate, dorința continuă de perfecționare și, nu în ultimul rând, surse de documentație. Cu alte cuvinte, lucrarea de față.

Nu este prima carte care abordează acest subiect și totuși se deosebește fundamental de cele dinainte.

Dacă primele erau realizate de specialiști pentru specialiști (și aici putem aminti pe inițiatorii domeniului de la noi din țară ca: V. Țepelea, C. Lupu, A. Petrescu, T. Moisa, Gh. Toacșe C. Stâncescu), trecând prin momentul apariției lucrării "Totul despre microprocesorul Z-80" realizată de un colectiv condus de N. Patrubby, lucrare adresată în principal studenților și elevilor de liceu de clase superioare, moment deosebit de important deoarece a marcat o abordare didactică a problemei, inclusiv cu asistență din partea calculatorului (am numit simulatorul grafic al funcționării microprocesorului, Visible Z-80, o creație software deosebită, dar care, din păcate, nu a fost proiectată și pentru HC), ajungem, în sfârșit, la lucrarea de față, care are meritul de a se adresa direct copiilor, elevilor din cursul gimnazial și liceal, de a pune la îndemâna lor instrumentul necesar pentru descoperirea minunatei lumi a HC-ului.

* * *

Programarea în limbaj mașină este urmarea firească a programării în BASIC. Cunoașterea aprofundată a acestuia constituie o recomandare prealabilă a învățării codului mașină. Metoda adoptată în lucrare pentru învățarea acestuia constă în transpunerea progresivă și cât mai completă a limbajului BASIC în limbaj mașină; lectorul deja familiarizat cu primul, accede astfel mult mai ușor la cunoașterea celuiilalt decât pornind de la o bază complet diferită. O dată învățată această tehnică, programarea în cod mașină pe orice calculator construit cu microprocesor Z-80 nu va prezenta dificultăți.

Avantajele programării în limbaj mașină sunt multiple. Printre ele putem menționa viteza superioară de execuție față de BASIC, ceea ce este foarte important în special la programele care folosesc animația. Consumul de memorie este comparativ mai mic la performanțe egale, programele în limbaj mașină necesitând un spațiu mai mic. Nu mai puțin importantă este și posibilitatea de realizare de comenzi mai orientate către nevoile specifice ale fiecărui utilizator. Astfel, folosind aceste avantaje, se pot realiza diverse tehnici speciale și foarte spectaculoase ca: efecte de acoperire a ecranului, efecte pe border, compactarea ecranului, memorarea unei ferestre, deplasarea pe orizontală și pe verticală, defilarea unui text pe orizontală sau pe verticală, mișcarea unui cursor pe ecran, în paralel fiind prezentate în lucrare și programele BASIC similare. Prezentarea în paralel are cel puțin trei scopuri. Lectorul obișnuit cu BASIC va înțelege ușor programele în acest limbaj și deci principiile și modalitățile folosite pentru rezolvarea problemei respective. În al doilea rând, prin comparare, lectorul va nota faptul că programele în cod sunt mai directe, deci mai scurte. Și, în ultimul rând, dar cel mai important, va remarca eficiența programelor în cod față de cele în BASIC; atunci când va executa programele introduse, cele în cod vor fi mult mai rapide (în efectele mult mai apropiate de cele reale) decât cele în BASIC. De asemenea, foarte eficientă este și folosirea unor rutine din memoria ROM.

Înainte de a începe studiul codului mașină sunt necesare câteva cunoștințe prealabile ca: sistemele de numerație în care se efectuează operațiile în limbaj mașină (binar, hexazecimal), modul în care aceste operații sunt realizate de calculator, precum și organizarea generală a unui calculator HC și a microprocesorului Z-80. Poate cel mai plastic dintre aspectele legate de organizarea generală a unui microcalculator și a microprocesorului Z-80 au fost redată pe înțelesul copiilor cel mai plastic sub formă de poveste în Manualul de prezentare a calculatorului PRAE (N. Patrubby și colectiv) din care spicuim următoarele paragrafe ce pot constitui de altfel o veritabilă introducere la lucrarea de față:

"Dacă, neputându-ne stăpâni curiozitatea, am desfăcut deja calculatorul și am admirat jungla de capsule de circuite integrate și trasee de circuite imprimate care șerpuiesc ca niște liane, să închidem frumos

cutia și, cu puțină fantezie, să ne imaginăm cum funcționează aparatul. Să pătrundem deci, fără șurubelniță, în interiorul calculatorului!

Vom vedea o sală imensă care strălucește în lumina neanelor. Lângă un perete observăm 8 dulapuri având fiecare 2048 de sertare, care, la o examinare mai atentă, se dovedesc a fi, de fapt, niște seifuri, prevăzute toate și cu un mic vizor. Fiind și noi curioși, ne uităm repede prin câteva vizoare și s-ar putea spune că suntem dezamăgiți. În fiecare seif vom vedea același lucru, și anume, un mic dispozitiv pentru afișarea unui număr binar format din 8 cifre și un beculeț care are rostul tocmai de a lumina numărul pentru a-l putea vedea. Ne aruncăm privirea pe al doilea perete și vedem iarăși niște dulapuri, dar mai multe, și tot cu câte 2048 de sertare. Uitându-ne la câteva dintre ele constatăm cu stupeoare că în fiecare regăsim același dispozitiv de afișare ca în seifurile admirate înainte, însă, de data aceasta, putem modifica cu ușurință numerele din sertare, acestea nemaîndeplinind rolul de seifuri... Dar ce ființe misterioase lucrează oare în această sală și ce activitate desfășoară?

Figura cea mai pitorească și totodată tipul cel mai deștept este microprocesorul Z-80. El este cel care, de la impunătorul său birou, dirijează și coordonează toată activitatea din această sală uriașă.

Ajutorul său neprețuit este un altul, iute ca fulgerul, care este într-un continuu dute-vino între sertare și seifuri și biroul lui Z-80, precum și între ușile din sală și microprocesor. Să-l botezăm pe atlet cu numele Impi (impulsul este ceva iute, nu-i așa?). Sunt trei uși de serviciu pe care putem citi: Tastatura, Interfața și Casetofon și la ele fac de serviciu trei lucrători ale căror nume seamănă cu inscripțiile de deasupra ușilor: Tasti, Seri, și Casi. Ultimii doi au o mutră somnoroasă și, se pare că, de obicei, au mai puțin de lucru decât colegul lor, Tasti. Încă o figură interesantă ne atrage atenția. Este Afi, cel care face curse cu o viteză

infernala între sertarele numerotate cu numere (adresele, ați înțeles!) și un ecran (de cinematograf) care este legat la această sală.

Dar ce muzică stranie auzim de când am intrat în această sală? Parcă ne-am fi întors în istorie și am asculta ritmul monoton al tobelor romane de pe o galeră plină cu sclavi care vâslesc cu ochii ațintiți în gol. Dar nuli Doar efectul sonor este asemănător. Un flăcău simpatic, numit Cuarț, bate cu însuflețire o tobă cu 250 000 de bătăi pe secundă. Microprocesorul are grijă ca toată activitatea din sală să se desfășoare conform ritmului impus de Cuarț. Toată lumea din sală este mulțumită de această disciplină în muncă. Marea majoritate a sefurilor conțin, de fapt, numere care reprezintă codurile unor instrucțiuni. Unele dintre ele sunt scurte și încap într-un singur seif, iar cele mai lungi se pot înșira pe două, trei sau chiar patru. Singurul personaj din această sală care înțelege instrucțiunile este microprocesorul Z-80.

Suntem deja foarte curioși cum se desfășoară activitatea în această încăpere bizară. Iată ce putem observa: când lumina neanelor inundă sala, toți lucrătorii se așază la locurile lor. Atletul Impi citește și notează imediat numărul din seiful O și, cu sufletul la gură, se duce și îl comunică lui Z-80. Acesta își dă seama imediat dacă numărul adus de Impi este o instrucțiune completă sau nu. Dacă e nevoie, Impi citește imediat partea a doua, a treia sau eventual a patra a instrucțiunii din sefurile imediat următoare. Având instrucțiunea completă pe biroul său, Z-80 ia măsuri ca aceasta să fie executată. Există o gamă variată de peste 700 de instrucțiuni, totalitatea lor constituind setul de instrucțiuni ale procesorului...

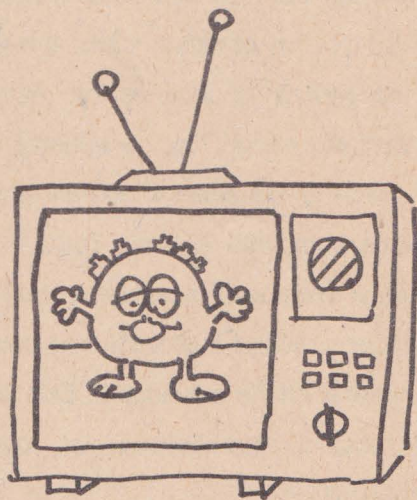
În mod normal, după ce atletul Impi a adus o instrucțiune pe biroul de lucru al lui Z-80, acesta îl trimite la seiful sau sertarul următor, unde va găsi o altă instrucțiune. Excepție face cazul în care se execută o

instrucțiune de salt în care se indică explicit seiful sau sertarul în care se află următoarea instrucțiune. Cum poate urmări stăpânul acestei instituții ciudate cele ce se întâmplă în sală? Nimic mai simplu! Racordează la sală un cablu care are celălalt capăt legat la intrarea de antenă a unui televizor și astfel va putea urmări pe ecranul acestuia ce se petrece în sală sau, mai bine zis, rezultatul muncii din sală."

Un foarte tânăr autor care a descoperit acest univers ne face o invitație în lumea minunată a HC-ului. E normal. Această lume a fost a tinerilor. Ea însă poate deveni și a copiilor iar lucrarea de față reprezintă una din cheile cu care se poate deschide poarta acestei lumi.

Deci fără teamă și cu încredere vă invităm să pătrundeți în lumea minunată a HC-ului!

Ion Diamandi



Dedic această carte domnului profesor Popovici, colegilor de liceu și tuturor celorlalți care m-au învățat să folosesc mașini și oameni.

27.09.1994
Blanc

Introducere

1.1 Scopul cărții

Scopul acestei cărți este nu numai de a prezenta o bibliotecă de programe utile, ci și acela de a oferi, prin aceste programe, exemple de utilizare a codului mașină pe microprocesorul Z80, exemple de care programatorii au nevoie pentru înțelegerea mai completă a limbajului de asamblare.

Cu microprocesorul Z80 sunt echipate următoarele calculatoare larg răspândite la noi în țară:

- ★ HC 85 și variantele sale HC 90 și HC 91
- ★ JET
- ★ CIP
- ★ TIM - S

Mai există și alte calculatoare care lucrează cu acest integrat, cum ar fi AMIC, PRAE sau chiar CUB-Z, dar pe acestea programele care vor fi prezentate aici nu vor putea fi rulate, deoarece ele sunt concepute să funcționeze doar pe calculatoare compatibile SPECTRUM (știu să folosească numai o structură similară de memorie, ecran sau unele chiar apelează subrutine din cele implementate în memoria ROM a acestor calculatoare).

Cartea este destinată în special celor care, fiind deja familiarizați cu programarea în limbajul BASIC pe aceste tipuri de calculatoare, vor să facă un pas în plus, să cunoască și limbajul secret al mașinii, programele lor câștigând astfel avantaje nebănuite.

Cu ajutorul acestui limbaj, devin posibile:

- ★ conceperea unui program profesional
- ★ realizarea de jocuri de viteză de genul celor care circulă, cu zecile, printre amatorii de HC-uri
- ★ programarea unor efecte performante (care pot fi eventual adăugate unor programe BASIC) prin gestiunea mult mai eficientă a ecranului și difuzorului.
- ★ "spargerea" jocurilor, înțelegerea și modificarea lor ("nemurire", timp infinit etc.)

Accesul în *mașină* fiind astfel deschis, se vor putea realiza mult mai multe acțiuni ale ei: se vor putea programa sunete și imagini imposibil de realizat din BASIC, se vor putea mișca rapid bucăți întregi din ecran, se vor putea combina culori și imagini cu o viteză foarte mare și chiar se va putea controla interfața cu caseta, încărcarea sau înregistrarea programelor!

1.2 Cum trebuie citită

Programele vor beneficia de o expunere detaliată, fiecare cuprinzând:

- ★ listingul programului în limbaj de asamblare
- ★ explicații cu privire la rolul instrucțiunilor folosite
- ★ explicații cu privire la algoritm
- ★ listingul programului în cod mașină (adică succesiunea de octeți care reprezintă traducerea programului asamblat) care permite introducerea în memorie a acestuia cu ajutorul unor instrucțiuni BASIC "POKE".
- ★ apelarea și punerea în execuție
- ★ modificări posibile ale datelor introduse în program

Varianta în limbaj de asamblare a programului poate fi introdusă în memorie cu ajutorul unui asamblor (de preferință GENS), iar varianta în cod mașină direct din interpretorul BASIC. Odată aflate în memorie, programele pot fi salvate sub formă de blocuri de cod (de exemplu, dacă s-au introdus la adresa 50000 programe în lungime totală de 2000 bytes, ele pot fi salvate cu

```
SAVE "nume" CODE 50000,2000
```

și încărcate cu

```
LOAD "nume" CODE
```

prin acest tip de păstrare evitându-se executarea, de fiecare dată când se rulează programul, a liniilor DATA (care acum devin inutile și pot fi șterse).

Pentru introducerea programelor scrise sub a doua variantă în memorie este necesar un algoritm simplu, care să preia un număr de valori dintr-o linie de DATA și să le depună în memorie, la adrese consecutive, folosind instrucțiunea POKE.

Datorită numărului mare de valori, se pot strecura greșeli în tastarea lor. Aceste greșeli sunt foarte periculoase, putând duce la proasta funcționare sau chiar la blocarea programului.

Pentru a evita această posibilitate, la unele programe (unde este posibil) se va testa suma totală a valorilor introduse cu cea pe care ar trebui să o aibă dacă au fost tastate corect. Prin "unde este posibil" înțelegem programele la care aceste valori sunt constante și nu se modifică. Pentru valori variabile, testul nu va mai funcționa.

Din acest motiv, vom folosi două subrutine de introducere a valorilor în memorie: una la linia 9997, de transfer de date cu test de corectitudine, și una la 9980, de transfer fără test.

La chemarea subrutinei, variabila S conține suma corectă a valorilor, X conține numărul de bytes ce trebuie transferați, ADR - adresa la care trebuie transferați, iar pointerul de date al interpretorului BASIC este fixat (cu instrucțiunea RESTORE) pe linia DATA ... necesară programului care trebuie introdus.

```
9997 LET SUMA=0:FOR Y=0 TO X-1:READ A:POKE ADR+Y,A
9998 LET SUMA=SUMA+A:NEXT Y:IF SUMA<>S THEN PRINT
"EROARE !":STOP
9999 RETURN
```

Subrutina fără test este:

```
9980 FOR Y=0 TO X-1:READ A:POKE ADR+Y,A:NEXT Y
9981 RETURN
```

Variabile printre aceste valori pot fi generate de exemplu de adresa unui program apelat prin CALL din cod mașină, adresă care poate să varieze și să fie schimbată în funcție de așezarea programului respectiv în memorie (această situație este întâlnită, de exemplu, la programele de repetare).

Noțiuni de bază

2.1 Diferența BASIC - cod mașină

Mai întâi, care este diferența între un program în BASIC și unul în cod mașină?
Am putea schematiza astfel:

Program BASIC

· Avantaje:

- ★ ușor de conceput
- ★ ușor de implementat
- ★ ușor de corectat

· Dezavantaje:

- ★ viteză foarte mică
- ★ nu permite accesul direct la resursele de bază ale calculatorului

Program în cod mașină:

· Avantaje:

- ★ viteză de lucru mare
- ★ posibilitatea folosirii la maximum a resurselor calculatorului

Dezavantaje:

- ★ greu de conceput
- ★ greu de corectat

După cum se poate vedea, calculatorul poate fi făcut mai "prietenos" pentru utilizator numai cu prețul unor avantaje esențiale, cum ar fi viteza de lucru.

Programul BASIC este mai ușor de implementat, mai ușor de corectat (execuția poate fi întreruptă în cazul unei erori și programul poate fi depanat), precum și mai ușor de conceput și înțeles. Are însă dezavantajul unei funcționări foarte lente, fiecare instrucțiune din limbajul de programare BASIC fiind interpretată și executată de subrutine lungi din memoria calculatorului.

Calculatoarele de acest tip dispun de un *interpretor* BASIC. Alte calculatoare, mai evoluate, dispun de *compilatoare*. Între aceste două noțiuni există o diferență majoră.

Compilerul preia instrucțiune cu instrucțiune programul scris în limbajul său (aici BASIC) și le traduce într-un program în cod mașină, care face ceea ce i s-a cerut programului BASIC. Odată parcurs programul, existența compilerului nu mai este necesară, programul putând funcționa și fără el.

Interpretorul lucrează în mai mulți pași, pentru majoritatea instrucțiunilor aceștia fiind:

- 1) caută linia curentă de executat;
- 2) dacă a trecut de ultima instrucțiune din linie, atunci sare la linia următoare;
- 3) caută instrucțiunea de executat;
- 4) testează dacă instrucțiunea are o formulare corectă (acesta este de fapt un al doilea test de acest tip, primul fiind efectuat la introducerea liniei);
- 5) caută dacă este o instrucțiune care lucrează cu variabile (de exemplu o atribuire, cum ar fi LET x=5);
- 6) dacă este nevoie de variabile, atunci le caută în memorie, pentru a le citi sau alocă spațiu în vederea scrierii lor; anunță "Variable not found" dacă

- instrucțiunea cere să caute pentru citire o variabilă care nu a fost încă scrisă;
- 7) execută instrucțiunea;
 - 8) dacă este nevoie, depune rezultatul în memorie sau execută alte secvențe necesare (de exemplu, instrucțiunea PLOT determină execuția, la sfârșitul ei, și a altor subrutine, de setare a atributelor cu care să apară punctul pe ecran, chiar dacă noi nu am avut intenția să-l colorăm);
 - 9) reia de la 1);

Observație:

Spre deosebire de compilator, interpretorul trebuie să fie tot timpul prezent, fără el execuția programului fiind imposibilă.

Este ușor de imaginat cât de mult timp s-ar câștiga executând numai pasul 7, adică numai executând pur și simplu o instrucțiune dorită, prin scrierea unui program în cod mașină, care să decupleze interpretorul. Acest program va avea avantajul vitezei, în schimb va fi mai greu de conceput, de schimbat, și practic ireparabil în cazul întâlnirii unei erori, în acest caz putând bloca sau chiar reseta sistemul.

Puteți să vă convingeți singuri de diferența de viteză, încercând un scurt program pentru acoperirea ecranului cu atributul de culoare "INK 0 , PAPER 0", dar fără ștergerea informației de pe el (aceasta devine numai invizibilă pentru utilizator, putând fi vizualizată prin reacoperirea ecranului cu un atribut cu INK diferit de PAPER, de exemplu atributul 7. Programul BASIC cel mai scurt și mai rapid ar fi următorul:

```
FOR N=22528 TO 23295:POKE N,0:NEXT N
```

Ați remarcat cât de încet a fost executat ? Acum încercați același program în cod mașină (este primul program prezentat la capitoul EFECTE DE CORTINĂ, cel de acoperire rapidă de sus în jos). Nici nu vă veți putea da seama de timpul de execuție, cu toate că a fost conceput după același algoritm.

2.2 Din ce este format un calculator

★ Memoria

Memoria calculatorului este o succesiune de elemente mici, numite bytes sau octeți. Fiecărui astfel de element îi este asociat un număr, numit adresă. Astfel, când spunem "locația 50325" înțelegem "al 50326-lea byte din memoria calculatorului". Octeții sunt numerotați de la 0 la 65535, și sunt împărțiți astfel:

- ★ de la 0 la 16383 se găsește memoria ROM (Read Only Memory), adică o porțiune în care nu se poate scrie informație, ci numai citi. Aici este amplasat interpretorul BASIC, care prelucrează programele scrise în acest limbaj. La unele calculatoare (cum ar fi CIP), memoria ROM lipsește, octeții din zona ei fiind accesibili și pentru scriere. În acest caz, interpretorul BASIC trebuie încărcat de pe casetă.
- ★ de la 16384 începe memoria RAM (Random Acces Memory), în care se poate scrie, însă al cărei conținut se pierde în cazul unei întreruperi a funcționării calculatorului. Primii 6912 octeți (de la 16384 la 23295) sunt reprezentarea ecranului. Aici se află memorată, în orice moment, imaginea care este recepționată pe monitor. Orice operație de scriere, desenare sau ștergere prelucrează de fapt octeții din această parte a memoriei.
- ★ de la 23296 la 23754 se găsește zona variabilelor sistem, unde interpretorul BASIC își depune informații temporare, cum ar fi titlul unui program la încărcare sau salvare, adrese de programe sau de variabile, informații despre modalitatea de afișare pe ecran (FLASH, BRIGHT, OVER sau INVERSE), etc.

Orice alterare a memoriei în această zonă poate avea consecințe grave, ducând până la resetarea calculatorului (unele programe folosesc acest lucru pentru protecție; de exemplu, prin schimbarea octetului de la adresa 23613 cu

valoarea 0, se poate obține resetarea sistemului în cazul în care se încearcă oprirea forțată, cu CAPS SHIFT + BREAK, a programului).

★ de la 23755 începe zona liberă a memoriei, zonă destinată programelor utilizatorului.

La rândul său, fiecare octet este format din 8 unități mai mici, numite biți. Fiecare bit poate lua două valori, 0 sau 1. Printr-un calcul simplu, se poate vedea că o grupare de 2 biți poate lua 4 valori (00,01,10,11), o grupare de 3 biți 8 valori și așa mai departe. Pentru o grupare de 8 biți, adică un byte, obținem 256 valori posibile.

Concluzie: într-un octet se poate stoca un număr cuprins între 0 și 255 inclusiv. Numerele mai mari decât 255 dar mai mici decât 65535 (cum ar fi o adresă din memorie) se memorează pe 2 bytes, astfel:

$$\text{număr} = \text{primul byte} + 256 * \text{al doilea byte}.$$

De exemplu, numărul 300 se poate scrie ca $44+1*256$, deci octeții pe care va fi memorat vor avea valorile 44 și 1.

Ca regulă de calcul, primul octet este restul împărțirii numărului la 256 iar al doilea este câtul acestei împărțiri.

★ *Microprocesorul*

O altă componentă de bază a calculatorului este microprocesorul. Acesta este de fapt "creierul" unui computer, el realizând toate operațiile cerute de utilizator, dar controlând și fluxul de informații între diversele părți ale memoriei, lucrul cu variabilele, etc.

Pentru a putea realiza aceasta, el cunoaște o serie de instrucțiuni, numite cod mașină. Limbajul acesta este de cel mai coborât nivel; cu el lucrează *masina*. Cu ajutorul lui, microprocesorul este determinat să "înțeleagă" și alte limbaje, ca BASIC, PASCAL, C și multe altele.

Practic, orice compilator realizează traducerea fiecărei instrucțiuni din limbajul pentru care este realizat în cod mașină, pentru a putea fi înțeleasă de către microprocesor și executată.

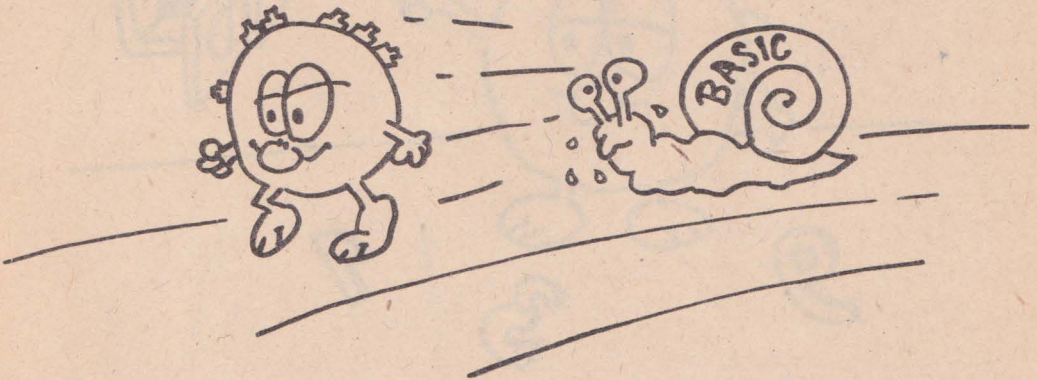
De exemplu, instrucțiunea PRINT "a" din BASIC este interpretată de rutine lungi din memoria ROM, realizând în final scrierea caracterului "a" pe ecran. Microprocesorul **nu are instrucțiuni pentru scriere, și nici nu știe literele**. Pentru el, o literă este doar o înșiruire de octeți în memorie. De exemplu litera "a", care arată astfel:

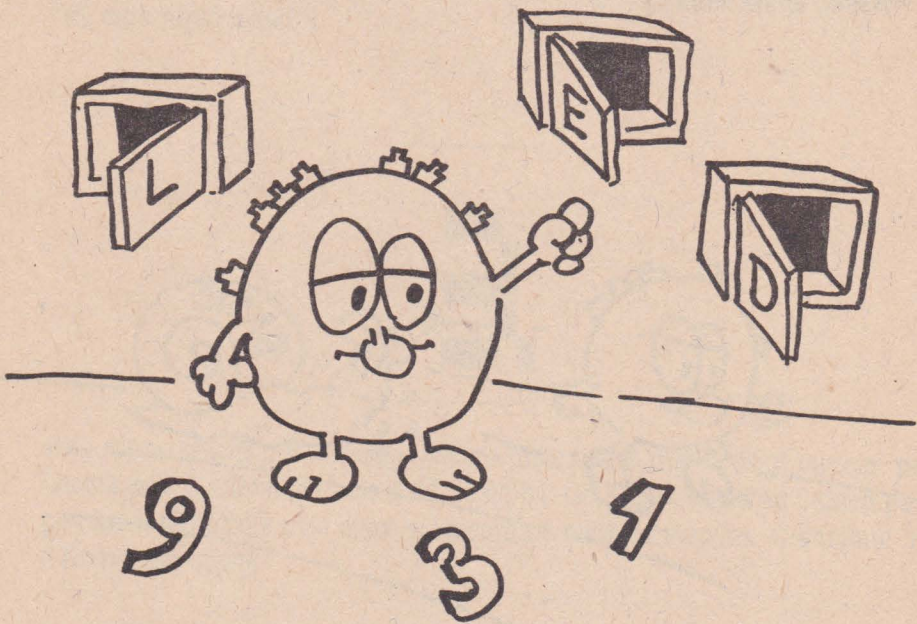


este reprezentată prin 8 octeți, fiecare însemnând o linie de 8 puncte: pentru fiecare punct, informația este dată de un bit. Un bit are valoarea 0 dacă punctul pe care îl reprezintă este stins, și valoarea 1 dacă este aprins. Caracterul "a" va fi format din octeții:

```
00000000  
00000000  
00111000  
00000100  
00111100  
01000100  
00111010  
00000000
```

Instrucțiunea PRINT determină, prin programele existente în ROM, transferul acestor 8 octeți de unde se găsesc în memorie, în memoria ecran, unul sub altul. Astfel, apare litera "a".





Introducere în cod mașină

3.1 Regiștrii

Pentru a putea lucra mai ușor cu memoria, microprocesorul dispune și el de câteva locații numite **regiștri**, unde își depune informația de prioritate imediată, și cu care poate lucra cu o viteză mult mai mare decât cu restul memoriei. Instrucțiunile limbajului cod mașină nu fac altceva decât să prelucreze acești regiștri.

Regiștrii sunt numiți cu litere: o singură literă pentru regiștri de 8 biți (pot memora numere între 0 și 255) și două litere pentru regiștri de 16 biți (care pot memora numere până la 65535). Astfel, regiștrii sunt:

- ★ simpli : A, B, C, D, E, F, H, L;
Regiștrul A (acumulator) este cel mai important; cu el se efectuează majoritatea operațiilor;
- ★ pereche : BC, DE, HL, AF - formați din asocieri de regiștri simpli; de exemplu, regiștrul BC conține numărul egal cu $256 \cdot B + C$;
SP, IX, IY, IP

Dintre aceștia, pot fi utilizați pentru operații doar **A, B, C, D, E, H, L, BC, DE, HL, SP, IX** și **IY**. Cu ajutorul lor, informația poate fi preluată din memorie, prelucrată și apoi transferată înapoi sau în altă zonă. Regiștrii pereche pot fi folosiți pentru a

adresa indirect variabile. De exemplu, instrucțiunea

LD A, (23606)

adică "încarcă în registrul A conținutul locației de memorie 23606" are același efect cu

LD BC, 23606
LD A, (BC)

adică "încarcă în BC adresa 23606 și apoi încarcă în A conținutul celulei de memorie a cărei adresă se află în BC".

Cu acumulatorul se pot efectua operații aritmetice (adunări sau scăderi), cum ar fi:

ADD A,8

adică "adună la numărul conținut în A numărul 8" sau

SUB 5

adică "scade din numărul conținut în A numărul 5".

Se pot efectua de asemenea operații logice:

★ AND, cu tabela de adevăr

0 AND 0 = 0

0 AND 1 = 0

1 AND 0 = 0

1 AND 1 = 1

Se observă că $x \text{ AND } y = 1$ doar dacă și x și y sunt 1 în același timp. Dacă avem în A numărul 19, adică biții 00010011, și efectuăm operația AND 254 (254 este succesiunea de biți 11111110) rezultatul va fi:

$$00010011 = 19 \text{ AND}$$

$$11111110 = 254$$

$$00010010 = 18$$

S-au păstrat doar biții care aveau valoarea 1 simultan și la 254, și la 19.

★ similar operațiile OR și XOR (eXclusive OR), cu tabelele de adevăr caracteristice:

$$0 \text{ OR } 0 = 0$$

$$0 \text{ OR } 1 = 1$$

$$1 \text{ OR } 0 = 1$$

$$1 \text{ OR } 1 = 1$$

$$0 \text{ XOR } 0 = 0$$

$$0 \text{ XOR } 1 = 1$$

$$1 \text{ XOR } 0 = 1$$

$$1 \text{ XOR } 1 = 0$$

Alte operații ce se pot efectua cu acumulatorul sunt deplasări de biți (shiftări) la dreapta sau la stânga. Biții sunt numerotați de la dreapta la stânga, astfel:

7 6 5 4 3 2 1 0

1 0 1 0 1 1 0 1

Dacă efectuăm o deplasare la dreapta (SRA A sau SRL A) a octetului de mai sus, înseamnă să deplasăm toți biții câte o poziție la dreapta:

7 6 5 4 3 2 1 0

0 1 0 1 0 1 1 0

În locul rămas liber se introduce automat valoarea 0 (dacă am folosit SRL A) sau rămâne nemodificat (dacă am folosit SRA A). Dacă însă efectuăm o deplasare la stânga (SLA A), bitul 0 primește valoarea 0:

```

7 6 5 4 3 2 1 0
-----
0 1 0 1 1 0 1 0

```

iar bitul 7 se pierde.

Se pot efectua de asemenea rotiri (RL- stânga, RR- dreapta), când bitul ce dispăre este pus în locul ce a rămas liber în capătul celălalt.

Operații de deplasare mai pot fi efectuate și asupra locațiilor de memorie adresate de regiștrii HL, IX și IY:

```

SRL (HL)
SLA (IX+00)
RR (IY+01)

```

Poate însă cea mai importantă operație la care poate fi folosit acumulatorul este cea de decizie. După orice operație efectuată asupra lui, putem testa rezultatul operației, cu ajutorul unor indicatori numiți *flags* (stegulețe). Astfel, instrucțiunile

```

AND 7
JR Z, ETI_1

```

au ca efect saltul la eticheta ETI_1 dacă operația AND 7 a avut ca efect obținerea rezultatului 0 în acumulator.

Putem folosi comparația:

```

CP 7

```


adică "compară conținutul lui A cu 7"
și imediat după ea salturile condiționate:

JR Z,.. - salt dacă A este 7;
JR NZ,..- salt dacă A nu este 7;
JR C,.. - salt dacă A este mai mic decât 7;
JR NC,..- salt dacă A nu este mai mic decât 7;

și prin aceste salturi să direcționăm algoritmul după dorința noastră. De exemplu, dorim să testăm dacă în octetul de la adresa 30000 este numărul 5:

```
LD A, (30000)
CP 5
JR Z, DA
JR NZ, NU
```

sau dacă bitul 5 al acumulatorului este "0"

```
BIT 5,A
JR Z,DA
JR NZ,NU
```

Putem realiza repetarea unei secvențe de instrucțiuni, folosind registrul B:

	LD B,9	- numărul de repetări
ETI_1	INC A	- mărește acumulatorul
	DEC DE	- micșorează DE
	LD (DE),A	- încarcă la adresa din DE conținutul lui A
	DJNZ ETI_1	- micșorează B cu 1 și, dacă nu a ajuns la 0, reia de la eticheta ETI_1.

Cu ajutorul registrului SP se plasează în memorie o zonă importantă pentru procesor, numită **stivă**. Aici se pot salva conținutul regiștrilor sau adrese, respectându-se o regulă: ultima valoare pusă în stivă este prima valoare scoasă.

stiva trebuie să fie întotdeauna liberă înainte de a se ieși din subrutina curentă.

3.2 O comparație BASIC - cod mașină

Am putea stabili o similaritate între anumite instrucțiuni din codul mașină și unele din limbajul BASIC, după cum se vede în tabelele următoare:

★ *Instrucțiuni pentru acumulator:*

Cod mașină	BASIC
LD A,1	LET A=1
LD A,X	LET A=X
INC A	LET A=A+1
DEC A	LET A=A-1
ADD A,3	LET A=A+3
SUB 3	LET A=A-3
ADD A,B	LET A=A+B

Cod mașină	BASIC
SUB B	LET A=A-B
CP 8	
JR NZ, ETI1	IF A=8 THEN GOTO ETI1
CP 17	
JR NC, ETI2	IF A>=17 THEN GOTO ETI2
CP 27	
JR C, ETI3	IF A<27 THEN GOTO ETI3
CP X	
JR Z, ETI4	IF A=X THEN GOTO ETI4

OBSERVAȚIE IMPORTANTĂ: în acumulator, ca și în oricare registru simplu (reprezentat printr-o singură literă) nu se poate memora decât un număr natural cuprins între 0 și 255 inclusiv. Dacă la o adunare sau scădere se iese din aceste limite, atunci rezultatul va fi restul împărțirii sale la 256.

De exemplu, memorăm în A numărul 5 (LD A,5) și scădem un număr mai mare, să zicem 200 (SBC A,200). Rezultatul este -195, dar, cum în acumulator nu se poate reprezenta un număr negativ, în el se va afla numărul 256-195, adică 61.

Pentru a ne putea da seama însă că "ceva s-a întâmplat", indicatorul C (carry) va fi poziționat pe 1, permițând astfel un salt condiționat (de exemplu JR C, ETI4 determină saltul la eticheta ETI4 dacă prin scădere s-a obținut un număr negativ).

Acest indicator folosește și la adunare, numai că aici va indica o depășire a

limitei de 255. De exemplu, dacă la $A=60$ adunăm 250, rezultatul va fi $250+60=310$, dar în acumulator va fi înmagazinat numărul $310-256=54$, iar indicatorul C va fi poziționat pe 1.

Exemplu:

10 LD A, 250	- încarcă în A numărul 250.
20 ADD A, B	- adună numărul conținut în registrul B.
30 JR C, MAI_MARE	- dacă rezultatul este mai mare decât 255, atunci sare la eticheta MAI_MARE.
40 JR NC, M_M_S_E	- dacă este mai mic sau egal cu 255 (deci în A este memorat chiar rezultatul), atunci sare la eticheta M_M_S_E.

După o comparație (CP operand), putem programa sărituri condiționate folosind indicatorii CARRY (depășire) sau ZERO (egalitate), astfel:

JR Z, ...	- salt dacă A = operand.
JR NZ, ...	- salt dacă A \neq operand.
JR C, ...	- salt dacă A < operand.
JR NC, ...	- salt dacă A \geq operand.

★ Instrucțiuni pentru regiștri simpli

Vom nota regiștrii simpli cu X sau cu Y, prin aceasta înțelegându-se orice registru dintre B,C,D,E,H,L):

3. INTRODUCERE ÎN COD MAȘINĂ

Cod mașină	BASIC
LD X,1	LET X=1
INC X	LET X=X+1
DEC X	LET X=X-1
LD A,X ADD A,3 LD X,A	LET X=X+3
LD A,X SUB 3 LD X,A	LET X=X-3
LD A,X ADD A,Y LD X,A	LET X=X+Y
LD A,X SUB Y LD X,A	LET X=X-Y
LD A,X CP 8 JR Z, ETI1	IF X=8 THEN GOTO ETI1
LD A,X CP Y JR Z, ETI4	IF X=Y THEN GOTO ETI4

OBSERVAȚIE: pentru unele operații cum ar fi adunări, scăderi, comparații, nu există instrucțiuni de lucru direct cu registrul. Acestea se pot efectua numai cu acumulatorul. De aceea operațiile se pot face în 3 pași:

a) încărcăm conținutul registrului în acumulator (**ATENȚIE !** orice este înmagazinat înainte în acumulator se alterează, așa că dacă avem informație importantă trebuie salvată înainte de a se face încărcarea, prin punerea într-un registru pe care nu-l vom folosi cu LD X,A sau prin transferul în stivă, cu PUSH AF).

b) efectuăm operația dorită asupra acumulatorului.

c) transferăm rezultatul înapoi în registrul în care dorim să fie memorat (dacă s-a efectuat altceva decât o comparație, deoarece la comparații oricum nu se modifică valorile).

De exemplu, pentru compararea valorii din registrul H cu 0:

- (a) LD A,H
- (b) CP 0
- JR Z,DA - dacă este 0, sare la eticheta DA.
- JR NZ,NU - dacă este diferit de 0, sare la eticheta NU.

Pentru a testa dacă L este mai mare decât 6, este necesar programul:

- (a) LD A,L
- (b) CP 6
- JR C, L_MAI_MIC_DECIT_6
- JR NC, MAI_MARE_SAU_EQUAL

Pentru a scădea din valoarea conținută în D numărul 30:

- (a) LD A,D
- (b) SUB 30
- (c) LD D,A

★ Instrucțiuni pentru registrul pereche HL

La fel cum acumulatorul permite câteva operații în plus pentru regiștrii simpli, HL permite câteva operații pentru regiștrii pereche). În cele ce urmează, vom nota cu YY sau ZZ orice registru pereche dintre BC, DE, HL:

Cod mașină	BASIC
LD HL,16000	LET HL=16000
INC HL	LET HL=HL+1
DEC HL	LET HL=HL-1
INC H	LET HL=HL+256
DEC H	LET HL=HL-256
LD HL,ZZ OR A ADC HL,BC LD ZZ,HL	LET ZZ=ZZ+BC
LD HL,ZZ OR A SBC HL,DE	

LD ZZ,HL

LET ZZ=ZZ-DE

LD HL,ZZ

ADC HL,YY

LD ZZ,HL

LET ZZ=ZZ+YY

OBSERVAȚII: instrucțiunea SBC (SUBtract with Carry) scade, pe lângă valoarea celui de-al doilea operand, și numărul 1 dacă CARRY este poziționat, de aceea trebuie să îl aducem la 0 înainte de a executa scăderea. Aceasta se poate face fie cu instrucțiunile SCF (Set Carry Flag) și CCF (Complement Carry Flag), deci punându-l pe 1 și apoi complementându-l, fie cu o instrucțiune al cărei efect este și ștergerea lui (care îl modifică în 0), cum ar fi OR A.

Registrul pereche HL este format din regiștrii simpli H și L, el conținând de fapt numărul dat de $256 * H + L$. La fel, registrul BC este format din B și C, iar registrul DE din D și E. Printr-un calcul simplu, se vede că într-un registru pereche pot fi înmagazinate numere de la 0 la 65535 inclusiv.

Spre deosebire de regiștrii simpli, regiștrii pereche nu pot lucra direct cu numere (de exemplu, nu există operația ADC HL,300, ci numai ADC HL,DE sau ADC HL,BC). Pentru efectuarea lor este necesar să transferăm numerele într-unul din regiștrii pereche BC sau DE, și să efectuăm cu aceștia operația asupra lui HL.

În cazul de mai sus, dacă dorim să adunăm la numărul conținut în HL numărul 300, va trebui să încărcăm mai întâi 300 într-unul din regiștrii pereche amintiți și apoi să efectuăm adunarea:

LD BC,300

ADC HL,BC

aceasta fiind singura soluție. La fel pentru scădere (scădem din HL conținutul registrului DE sau BC).

Depășirea limitelor la adunare sau scădere ne va fi dată tot de indicatorul C (carry), rezultatul fiind dat în același mod: dacă după o scădere, rezultatul din HL trebuie să fie mai mic decât 0, acesta va conține (65536-rezultat), iar C va fi poziționat ca să ne anunțe depășirea limitei; dacă se depășește limita superioară la o adunare, HL va conține (rezultat-65536), iar C va fi de asemenea poziționat.

Pentru regiștrii pereche nu avem nici o instrucțiune de comparare (de exemplu, nu este acceptată o instrucțiune de genul CP HL,450 sau CP HL,BC). Compararea se face prin scădere sau prin recurgerea la regiștrii simpli din care este format registrul pereche respectiv.

Exemplu: compararea lui HL cu BC. Evident, dacă HL este mai mare decât BC, atunci la scădere rezultatul va fi pozitiv, deci indicatorul C nu va fi poziționat. Dacă rezultatul va fi negativ, atunci indicatorul C va marca aceasta. Înainte de a face scăderea, ștergem indicatorul CARRY, cu instrucțiunea OR A.

```
OR A
SBC HL,BC
JR C,MAI_MIC
JR NC,MAI_MARE_SAU_EGAL
```

Observație: la regiștrii simpli, prin comparare nu se alterează conținutul acumulatorului, pe când aici, dacă efectuăm scăderea, conținutul lui HL va fi pierdut.

Pentru a evita acest lucru, trebuie să-l salvăm în stivă înainte de scădere și să-l refacem imediat după aceea:

```
PUSH HL
OR A
SBC HL,BC
POP HL
```

```
JR C,MAI_MIC
JR NC,MAI_MARE_SAU_EQUAL
```

Această operație nu modifică cu nimic decizia, deoarece la scoaterea din stivă nu se schimbă starea indicatorului C.

Pentru compararea unui registru pereche cu o valoare diferită de 0, trebuie să recurgem la următoarea suită de instrucțiuni:

- a) încărcăm valoarea registrului în HL.
- b) încărcăm valoarea cu care dorim să-l comparăm în BC.
- c) comparăm registrul HL cu BC (ca mai sus).

Presupunem că dorim să comparăm conținutul registrului DE cu 700. Programul va fi următorul:

```
(a) LD H,D
    LD L,E
(b) LD BC,700
(c) PUSH HL
    SBC HL,BC
    POP HL
    JR C,MAI_MIC
    JR NC,MAI_MARE_SAU_EQUAL
```

Pentru a compara un registru cu 0, va fi nevoie să îl desfacem în regiștrii simpli componenți și să testăm dacă ei sunt simultan 0:

- a) compară primul registru component cu 0.
- b) dacă nu este 0, atunci sare la eticheta NU.

- c) altfel compară al doilea registru cu 0.
- d) dacă nu este 0, atunci sare la eticheta NU.
- e) altfel, sare la eticheta DA.

Compararea regiștrilor simpli cu o valoare o vom face ca la punctul (2), adică încărcându-i în acumulator și comparând acumulatorul cu acea valoare (0).

De exemplu, pentru compararea lui HL cu 0:

- (a) LD A,H - încarcă primul registru component în acumulator.
CP 0 - îl compară cu 0.
- (b) JR NZ,NU
- (c) LD A,L - încarcă al doilea registru component în acumulator.
CP 0 - îl compară cu 0.
- (d) JR NZ,NU
- (e) JR Z,DA

Compararea cu 0 se poate face mai simplu folosind instrucțiunea OR (SAU logic) între cei doi regiștri simpli ce compun registrul pereche; rezultatul va fi 0 numai dacă ambii sunt 0. Astfel, programul de mai sus se poate scrie și în forma următoare:

- LD A,H - încarcă conținutul registrului H în A.
- OR L - execută SAU logic între A și registrul L.
- JR Z, DA - dacă rezultatul a fost 0, sare la eticheta DA.
- JR Z, NU - dacă nu, atunci sare la eticheta NU.

★ *Instrucțiuni care prelucrează locații de memorie*

Prin locație de memorie înțelegem unul din octeții care se află în memoria calculatorului, numerotat printr-o adresă de la 0 la 65535. Adresa o vom nota prin

NN, iar conținutul ei prin (NN).

Cod mașină	BASIC
LD A,(50000)	LET A=PEEK (50000)
LD (50000),A	POKE 50000,A
LD A,(HL)	LET A=PEEK (HL)
LD (HL),50 INC (HL)	POKE HL, 50 POKE HL, PEEK (HL)+1 (mărește cu 1 valoarea de la adresa HL)
ADD A,(HL)	LET A=A+PEEK (HL)
SUB (HL)	LET A=A-PEEK (HL)
CP (HL) JR Z, ETI1	IF A=PEEK (HL) THEN GOTO ETI1

OBSERVAȚII: locațiile de memorie pot fi tratate ca un registru simplu, prin încărcarea adresei lor în HL și folosirea, în calcule, a formei "(HL)". De exemplu, programul:

```
LD HL,40000
LD (HL),5
INC HL
LD (HL),20
```

este echivalent cu

POKE 40000,5
POKE 40001,20

adică memorează în octetul de la adresa 40000 numărul 5 iar în octetul de la adresa 40001 numărul 20.

Nu există o instrucțiune de încărcare directă a unui octet de la o adresă cu un număr, ca LD (50000),30 , similară lui POKE 50000,30 din BASIC. Pentru aceasta este nevoie să încărcăm numărul 30 în acumulator, și pe urmă să încărcăm acumulatorul la adresa 50000:

LD A,30
LD (50000),A

Locațiile de memorie pot fi adresate nu numai prin HL, ci și prin regiștrii IX și IY, ei permițând un domeniu mai mare de adresare. De exemplu, există instrucțiunea

LD (IX+n), A

unde n este un număr cuprins între -128 și 127, el reprezentând distanța, în octeți, a adresei căutate față de adresa din IX.

Exemplu:

LD IX, 50000
LD H, (IX+40) - încarcă în registrul H valoarea de la adresa (IX+40=50040).
LD E, (IX-30) - încarcă în registrul E valoarea de la adresa (IX-30=49970).

este echivalent cu

- LD A, (50040) - încarcă în acumulator conținutul adresei 50040.
 LD H,A - încarcă în H conținutul acumulatorului.
 LD A, (49970) - încarcă în acumulator conținutul adresei 49970.
 LD E,A - încarcă în registrul E valoarea din acumulator.

★ *Instrucțiuni care lucrează cu cuvinte de memorie*

Prin cuvânt de memorie înțelegem o succesiune de 2 octeți din memorie calculatorului, în care este memorat un număr cuprins între 0 și 65535 inclusiv Vom nota adresa acestei succesiuni prin NN, iar conținutul ei prin (NN).

Observație: nu este nici o diferență între adresarea unui cuvânt de memorie și o locație de memorie, diferența făcându-se la nivel de instrucțiune: dacă operandul respectiv trebuie să fie pe 8 biți, atunci (NN) va fi luat ca locație, dar dacă operandul trebuie să fie pe 16 biți (2 octeți) atunci (NN) va fi luat ca cuvânt.

Astfel:

LD A,(NN)

cere operanzi pe 8 biți, deci simplă locație (acumulatorul are 8 biți, deci nu poate înmagazina decât un număr tot pe 8 biți). Dacă însă avem

LD HL, (NN)

atunci operanzii trebuie să aibă 16 biți, deci (NN) va fi luat ca cuvânt. Efectul instrucțiunii de mai sus este încărcarea în registrul L a conținutului locației de memorie de la adresa NN și încărcarea în registrul H a conținutului locației de la adresa NN+1. În informatică, aceste valori sunt denumite **Least Significant Byte** sau, prescurtat, **LSB** (prima, cea de la adresa NN, sau registrul L) și **Most**

Significant Byte sau MSB (a doua, cea de la adresa NN+1, sau registrul H). Denumirile înseamnă "cel mai puțin semnificativ" respectiv "cel mai semnificativ byte". Aceasta pentru că valoarea conținută în acești 2 bytes este egală cu $LSB + MSB * 256$. Deci, dacă avem o valoare și o reprezentăm pe 2 octeți, primul, adică LSB, va conține restul împărțirii valorii la 256, iar MSB va conține câtul acestei împărțiri.

Exemplu: vrem să punem în memorie, pe 2 octeți, valoarea 40000:

```
LD HL,40000
LD (65000),HL
```

Înmagazinarea s-a făcut la adresa 65000, valorile celor doi octeți devenind:

$(65000) = \text{restul împărțirii lui } 40000 \text{ la } 256 = 40000 - 256 * \text{INT}(40000/256) = 64$
 $(65001) = \text{câtul împărțirii lui } 40000 \text{ la } 256 = \text{INT}(40000/256) = 156$

Este deja știut că valoarea lui HL este dată de formula $256 * H + L$, deci exact $256 * MSB + LSB$.

Important ! Prin convenție, întotdeauna, într-un cuvânt de memorie, LSB se află în prima locație, iar MSB în a doua.

Cod mașină	BASIC
LD HL,(50000)	LET HL=PEEK (50000) + 256 * PEEK (50001)
LD (50000),HL	POKE 50000,HL-256* INT(HL/256): POKE 50001, INT(HL/256)

În tabelul de mai sus, în loc de HL se poate pune și BC sau DE.

★ Instrucțiuni de salt

Cod mașină	BASIC
JR ET11	GOTO ET11
LD A,X CP 3 JR NZ,ET12	IF X<>3 THEN GOTO ET12

Instrucțiunile de salt din cod mașină sunt în general echivalente cu instrucțiunea GOTO din BASIC, ele realizând saltul la o anumită adresă, tot așa cum GOTO realizează saltul la o anumită linie. Există instrucțiuni de salt condiționat, adică saltul se face în funcție de un rezultat anterior, prin testarea indicatorilor de condiție. Cei mai folosiți sunt CARRY (C) și ZERO (Z).

Exemplu:

După secvența:

```
LD A,X
CP 3
```

se poate folosi:

```
JR Z, ET11 - egalitate;
JR NZ, ET12 - inegalitate;
JR C, ET13 - X < 3
JR NC, ET14 - X ≥ 3
```


★ *Alte instrucțiuni:*

★ instrucțiunea DJNZ (Decrement and Jump if Not Zero - micșorează și sari dacă nu a ajuns la 0). Instrucțiunea este o instrucțiune de salt condiționat, dar servește și la ciclare. Efectul ei este:

- ★ micșorează valoarea din registrul B cu 1;
- ★ testează dacă a ajuns la 0;
- ★ dacă nu, atunci execută saltul comandat;
- ★ dacă da, trece mai departe.

Principala utilizare este pentru repetarea unei secvențe de instrucțiuni. Astfel, programul:

```
LD B, 70
ETI1 INC HL
      DJNZ ETI1
```

va produce repetarea de 70 de ori a instrucțiunii INC HL. Programul echivalent în instrucțiuni BASIC ar fi:

```
FOR B=1 TO 70
LET HL=HL+1
NEXT B
```

La ieșirea din buclă, HL va fi, evident, mai mare cu 70 decât valoarea pe care o avea înainte.

Programul:

```
LD HL,50000
```

```
LD B,100
ETI1 LD (HL),0
      INC HL
      DJNZ ETI1
```

va avea ca efect repetarea de 100 de ori a instrucțiunilor "LD HL,0" și "INC HL". Programul BASIC ar fi fost:

```
LET HL=50000
FOR B=1 TO 100
POKE HL,0
LET HL=HL+1
NEXT B
```

și ar fi avut același efect: umplerea unei zone de memorie de 100 octeți, începând la adresa 50000, cu valori de 0 (deci memorarea valorii 0 la adresele 50000-50099).

Atenție ! Multe din cauzele blocării programelor este proiectarea unor bucle infinite. De exemplu, schimbarea valorii lui B în interiorul unui ciclu ca cel de mai sus ar putea duce la blocaj, deoarece s-ar putea ca aceasta să nu mai atingă niciodată valoarea 0.

★ instrucțiunea LDIR (LoaD, Increment and Repeat - încarcă, mărește și repetă)

Această instrucțiune este folosită de obicei la transferul blocurilor de memorie. Este tot un ciclu, însă de data aceasta nu este nevoie de precizarea unei adrese de salt, instrucțiunea în sine producând execuția unui ciclu. De fapt, ea urmărește pașii prezentați mai jos:

★ încarcă octetul de la adresa memorată în registrul DE cu cel de la adresa memorată în registrul HL, ceea ce este echivalent cu instrucțiunile:

```
PAS LD A,(HL)
      LD (DE),A
```

★ mărește conținuturile lui HL și DE cu 1:

```
INC HL
INC DE
```

★ micșorează conținutul registrului BC cu 1:

```
DEC BC
```

★ testează dacă BC a ajuns la 0:

```
LD A,B
OR C
```

★ și dacă nu, atunci repetă de la primul pas, până când BC este 0:

```
JR NZ, PAS
```

Astfel, se realizează copierea a BC octeți de la adresa din HL la adresa din DE, echivalent cu programul BASIC:

```
10 FOR N=1 TO BC
20 POKE DE,PEEK(HL)
30 LET HL=HL+1
40 LET DE=DE+1
50 NEXT N
```

însă cu o viteză mult mai mare. Dacă, de exemplu, avem în DE valoarea 50000, în HL valoarea 20000 iar în BC valoarea 10000, instrucțiunea LDIR va produce copierea a 10000 bytes de la adresa 20000 la adresa 50000.

Exemplu: un program care realizează copierea memoriei ecran (6912 bytes, începând de la adrese 16384) în altă parte a memoriei, să zicem la adresa 45000:

```
LD DE,45000  ★ adresa de destinație - memoria
LD HL,16384  ★ adresa de origine - ecranul
LD BC,6912   ★ numărul de octeți ce trebuie transferați
LDIR         ★ se execută transferul
```

După rularea programului de mai sus, ecranul se află copiat în memorie, la adresa 45000. Chiar dacă ștergem cu CLS informația de pe monitor, o putem readuce înapoi cu transferul invers (din memorie în zona ecranului). Pentru aceasta este suficient să inversăm regiștrii DE și HL (destinația și sursa):

```
LD DE,16384  ★ adresa de destinație - ecranul
LD HL,45000  ★ adresa de origine - memoria
LD BC,6912   ★ numărul de octeți ce trebuie transferați
LDIR         ★ se execută transferul
```

3.3 Stiva

★ Cum se folosește

Având la dispoziție destul de puțin regiștri de lucru, de multe ori valorile acestora vor trebui păstrate temporar în memorie, pentru a-i folosi la alte operații. De exemplu, proiectarea unui ciclu cu DJNZ duce la alterarea valorii registrului BC, care poate conține informații importante. De aceea, el trebuie păstrat undeva, până vom avea nevoie din nou de acele informații.

3. INTRODUCERE ÎN COD MAȘINĂ

Avem două soluții: fie îl depunem la o adresă fixă în memorie, cu LD (NN),BC, fie folosim o facilități a procesorului dezvoltată special în acest scop: *stiva*.

Aceasta este creată în așa fel, încât preia controlul asupra adreselor la care se face salvarea, și nu trebuie să le mai controlăm noi. Ea funcționează pe principiul "LAST IN FIRST OUT" adică întotdeauna vom scoate din stivă ce am pus ultima oară. Pentru înțelegere, vom da un exemplu.

Să presupunem că dorim să punem în stivă registre ce conțin valorile 30000,20500,50000,800. Stiva va arăta astfel:

★ punem valoarea 30000:	30000 ↑
★ punem valoarea 20500:	20500 30000 ↑
★ punem valoarea 50000:	50000 20500 30000 ↑
★ punem valoarea 800:	800 50000 20500 30000 ↑

Am reprezentat prin ↑ vârful stivei. Acesta indică întotdeauna ultima valoare intrată, adică prima valoare ce va fi obținută la ieșire. Operația de intrare este PUSH ZZ, unde ZZ este orice registru pereche (nu pot fi puși în stivă regiștri simpli) dintre {BC,DE,HL,AF,IX,IY} și realizează punerea, în vârful stivei, a valorii conținute de registrul ZZ (**atenție!** nu se memorează și **ce registru** a conținut această valoare !).

Operația de ieșire este POP ZZ , unde ZZ este un registru pereche, ca la cazul anterior, și reprezintă registrul care va prelua valoarea din vârful stivei.

Exemplu: avem următorul șir de operații:

OPERAȚIA	EFECTUL ASUPRA STIVEI	EFECTUL ASUPRA REGIȘTRILOR
----------	-----------------------	----------------------------

LD BC,50000		BC=50000
-------------	--	----------

PUSH BC	50000	
---------	-------	--

↑

LD DE,45000	50000	DE=45000
-------------	-------	----------

↑

PUSH DE	45000 50000	
---------	-------------	--

↑

LD BC,400	45000 50000	BC=400
-----------	-------------	--------

↑

OPERAȚIA	EFECTUL ASUPRA STIVEI	EFECTUL ASUPRA REGIȘTRILOR
----------	-----------------------	----------------------------

PUSH BC	400 45000 50000	
---------	-----------------	--

↑

POP DE	45000 50000	DE=400
--------	-------------	--------

↑

POP HL	50000	HL=45000
--------	-------	----------

↑

POP BC		BC=50000
--------	--	----------

Stiva este folosită de microprocesor și pentru memorarea adresei de întoarcere dintr-o procedură apelată cu CALL. La o asemenea instrucțiune, microprocesorul pune adresa instrucțiunii imediat următoare în vârful stivei și execută un salt la adresa indicată. Dacă, de exemplu, s-a ajuns cu execuția unui program la adresa 30100 și s-a întâlnit o instrucțiune de CALL:

30100 CALL 56000	★ încarcă în stivă adresa instrucțiunii următoare (adică a instrucțiunii LD HL,50100), respectiv numărul 30102
------------------	--

și execută saltul la adresa 56000 (JP 56000).

```
30102 LD HL,50100
```

...

Odată ajuns la adresa 56000, procesorul va executa instrucțiunile găsite, până la întâlnirea unei instrucțiuni RET (ântoarcere din subrutină). Atunci el descarcă cuvântul găsit în vârful stivei și execută saltul la adresa reprezentată în acesta, adică, **dacă stiva a rămas neschimbată**, va descărca valoarea 30102 și va executa JP 30102.

Este important ca stiva să fie, în momentul întoarcerii din procedură, la fel cum era în momentul apelării acesteia, altfel se va realiza întoarcerea la o adresă greșită și programul se va strica definitiv. Dacă, de exemplu, la adresa 56000 vom avea următoarele instrucțiuni:

```
56000 LD HL,40000
      PUSH HL
      RET
```

vârful stivei în momentul întâlnirii instrucțiunii RET nu va conține valoarea 30102, ci valoarea depusă cu PUSH HL, adică 40000, și la această adresă se va face saltul. Cu alte cuvinte, la revenirea din procedură execuția va continua nu la adresa 30102, unde era următoarea instrucțiune, ci la adresa 40000 unde se poate afla cu totul altceva !

De asemenea, se poate produce confuzie și la o secvență de instrucțiuni de genul:

```
????? LD HL,29500   => HL=29500
      PUSH HL       => stiva: 29500
      ...
30000  CALL 56000   => stiva: 30102,29500; sare la 56000 (JP 56000)
30102  LD HL,50100
```

...
56000 POP HL ⇒ stiva: 29500; HL=30102 !!

...
????? RET ⇒ sare la 29500 !!

pentru că POP HL nu va aduce valoarea pusă cu PUSH, ci pe cea aflată în vârful stivei după execuția instrucțiunii CALL, adică 30102, iar revenirea se va face ... la adresa pusă în stivă cu instrucțiunea PUSH HL !

De aceea, reânnoim avertismentul, și îl vom face pe tot parcursul cărții: stiva primită la apelarea cu CALL a unei proceduri trebuie să fie găsită "curată" când se face revenirea cu RET !

Dacă condiția de mai sus este respectată, instrucțiunile CALL și RET din cod mașină pot fi asimilate cu instrucțiunile GOSUB și RETURN din BASIC.

★ *Cum este construită*

Din punct de vedere al construcției, stiva este o bucată de memorie, cu adresa de început în registrul SP (Stack Pointer). Când în stivă se depune un element, SP este micșorat cu 2, iar în cuvântul de memorie a cărui adresă o conține acum va fi pusă valoarea registrului comunicat cu PUSH. Astfel, PUSH HL este echivalent, ca efect, cu instrucțiunile:

```
DEC SP
DEC SP
LD (SP),HL
```

Adresa stivei poate fi inițializată de noi, la o valoare dorită. Dacă, de exemplu, începem un program cu LD SP,50000 , atunci primul registru pus în stivă cu PUSH se va afla la adresa 50000, al doilea la adresa 49998, al treilea la 49996 și așa mai departe. în SP se va afla întotdeauna adresa ultimei valori depuse în

stivă:

```
LD HL,40000
PUSH HL      ⇒ SP=49998, (49998)=40000
LD HL,30000
PUSH HL      ⇒ SP=49996, (49996)=30000
POP BC       ⇒ SP=49998, BC=30000
POP DE       ⇒ SP=50000, DE=40000
```

Am notat prin (49998) și (49996) conținutul cuvântului de memorie ce începe la adresa 49998, respectiv 49996.

Observați că scoaterea din stivă se face prin operația inversă: cuvântul de memorie adresat de SP este depus în registrul destinație iar SP este mărit cu 2 (ajungând la adresa valorii puse anterior).

IMPORTANT ! Ieșirea dintr-un program scris în cod mașină și revenirea în BASIC se face cu o instrucțiune RET, deci aceasta trebuie să fie nelipsită din orice program !

3.4 Cum se folosește asamblorul

Asamblorul este un program care, odată încărcat, permite scrierea și execuția programelor în limbaj de asamblare. În plus, el realizează și conversia din limbaj de asamblare în cod mașină, depunând programul rezultat la o adresă indicată de utilizator.

Programele care sunt prezentate în această carte au fost rulate pe un calculator Sinclair ZX Spectrum, sub asamblorul GENS. Acest asamblor circulă pe casete, sub forma unui bloc de cod sau sub formă de program, împreună cu

dezasamblorul MONS (programul GENS-MONS).

Dacă aveți intenția să încercați variantele în limbaj de asamblare ale programelor din această carte, recomandăm încărcarea asamblorului la adresă 26000, cu instrucțiunea:

```
LOAD "GENS" CODE 26000
```

Lansarea în execuție se face cu comanda

```
RANDOMIZE USR 26000
```

Pe ecran va apare un prompter (">"), semn că asamblorul vă așteaptă comanda. Din acest moment, se poate începe introducerea liniilor programului dorit, ca în exemplele de mai jos.

★ Liniile fără etichete, ca de exemplu:

```
10   ORG 50000
```

se introduc cu succesiunea de taste

"10", CAPS SHIFT și 8 (săgeată dreapta), "ORG 50000", CR (ENTER)

★ Liniile cu etichete, cum ar fi

```
20 L1. LD HL,16384
```

necesită secvența

"20 L1", CAPS SHIFT și 8, "LD HL,16384", CR

Dacă, înainte de introducerea unor linii, se introduce de la prompter comanda !:

>I

atunci liniile vor fi numerotate automat, din 10 în 10, nemaifiind necesară scrierea numărului lor.

Se poate modifica distanța între linii și numărul primei linii, punându-le după I. De exemplu, comanda >I8,5 va determina numerotarea automată a liniilor din 5 în 5, începând de la linia 8.

După ce s-a terminat scrierea unei linii se trece la următoarea, până când se apasă EDIT (CAPS SHIFT și 1). La apăsarea acestor taste se revine în modul prompter, iar programul este păstrat în memorie.

Listarea unui program poate fi obținută din modul prompter, cu comanda L:

>L

Aceasta produce listarea întregului program. Pentru listarea unei singure părți, se acceptă comenzi de genul >L100,200 , aceasta realizând listarea liniilor numerotate între 100 și 200 inclusiv. Comanda L100 produce listarea liniilor care urmează liniei 100, până la sfârșitul programului.

Pentru ștergerea unei linii tastăm numărul liniei și apăsăm ENTER (ca în BASIC). De exemplu, pentru ștergerea liniei 130 se dă comanda

>130

Pentru ștergerea mai multor linii dintr-un program folosim comanda

>D prima_linie,ultima_linie

aceasta realizând ștergerea tuturor liniilor dintre prima_linie și ultima_linie. De

exemplu, comanda

>D30,130

va produce ștergerea tuturor liniilor numerotate între 30 și 130.

La un moment dat, este posibil să se ajungă la necesitatea de a insera mai multe linii între două linii consecutive, între care nu mai există numere disponibile. De exemplu, avem liniile

```
130 LD A,(HL)
131 LD (HL),A
```

și dorim să introducem între ele și comanda XOR 255. Pentru aceasta nu este necesar să mutăm linia 131 la 132 și să scriem linia "131 XOR 255", ci putem folosi o facilități oferită de editor, și anume renumerotarea liniilor, cu comanda

>N prima_linie,distanța_între_linii

De exemplu, comanda

>N15,50

va renumerota toate liniile care urmează după linia 15 cu numere din 50 în 50. În cazul nostru, dacă dăm comanda

>N130,20

Linia care era numerotată cu 131 va deveni linia 150, linia următoare va deveni 170 și așa mai departe. Astfel, vom avea destul loc să inserăm comanda dorită.

Pentru mutarea unei linii, se folosește comanda

>M număr_vechi,număr_nou

De exemplu, cu

>M50,160

conținutul liniei 50 va fi mutat la linia 160.

O linie poate conține maxim o instrucțiune. Pentru a indica asamblorului unde să depună programul în memorie, este bine ca prima linie din program să fie o instrucțiune

ORG adresă

Aceasta nu este o instrucțiune cod mașină, ci este o instrucțiune care comunică asamblorului adresa unde să depună programul care urmează liniei respective. De exemplu, un program care începe cu

10 ORG 50000

va fi compilat la adresa 50000, deci va putea fi adresat cu

RANDOMIZE USR 50000

Altă instrucțiune a asamblorului pe care o vom folosi este instrucțiunea DB (Define Byte). Această instrucțiune comunică asamblorului să lase un octet liber, deoarece avem nevoie de el în program (pentru memorarea unor valori, a unor stări etc). Octetului respectiv i se dă un nume, și ne vom putea referi la el, pe parcursul programului, cu numele respectiv.

Exemplu:

10	ORG 50000	★ stabilește adresa de compilare la 50000
20	NM DEFB 0	★ octetului de la adresa curentă (50000) i se atribuie numele NM și este pus pe valoarea 0 (echivalent cu instrucțiunile BASIC LET NM=50000:POKE NM,0
30	LD A,(NM)	★ încarcă în acumulator conținutul octetului NM. Este echivalent cu a scrie LD A,(50000)
40	INC A	★ mărește conținutul acumulatorului cu 1
50	LD (NM),A	★ copiază conținutul acumulatorului în locația NM
...		

Atenție ! Un program care începe ca cel de mai sus va trebui să fie apelat cu

RANDOMIZE USR 50001

deoarece la adresa 50000 nu este o instrucțiune, ci o dată a programului, cu care procesorul nu știe ce să facă !

3.5 Primul program

Să încercăm să realizăm concret, pas cu pas, scrierea unui program. Să luăm de exemplu programele prezentate la pagina 39, în cadrul exemplelor de folosire a instrucțiunii LDIR. Acestea pot memora informația de pe ecran în memorie, respectiv o aduc înapoi, la momentul dorit. Pașii ce trebuie urmăriți pentru scrierea lor sunt:

★ se încarcă asamblorul, cu instrucțiunea

LOAD "GENS" CODE 26000

★ se lansează în execuție, cu comanda

RANDOMIZE USR 26000

- ★ se dă comanda pentru numerotare automată a liniilor:

>I

- ★ se comunică asamblorului adresa programului, punând în prima linie instrucțiunea

ORG 44000

- ★ se scriu liniile, apăsând ENTER (CR) la sfârșitul fiecăreia (ca în BASIC):

```
LD DE,45000
LD HL,16384
LD BC,6912
LDIR
```

- ★ ca orice program, pentru întoarcerea în BASIC trebuie să se facă cu o instrucțiune RET, care constituie o nouă linie:

```
RET
```

- ★ se iese din modul editare, apăsând tastele CAPS SHIFT și 1, adică EDIT
- ★ dacă dorim listarea programului, cerem aceasta

>L

Se va afișa:

```
10   ORG 44000
20   LD DE,45000
```

30 LD HL,16384
40 LD BC,6912
50 LDIR
60 RET

★ în această variantă programul este complet, deci se poate trece la asamblarea lui:

>A

Se apasă de două ori ENTER (CR), la întrebările "Tables:" și "Options" puse de asamblor. Programul va fi asamblat, și se revine la prompter.

★ se dă comanda

>B

pentru revenirea în BASIC.

★ se încarcă de pe casetă ecranul ce trebuie memorat sau se desenează ceva cu instrucțiunile PLOT, DRAW, CIRCLE

★ se memorează ecranul la adresa 45000, lansând programul în cod mașină, cu comanda

RANDOMIZE USR 44000

★ se relansează în execuție asamblorul, cu comanda

RANDOMIZE USR 26000

★ se șterg liniile programului anterior, cu comanda

>D10,60

★ se dă comanda I, pentru numerotare automată

★ se scrie al doilea program, ca și primul, punând la prima linie o adresă de ORG diferită de prima:

ORG 44050

★ se scriu liniile noului program:

LD DE,16384

LD HL,45000

LD BC,6912

LDIR

★ se pune ultima instrucțiune

RET

★ se iese din editare, cu

CAPS SHIFT și 1

★ se assemblează programul, cu comanda

>A

și de două ori ENTER.

★ se iese din asamblor, cu comanda

>B

★ se lansează programul în execuție, cu comanda

RANDOMIZE USR 44050

Observați că pe ecran se află acum chiar ecranul care a fost salvat în memorie, și el poate fi readus oricând, cu comanda de mai sus.

Dacă doriți să vedeți cum arată programul în tradus cod mașină, puteți afișa octeții de la adresa 44000, unde se află primul program:

```
FOR N=44000 TO 44049: PRINT PEEK(N);", ";;NEXT N
```

Pe ecran sunt afișate 50 de numere, cu virgulă între ele. Toate acestea formează programul nostru ?

Nu chiar. Programul ocupă mai puțin, el terminându-se cu o linie de RET. Aceasta înseamnă că ultimul octet al său este cel care conține numărul 201, care reprezintă corespondentul în cod mașină al instrucțiunii RET.

3.6 Rutine din memoria ROM

Parcurgând programele prezentate, veți observa că, în unele programe, apar instrucțiuni de CALL către adrese aflate în memoria ROM.

La aceste adrese se află rutine utile, scrise în cod mașină, pe care putem să le folosim, tot așa cum le folosește și interpretorul BASIC în pașii săi de execuție a programelor.

Cu alte cuvinte, dacă le transmitem date corecte, vom obține rezultatul dorit.

Rutinele apelate sunt:

3. INTRODUCERE ÎN COD MAȘINĂ

★ rutina de citire a tastaturii

Aceasta parcurge tastele, putând determina dacă este vreuna apăsată, și, dacă este, să o poată recunoaște.

Adresa ei este 654 (deci se poate apela cu CALL 654).

După execuție, se obțin următoarele rezultate:

★ în registrul E, numărul 255 dacă nu a fost detectată nici o tastă apăsată sau

★ un cod, în funcție de tasta care a fost găsită apăsată. Codurile corespunzătoare tuturor tastelor le puteți vedea în tabelul următor:

1 36	2 28	3 20	4 12	5 4	6 3	7 11	8 19	9 27	0 35
Q 37	W 29	E 21	R 13	T 5	Y 2	U 10	I 18	O 26	P 34
A 38	S 30	D 22	F 14	G 6	H 1	J 9	K 17	L 25	ENT 33
CS 39	Z 31	X 23	C 15	V 7	B 0	N 8	M 16	SS 24	BRK 32

★ rutina de producere a unui sunet

La apelarea rutinei, registrul HL trebuie să conțină frecvența sunetului, iar registrul DE, durata lui.

Adresa rutinei este 949. Nu se returnează nimic.

★ rutina de conversie coordonate ecran - adresă în memoria ecran

Aceasta realizează un calcul deosebit de util în toate instrucțiunile grafice: primind coordonatele X și Y ale unui punct pe ecran, returnează adresa lui în memoria ecran. Aceasta este un mare avantaj, deoarece, ecranul la calculatoarele de tip Spectrum având o structură deosebit de complicată a memoriei, programul acesta ar lua mult timp și spațiu în oricare din rutinele care ar avea nevoie de ce face el.

Adresa ei este 8874. La apelare, registrul B trebuie să conțină coordonata Y a punctului a cărui adresă se caută, iar registrul C, coordonata X.

La întoarcere, registrul HL conține adresa, în memoria ecran, la care se află acel punct. O adresă în memoria ecran conține informații despre 8 puncte, fiecăruia corespunzându-i un bit, care reprezintă starea punctului: aprins sau stins. Numărul bitului care reprezintă punctul căutat este întors în registrul A.

★ Rutina de aprindere/stingere a unui punct pe ecran

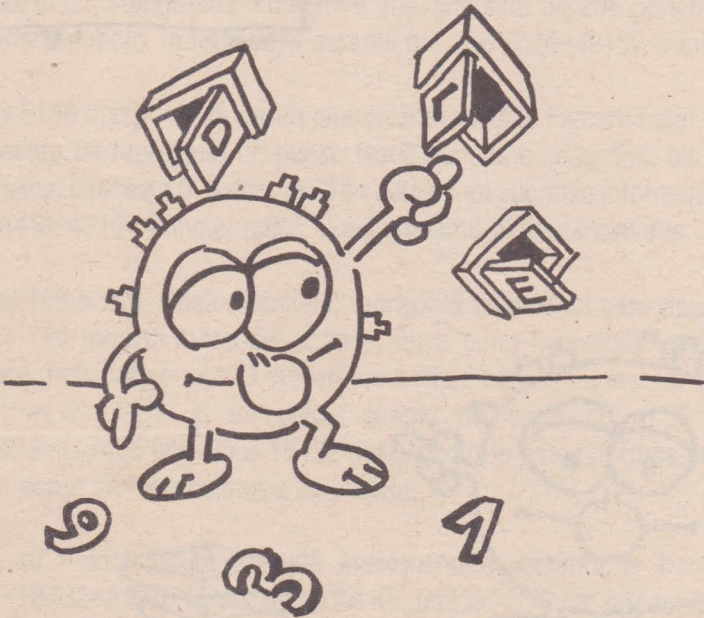
Din nou, o rutină deosebit de folositoare. Ea realizează ambele operații (aprindere sau stingere), în funcție de starea bitului de control din zona variabilelor sistem.

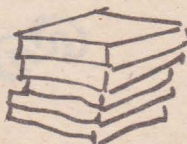
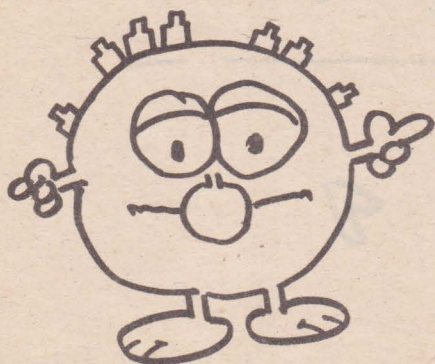
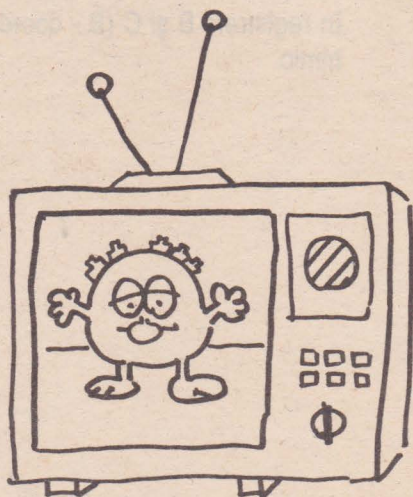
Acest octet, aflat la adresa 23697, conține informații cu privire la cum va apare următorul punct desenat pe ecran: în mod OVER, INVERSE, FLASH sau altfel. În programele din această carte, vom folosi numai comutatorul de OVER, care se află în bitul 0 al acestui octet. Dacă poziționăm bitul 0 (cu SET 0,(HL) unde HL are valoarea 23697), punctul care va apare va fi scris în modul OVER (va fi stins dacă e aprins și aprins dacă este stins).

Adresa rutinei este 8933.

3. INTRODUCERE ÎN COD MAȘINĂ

La apelarea rutinei, coordonatele punctului ce trebuie să fie aprins (sau stins) în registrele B și C (B - coordonata Y, C - coordonata X). Nu se returnează nimic.





Efecte de cortină

4.1 Prezentare

În memoria calculatorului, ecranul este reprezentat pe 6912 bytes, dintre care ultimii 768 reprezintă atributele (culorile). El are 24 linii și 32 coloane, numerotate din colțul din stânga sus. Lungimea unei linii este de 256 puncte iar lățimea ei este de 8 puncte. În total deci, ecranul are $256 \cdot 8 \cdot 24 = 49152$ puncte.

Primii 6144 octeți relevă starea punctelor ecranului. Fiecare octet cuprinde 8 biți, iar fiecare bit reprezintă un punct. Dacă punctul este aprins, bitul este 1; dacă este stins, bitul este 0. Astfel, în 6144 octeți sunt cuprinse informații despre starea a $6144 \cdot 8 = 49152$ puncte, adică exact valoarea obținută mai sus.

Ultimii 768 octeți, adică atributele, reprezintă culorile cu care apar punctele pe ecran. Din motivul memoriei limitate, nu a putut fi posibilă implementarea de culoare pe fiecare punct. O culoare necesită 3 biți pentru memorare (numărul de culori = $8 = 2^3$), și un calcul simplu ne arată că ar fi fost necesari $49152 \cdot 3 = 147456$ biți, adică 18432 octeți pentru a putea permite ca fiecare punct de pe ecran să aibă culoarea lui proprie.

Dacă ar fi fost făcută această implementare, ecranul ar fi ocupat în total $6144 + 18432 = 24576$ bytes, adică 24 Kb, față de 6.75 cât necesită în realitate. O asemenea dimensiune ar fi redus spațiul ce putea fi cedat programelor la 24 Kb față de 40.8 Kb cât are în implementarea actuală, ceea ce ar fi fost un dezavantaj enorm. La ce bun un ecran performant, dacă nu ai loc în memorie

pentru un program care să-l folosească ?

Având în vedere această posibilitate, s-a recurs la un artificiu: împărțirea ecranului în pătrate de 8 linii și 8 coloane (deci $8 * 8 = 64$ puncte) fiecare. Într-un astfel de pătrat se reprezintă un caracter, și tot la un astfel de pătrat se referă și **atributul de culoare**.

Memoria necesară fiind astfel micșorată foarte mult, s-a putut permite ca un atribut să fie reprezentat nu pe 3 biți, ci pe un octet întreg, deci pe 8 biți. Aceasta oferă posibilitatea folosirii a două culori, deci în cadrul aceleiași pătrat să apară și culoare de cerneală (INK - culoarea cu care apar punctele aprinse pe ecran) și culoare de fond (PAPER - culoarea cu care apar punctele stinse).

Aceste două culori cereau câte 3 biți fiecare, deci 6 biți. Pe cei 2 biți rămași din octet s-a hotărât implementarea atributelor de BRIGHT (strălucire), respectiv FLASH (clipire). Dacă bitul 7, care reprezintă FLASH, este poziționat pe 1, atunci pătratul căruia îi corespunde atributul respectiv va clipi intermitent pe ecran.

Memoria necesară atributelor a devenit astfel de 768 bytes (âncepând de la adresa 22528, la care se termină memoria necesară punctelor), câte un byte pentru fiecare pătrat de pe ecran. Pătratele sunt numerotate de la dreapta la stânga și de sus în jos.

De exemplu, pătratul din linia 0, coloana 0 îi corespunde atributul de la adresa 22528, celui de la coloana 1, atributul de la adresa 22529, și așa mai departe. Dacă adăugăm la adresa unui atribut valoarea 32 (lungimea unei linii) obținem adresa atributului de sub el.

Pentru a vă convinge, încercați următorul program:

```
10 FOR N=0 TO 21
20 POKE 22528+32*N,0
30 PRINT AT N,3;"ADRESA ";32*N+22528
```


40 NEXT N

Programul realizează acoperirea cu negru a pătratelor de pe prima coloană. Veți vedea că, adăugând 32 la fiecare adresă, vom obține adresa pătratului de dedesubt (dovadă că pătratele marcate sunt unele sub altele).

Observație: schimbarea atributului unui pătrat nu duce la pierderea conținutului acestuia, nici chiar dacă culorile cernelii și ale fondului sunt aceleași ! În memoria punctelor se păstrează informația cu privire la starea lor (aprinse sau stinse), deci la schimbarea atributului cu unul care are aceste culori diferite, conținutul pătratului va ieși din nou la iveală !

Exemplu: încercați programul:

```

10 PRINT AT 0,0;"MESAJ"
15 REM S-A SCRIS MESAJUL, ACUM SE ASTEAPTA
    APASAREA UNEI TASTE PENTRU ASCUNDEREA LUI
20 PAUSE 0
25 REM S-A APASAT O TASTA, ACUM SE VOR PUNE CELE
    5 ATRIBUTE ALE MESAJULUI PE 0 ("INK 0, PAPER
    0")
30 FOR N=22528 TO 22532
40 POKE N,0
50 NEXT N
60 PRINT AT 2,0 "APASATI O TASTA PENTRU DEZVELIREA
    MESAJULUI"
70 PAUSE 0
80 REM PENTRU RELEVAREA MESAJULUI, SE VOR PUNE
    ATRIBUTELE ACESTUIA PE 7 (INK 0, PAPER 7)
90 FOR N=22528 TO 22532
100 POKE N,7
110 NEXT N
120 REM NEGRU PE ALB

```

Se poate realiza un efect frumos, dacă în timp ce mesajul nu se vede, noi vom

schimba conținutul lui. Pentru aceasta este suficient să adăugăm linia

```
65 PRINT AT 0,0; INK 0; PAPER 0; "BAU !"
```

și atunci, la "dezvelire" va apare nu primul, ci al doilea mesaj.

Efectul, scris în BASIC, are un dezavantaj: viteza mică. Dacă dorim, de exemplu, să ascundem conținutul întregului ecran și să-l afișăm apoi schimbat, umplerea întregului spațiu cu atributul 0 va lua ceva timp.

În cod mașină însă, totul se petrece atât de repede, încât ne putem permite și să facem acoperirea mai artistică, în diverse sensuri (de la dreapta la stânga, de sus în jos, în diagonală) și chiar să trecem repetat toate atributele ecranului prin toate valorile intermediare, până ajung la valoarea dorită !

Programele sunt astfel concepute încât să poată fi modificate, permițând acoperirea numai a unor părți din ecran, precum și schimbarea atributului cu care se face aceasta (printr-un POKE). Pentru folosirea lor, vom proceda după următorii pași:

- ★ se afișează primul mesaj
- ★ în momentul dorit, se apelează programul în cod mașină, cu atributul de acoperire 0, pentru ascundere
- ★ se afișează al doilea mesaj, cu INK 0, PAPER 0
- ★ la momentul "dezvelirii" se apelează iar subrutina în cod mașină, însă de data aceasta cu atributul 7, pentru descoperire.

Subrutina cu care se face acoperirea nu trebuie să fie neapărat aceeași cu cea cu care se face dezvelirea. Se poate, de exemplu, acoperi de la dreapta la stânga și dezveli în diagonală !

4.2 Acoperirea ecranului de sus în jos

Următorul program realizează acoperirea unei ferestre de sus în jos, prin acoperirea fiecărei linii în parte. Pentru aceasta, pornește din colțul din stânga sus al ferestrei, acoperă un număr de bytes dați de lungimea ferestrei (deci o linie), apoi trece la linia următoare (adunând 32 la adresa de început a liniei curente) și repetă până la acoperirea completă.

1	ORG 49000	★adresa la care va fi asamblat.
10	LD HL, 22528	★adresa atributului din colțul stânga sus al ferestrei (22528+linia de start*32).
20	LD B, 24	★lățimea ferestrei (în linii).
30	L0 PUSH HL	★înmagazinează datele inițiale în stivă.
40	PUSH BC	
50	LD B, 32	★lungimea ferestrei (în coloane).
60	L1 LD (HL), 0	★poziționează atributul curent pe ' INK 0 , PAPER 0 '.
70	INC HL	★trece la atributul următor.
80	DJNZ L1	★repetă pentru toată linia curentă.
90	LD DE, 32	★adună 32 (aici, lungimea ferestrei) la adresa atributului pentru a obține adresa atributului de la începutul liniei următoare.
100	POP BC	
110	POP HL	
120	ADC HL, DE	
130	HALT	★pauză; dacă se înlocuiește cu NOP, procesul decurge mai rapid.
140	DJNZ L0	★repetă pentru toate liniile.
150	RET	★se reîntoarce în BASIC.

Program BASIC:

```

10 LET ADR1=49000:LET X=25:LET
   S=2592:RESTORE 30
20 LET ADR=ADR1:GOSUB 9997
30 DATA 33,0,88,6,24,229,197,6,32,54,255,35,
   16,251,17,32,0,193,225,237,90,118,16,237,201

```

Se pot schimba următoarele:

- ★ adresa colțului din stânga sus al ferestrei, la locațiile ADR1+1 și ADR1+2.
- ★ lățimea ferestrei, la locația ADR1+4.
- ★ lungimea ferestrei, la adresa ADR1+8.
- ★ atributul cu care se acoperă, în locația ADR1+10.
- ★ viteza acoperirii, la adresa ADR1+21 (118 pentru lent, 0 pentru rapid).

4.3 Acoperirea ecranului de jos în sus

Programul este asemănător cu cel de mai sus, cu deosebirea că aici acoperirea se face de la ultima linie la prima, deci de la adrese mari la adrese mici.

```

1      ORG 49025
10     LD HL,23265      ★adresa atributului din colțul stânga jos al
                        ferestrei.
20     LD B,24         ★lățimea ferestrei (în linii).

```

30	L0	PUSH HL	★înmagazinează datele inițiale în stivă.
40		PUSH BC	
50		LD B, 32	★lungimea ferestrei (în coloane).
60	L1	LD (HL), 0	★poziționează atributul curent pe ' INK 0 , PAPER 0 '.
70		DEC HL	★trece la atributul următor.
80		DJNZ L1	★repetă pentru toată linia curentă.
90		LD DE, 32	★scade 32 la adresa atributului pentru a obține adresa atributului de la începutul liniei de deasupra liniei curente.
100		POP HL	
110		SBC HL, DE	
120		POP BC	
130		HALT	★pauză (poate fi înlocuit cu NOP).
130		DJNZ L0	★repetă pentru toate liniile.
140		RET	★se reîntoarce în BASIC.

Program BASIC:

```

40 LET ADR2=49025:LET X=25:LET
S=2817:RESTORE 60
50 LET ADR=ADR2:GOSUB 9997
60 DATA 33,223,90,6,24,229,197,6,32,54,255,43,
16,251,17,32,0,193,225,237,82,118,16,237,201

```

Se pot schimba următoarele:

- ★ adresa colțului din stânga jos al ferestrei, la locațiile ADR2+1 și ADR2+2.
- ★ lățimea ferestrei, la locația ADR2+4.
- ★ lungimea ferestrei, la adresa ADR2+8.
- ★ atributul cu care se acoperă, în locația ADR2+10.

★ viteza acoperirii, la adresa ADR2+21 (118 pentru lent, 0 pentru rapid).

4.4 Acoperirea de la stânga la dreapta

Programul preia fiecare pătrat dintr-o coloană, îi schimbă atributul și trece la pătratul de dedesubt acestuia (prin adăugarea valorii 32 la adresa primului), până la terminarea unei coloane, apoi repetă pentru toate coloanele (trecerea la coloana următoare se face măbind cu 1 adresa de început a coloanei curente)

1	ORG 49050	
10	LD HL, 22528	★ încarcă coordonatele colțului stânga sus al ferestrei în
		registrul HL.
20	LD B, 32	★ lungimea ferestrei.
30 L3	PUSH BC	
40	PUSH HL	
50	LD B, 24	★ lățimea ferestrei.
60	LD DE, 32	
70 L4	LD (HL), 0	★ atributul cu care se acoperă.
80	ADC HL, DE	★ trece la poziția de dedesubt.
90	DJNZ L4	★ repetă până s-a acoperit o coloană.
100	POP HL	★ reface adresa de început a coloanei.
110	POP BC	
120	INC HL	★ trece la coloana următoare.
130	HALT	★ pentru micșorarea vitezei.
140	DJNZ L3	★ repetă pentru toate coloanele.
150	RET	

Program BASIC:

```
70 LET ADR3=49050:LET X=25:LET
S=2336:RESTORE 68
80 LET ADR=ADR3:GOSUB 9997
```

90 DATA 33,0,88,6,32,197,229,6,24,17,32,0,54,0,
237,90,16,250,225,193,35,118,16,237,201

Se pot schimba următoarele:

- ★ adresa colțului din stânga jos al ferestrei, la locațiile ADR3+1 și ADR3+2.
- ★ lățimea ferestrei, la locația ADR3+4.
- ★ lungimea ferestrei, la adresa ADR3+8.
- ★ atributul cu care se acoperă, în locația ADR3+13.
- ★ viteza acoperirii, la adresa ADR3+21 (118 pentru lent, 0 pentru rapid).

4.5 Stingere cu flash

Programul realizează aducerea atributelor ecranului la o aceeași valoare, trecându-le prin toate stările intermediare. De exemplu, dacă avem un atribut cu valoarea 7 (PAPER 0, INK 7) și valoarea la care dorim să-l aducem este 0 (INK 0, PAPER 0), atunci el va fi trecut prin toate valorile din intervalul 0-7, respectiv 6,5,4,3,2,1,0.

Pe ecran, aceasta se va traduce prin schimbarea permanentă a culorii, din alb devenind galben, apoi albastru, verde, magenta, roșu, albastru închis și în sfârșit negru.

Algoritmul este relativ simplu: se traversează ecranul de un număr de ori, micșorând de fiecare dată valoarea atributelor care nu coincid deja cu cele cerute. Numărul maxim de valori ale unui atribut sunt 256, deci în mod sigur, după 255 de traversări de acest gen, atributele ecranului vor ajunge toate în starea cerută.

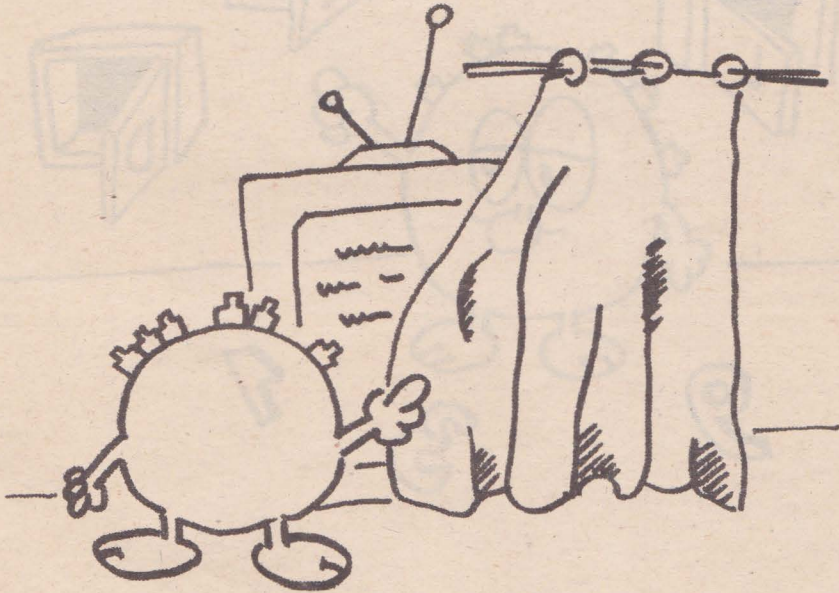
1	ORG 49075	
10	LD DE, 255	★ numărul de repetări a rutinei de micșorare a culorii estimate ca necesare pentru a se "stinge" tot ecranul. Dacă pe ecran nu sunt atribute cu FLASH sau BRIGHT, vor fi suficiente 64 de repetări, dacă sunt atribute cu BRIGHT și nu sunt cu FLASH vor ajunge 128 de repetări. -
20	L2 LD HL, 22528	★ adresa atributelor.
30	LD BC, 768	★ numărul de atribute de pe ecran.
40	L1 LD A, 0	★ în A se încarcă starea finală în care se dorește să ajungă atributele.
50	CP (HL)	★ verifică dacă atributul curent a ajuns în starea finală.
60	JR NZ, DEC	★ dacă nu, atunci sare la subrutina care îl modifică.
70	FIN1 DEC BC	★ BC reprezintă numărul de atribute rămase de verificat; este micșorat cu 1.
80	INC HL	★ se trece la atributul următor.
90	LD A, B	
100	OR C	
110	JR Z, FIN2	★ dacă au fost verificate toate caracterele din ecran, atunci sare mai jos.
120	JR L1	★ dacă nu, atunci reia pentru caracterul următor.
130	FIN2 DEC DE	★ verifică dacă s-a terminat numărul de repetări cerute.
140	LD A, D	
150	OR E	
160	RET Z	★ dacă da, atunci se întoarce în BASIC.
170	JR L2	★ dacă nu, atunci reia pentru următoarea repetare.
180	DEC DEC (HL)	★ aici se ajunge pentru schimbarea atributului curent. În varianta prezentată, facem aceasta prin micșorare, însă se poate face și prin mărire, cu INC (HL).
190	JR FIN1	★ după ce modifică atributul, sare la rutina de verificare a sfârșitului de ecran.

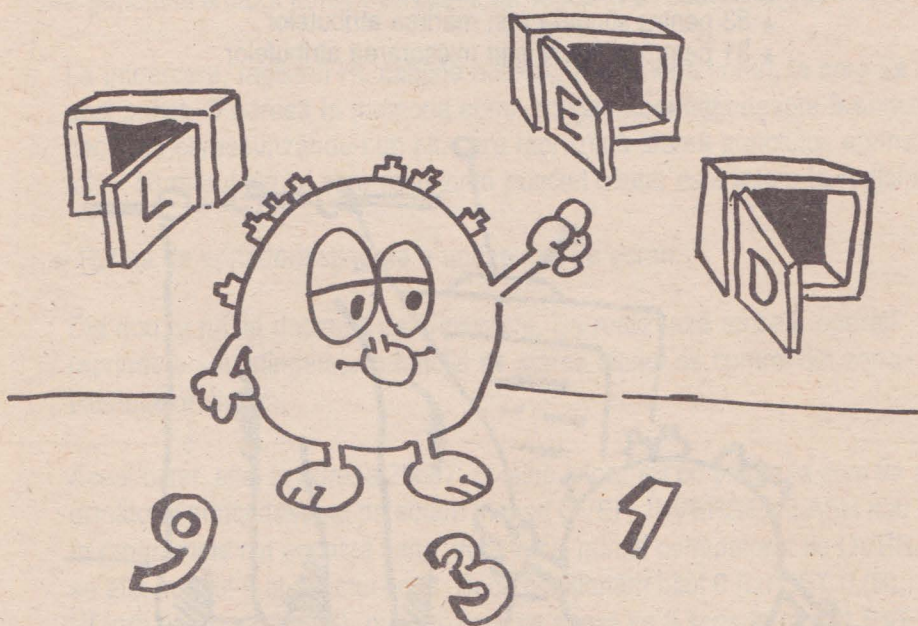
Program BASIC:

```
100 LET ADR4=49075:LET X=31:LET  
S=2446:RESTORE 120  
110 LET ADR=ADR4:GOSUB 9997  
120 DATA 17,255,0,33,0,88,1,0,3,62,0,190,32,14,  
11,35,120,177,40,2,24,243,27,122,179,200,24,  
231,53,24,239
```

Modificări posibile:

- ★ la adresa ADR4+10, valoarea atributului la care trebuie să fie adus ecranul.
- ★ la adresa ADR4+28 se modifică astfel:
 - ★ 53 pentru stingere prin mărirea atributelor
 - ★ 31 pentru stingere prin micșorarea atributelor.





Efecte de așteptare

5.1 Prezentare

Efectele care urmează sunt concepute pentru situații deosebite, când calculatorul trebuie să stea și să aștepte o decizie a utilizatorului, ecranul "încremenind" în așteptarea apăsării unei taste. Programele care urmează oferă o alternativă mai plăcută, și anume prezentarea unor efecte mai atrăgătoare, vizuale și/sau sonore, până când este apăsată o tastă.

5.2 Flash colorat

Programul realizează o variantă a instrucțiunii FLASH din BASIC, ajutând la scoaterea în evidență a unor porțiuni dintr-un ecran sau text, prin trecerea succesivă a acestora prin toate culorile oferite de placa grafică a calculatorului.

```
1   ORG 49150
10  LD HL,22528
20  LD BC,768

30  L10 LD A,(HL)
40  AND 64
```

- ★adresa atributelor.
- ★lungimea zonei de memorie care conține atributele.
- ★încarcă atributul curent în acumulator.
- ★verifică dacă este scris cu BRIGHT.

50	JR Z,DC	★dacă nu, atunci sare la eticheta DC, unde verifică dacă s-a ajuns la ultimul atribut din ecran.
70	INC (HL)	★dacă da, atunci mărește culoarea cernelii cu 1.
80	LD A, (HL)	
90	AND 7	
100	CP 7	★dacă cerneala a ajuns la 7 (alb) atunci revine la cerneala 1 (albastră).
110	JR Z,Z1	
120	DC DEC BC	★subrutina verifică dacă au fost analizate toate atributele ecranului. Dacă nu au fost terminate, repetă programul pentru atributul următor.
130	LD A,B	
140	OR C	
150	RET Z	
160	INC HL	
170	JR L10	
180	Z1 LD (HL),64	★subrutina realizează revenirea atributului curent la culoarea albastră.
190	JR DC	

Program BASIC:

```

130 LET ADR5=49150:LET X=30:LET
S=2507:RESTORE 150
140 LET ADR=ADR5:GOSUB 9997
150 DATA 30,0,88,1,0,3,126,230,64,40,8,52,126,
230,7,254,7,40,7,11,120,177,200,35,24,236,54,65,
24,245
    
```

Pe calculatoarele HC, instrucțiunea BRIGHT nu are nici un efect. în mod normal,

ar trebui să determine afișarea culorilor mai puternic pe ecran (BRIGHT înseamnă strălucire). De aceea, afișarea în acest mod nu va determina probleme de ordin estetic.

Practic, folosirea ei duce la poziționarea penultimului bit din atributul caracterului ce trebuie afișat (un atribut este reprezentat pe un octet: primii 3 biți reprezintă culoarea cernelii 0-7, următorii 3 biți culoarea fondului, un bit este poziționat pentru BRIGHT iar altul pentru FLASH).

Programul detectează atributele care au acest bit poziționat și modifică culoarea cernelii, trecând la următoarea (dacă este albastră trece la roșu, apoi la magenta și așa mai departe până la alb; dacă este alb o readuce la albastră). Repetând apelarea programului într-un ciclu vom obține un efect de "clipire" mult mai frumos decât cel oferit din fabricație.

Exemplu de folosire:

```
100 PRINT BRIGHT 1;" TREBUIE "; BRIGHT 0; "SA
    STIU INFORMATICA"
110 RANDOMIZE USR ADR5 (adresa programului)
120 IF INKEY$<>" " THEN STOP
130 GOTO 110
```

5.3 Efecte pe BORDER

În timpul așteptării, un efect deosebit în poate da animarea părții de ecran ce înconjoară zona activă, parte numită BORDER. Programul de mai jos oferă mai multe variante, vizuale și sonore, de activare a acestei zone.

Pentru a scoate un semnal în această parte, vom folosi o instrucțiune care este

folosită și în subrutina de LOAD din memoria ROM a calculatorului, și anume
OUT (#FE) , A

Practic, tot ce vom face va fi să transmitem cu o frecvență variabilă semnale pe
BORDER, prin combinarea lor realizându-se efectul.

1	ORG 49180	
10	LD BC,3000	★stabilește de câte ori se va repeta grupul de semnale (durata efectului).
20	L1 PUSH BC	★păstrează durata în stivă.
30	LD B,250	★stabilește numărul de semnale ce fac parte dintr-un grup.
40	L2 INC A	★la fiecare pas, modifică semnalul (instrucțiunea poate fi înlocuită cu oricare alta care prelucrează registrul A).
50	OUT (#FE) , A	★transmite semnalul pe BORDER.
60	DJNZ L2	★repetă de B ori.
70	POP BC	★reface registrul BC, care conține numărul de grupuri de semnale.
80	DEC BC	★micșorează BC cu 1.
90	LD A,B	
100	OR C	★testează dacă BC a ajuns la 0.
110	JR NZ,L1	★dacă nu, atunci se întoarce la L1.
120	RET	★altfel iese în BASIC.

Program BASIC:

```
160 LET ADR6=49180:LET X=18:LET
S=2417:RESTORE 180
170 LET ADR=ADR6:GOSUB 9997
```

180 DATA 1,184,11,197,6,250,60,211,254,16,251,
193,11,120,177,32,242,201

Modificări posibile:

★ la adresele ADR6+1 și ADR6+2 se află durata semnalului (aici $3000=11*256+184$ ⇔ reprezentat pe 2 bytes, 184 și 11).

★ la locația ADR6+5 este numărul de semnale dintr-un grup.

Variantă: pentru durate mari, programul următor realizează întoarcerea în BASIC la apăsarea unei taste, nu numai la terminarea semnalelor.

1	ORG 49200	
10	LD BC,3000	★stabilește de câte ori se va repeta grupul de semnale (durata efectului).
20	L1 PUSH BC	★păstrează durata în stivă.
30	LD B,250	★stabilește numărul de semnale ce fac parte dintr-un grup.
40	L2 INC A	★la fiecare pas, modifică semnalul (instrucțiunea poate fi înlocuită cu oricare alta care prelucrează registrul A).
50	OUT (#FE),A	★transmite semnalul pe BORDER.
60	DJNZ L2	★repetă de B ori.
70	CALL 654	★cheamă subrutina de citire a tasturii.
80	LD A,E	
90	CP 255	★verifică dacă a fost apăsată vreo tastă.
100	JR NZ,RT	★dacă da, atunci sare la rutina de eliberare a stivei și întoarcere în BASIC.
110	POP BC	★reface registrul BC, care conține numărul de grupuri de semnale.
120	DEC BC	★micșorează BC cu 1.

130	LD A, B	
140	OR C	★testează dacă BC a ajuns la 0.
150	JR NZ, L1	★dacă nu, atunci se întoarce la L1.
160	RET	★altfel iese în BASIC.
170	RT POP BC	★la această linie se ajunge dacă programul a fost întrerupt forțat, prin apăsarea unei taste. înainte de a ieși în BASIC, trebuie eliberată stiva.
180	RET	

Program BASIC:

```

190 LET ADR7=49200:LET X=28:RESTORE 210
200 LET ADR=ADR7:GOSUB 9980
210 DATA 1,184,11,197,6,250,60,211,254,16,251,
205,142,2,123,254,255,32,7,193,11,120,177,32,
234,201,193,201
    
```

Modificări:

- ★ la adresele ADR7+1 și ADR7+2 se află durata semnalului (aici $3000=11*256+184 \leftrightarrow$ reprezentat pe 2 bytes, 184 și 11).
- ★ la locația ADR7+5 este numărul de semnale dintr-un grup.

Lucrul cu imagini

6.1 *Prezentare*

Manipularea imaginilor este un subiect destul de tratat în ultima vreme, și orice nou limbaj cuprinde obligatoriu și elemente de lucru cu ferestre. Pe vremea când a fost conceput calculatorul SPECTRUM însă, cea mai bună configurație pentru lansarea pe piață a primului computer personal a părut cea oferită de interpretorul BASIC standard, fără prea multe facilități hardware sau software.

Într-adevăr, calculatoarele de acest tip s-au constituit într-un pas important în dezvoltarea informaticii, milioane de programatori luând prima oară cunoștință cu lumea informaticii pe tastaturile lor. Treptat însă, cerințele s-au mărit, iar SPECTRUM nu le-a mai putut face față. Omul avea nevoie de mașini cu putere de calcul și cu viteză mult mai mare, capabile să înțeleagă limbaje tot mai complexe, tinzând spre inteligența artificială.

Și și le-a construit. Limbaje ca C sub compilatorul BORLAND, medii de utilizare ca WINDOWS, care sunt lucruri imposibile pentru Z80, au devenit realitate pe mașinile din familia INTEL și MOTOROLA. Aceste aplicații, la care numai minimul de fișiere necesar pentru funcționare ocupă o memorie de 200 de ori mai mare decât a calculatorului SPECTRUM, știu să înțeleagă concepte complexe de inteligență artificială, să creeze și să utilizeze sub-limbaje, să definească obiecte.

Între aceste obiecte, *imaginea* ocupă un loc aparte, ea fiind elementul de bază al multor aplicații, dintre care cea mai cunoscută este WINDOWS. În orice compilator, de la TURBO BASIC la PROLOG și BORLAND C există seturi întregi de instrucțiuni ce pot fi folosite la manipularea ferestrelor, iar imaginea este numai unul dintre tipurile ferestrelor.

Nu vom încerca imposibilul. Nu avem pretenția că vom reuși să aplicăm pe Z80 concepte caracteristice lui 80386. Însă, chiar dacă memoria ecran a SPECTRUM-ului este de 150 de ori mai mică decât a unei plăci grafice SVGA, credem că o putem manevra astfel încât să putem obține și noi efecte de calitate, pentru nivelul de hardware la care lucrăm.

În mod sigur însă, cei care ajung să stăpânească bine un computer cu resursele minime ale SPECTRUM-ului, se vor adapta imediat în lumea PC-urilor. Codul mașină este mult îmbunătățit, și totuși, acolo, aplicațiile ating dimensiuni astronomice, și aceasta pentru că toată lumea programează prin intermediul compilatoarelor. Asta costă timp și spațiu.

Pentru cine s-a născut însă cu programarea "în sânge" și a fost educat în spiritul economiei de memorie de pe un HC sau CIP, programarea, pe PC, a unei aplicații profesionale ce să se încadreze în limite performante de viteză și memorie nu este un lucru foarte dificil.

6.2 Compactarea ecranului

La calculatoarele de tip Sinclair Spectrum ecranului îi corespunde zona de memorie de la adresa 16384 până la 23296, deci 6912 bytes. Dintre aceștia 6144 reprezintă punctele de pe ecran (1 bit pentru fiecare punct - un octet pentru 8 puncte) și 768 reprezintă atributele (prin atribut se înțelege ansamblul de culori - cerneală, fond, strălucire, flash - cu care apare pe ecran un careu de 8x8 puncte).

Un ecran poate fi înmagazinat ca orice altă parte a memoriei, pe casetă sau în altă zonă de adrese, dar peste tot ocupă 6912 bytes. Această dimensiune mare face dificilă manevrarea a mai multe imagini în cadrul unui program mai complex.

Dacă înmagazinăm pe casetă cu `SAVE "nume" SCREEN$` atunci derularea programului va fi foarte anevoioasă, la fiecare schimbare de imagine trebuind să manevrăm caseta, să așteptăm încărcarea, etc. Bineînțeles, dacă dispunem de o unitate de disc magnetic aceste neajunsuri se reduc mult.

Dacă însă nu avem la dispoziție decât memoria calculatorului, un calcul simplu ne duce la concluzia că, dacă putem reduce programul nostru la 7 Kbytes mai rămâne memorie doar pentru 6 ecrane.

Cu programul care urmează, dimensiunea unui ecran poate fi redusă foarte mult, putând fi înmagazinat pe o bucată mult mai mică de memorie (poate ajunge chiar la 100-200 bytes). Înmagazinarea sau readucerea lui pe ecran nu prezintă probleme de timp, datorită algoritmului. Cu cât un ecran este mai liber, cu atât va ocupa mai puțin spațiu în memorie.

Programul realizează compactarea spațiilor (octeți 0). De exemplu, dacă întâlnește o secvență de 200 de octeți liberi, îi înlocuiește cu numai 2 octeți: 0 și 200. Desigur, nu este cel mai performant program, existând și algoritmi mult mai eficienți, dar este cel mai simplu de programat și înțeles în limbaj de

asamblare.

De asemenea, ocupă foarte puțin spațiu în memorie.

Programul în limbaj de asamblare este următorul:

1	ORG 49300	
10	LD C, 0	★în registrul C se va afla permanent numărul de bytes de 0 consecutivi găsiți.
20	LD DE, 16384	★adresa ecranului.
30	LD HL, 50000	★adresa la care este memorat ecranul compactat (presupunem că acolo nu se află nici un program care să poată fi alterat).
40 LIT	LD A, (DE)	
50	CP 0	★dacă octetul curent este 0, atunci sare mai jos.
60	JR Z, ZERO	
70	LD (HL), A	★dacă nu, atunci îl transferă așa cum este.
80	INC HL	
90	INC DE	★mărește adresa sursei și a destinației.
100	JR ANL	★sare mai jos (verifică dacă a fost memorat tot ecranul).
110 ZERO	INC C	★aici se ajunge dacă au fost găsiți mai mulți octeți de 0 consecutivi. Mai întâi mărește C, care reprezintă numărul lor.
120	LD A, 255	★il compară cu 255 (numărul maxim care poate fi reprezentat pe un octet).
130	CP C	
140	JR Z, DCC	★dacă au fost găsiți 255 octeți de 0, sare la eticheta DCC (ânmagazinează 0,255 și readuce C la 0)
150 RET	INC DE	★aici se testează dacă, după un șir de bytes consecutivi de 0, s-a găsit unul care este

diferit de 0.

160	LD A, (DE)	
170	CP 0	
180	JR NZ, PUN	★dacă da, atunci înmagazinează șirul care a fost acumulat, ca să trateze octetul curent.
190	JR LIT	★dacă nu, îl adaugă la șir.
200	PUN LD (HL), 0	★aici se realizează memorarea unui șir ca 2 octeți: 0 și lungimea șirului.
210	INC HL	
220	LD (HL), C	
230	LD C, 0	★se readuce contorul la 0.
240	INC HL	
250	ANL LD A, 88	
260	CP D	
270	JR NZ, LIT	
280	RET	
290	DCC LD (HL), 0	★aici se memorează un șir de 255 de bytes, dacă C a ajuns la această valoare.
300	INC HL	
310	LD (HL), 255	
320	INC HL	
330	LD C, 0	★se readuce contorul la 0.
340	JR RET	

Program BASIC:

```

230 LET ADR8=49300:LET X=99:
LET S=3846:RESTORE 83
240 LET ADR=ADR8:GOSUB 9997
250 DATA 14,0,17,0,64,33,80,195,26,254,0,40,5,
119,35,19,24,21,12,62,255,185,40,21,1926,254,0,
32,2,24,232,54,0,35,113,14,0,35,62,88,186,32,
220,201,54,0,35,54,255,35,14,0,24,225

```

6.3 Refacerea unui ecran compactat

De fapt, această subrutină trebuie inclusă într-un program care folosește ecrane compactate, deoarece ea le restaurează din forma compactată în imagine pe ecran.

Spre deosebire de primul program, acesta este mult mai scurt, singurul test făcut fiind cel al octetului "0". Dacă acest octet este întâlnit, programul citește octetul următor și pune pe ecran atâți bytes de "0" cât reprezintă acesta.

Programul în limbaj de asamblare este următorul:

1	ORG 49400	
10	LD HL, 16384	
20	LD DE, 50000	★adresa la care se află ecranul compactat.
30	LIT LA A, (DE)	
40	CP 0	★compară octetul curent cu 0.
50	JR Z, ZERO	★dacă este 0, atunci sare mai jos.
60	LD (HL), A	★dacă nu, atunci îl transferă pe ecran așa cum este.
70	INC HL	
80	INC DE	★trece la adresele următoare.
90	JR ANL	★continuă de la eticheta ANL.
100	ZERO INC DE	★aici se ajunge dacă a fost întâlnit octetul 0. În acest caz, octetul următor reprezintă numărul de repetări al acestuia pe ecran.
110	LD A, (DE)	
120	LD B, A	★încarcă în B numărul de octeți de 0 care trebuie puși pe ecran.
130	L1 LD (HL), 0	★pune un octet de 0 la adresa curentă pe ecran,
140	INC HL	★trece la adresa următoare
150	DJNZ L1	★și repetă de B ori.

```
160      INC DE
170 ANL LD A,88
```

★ aici se ajunge după fiecare octet analizat, pentru că aici se face testul de sfârșit de ecran. Dacă ecranul a fost umplut, atunci iese din program.

```
180      CP H
190      JR Z,LIT
```

★ dacă nu a fost încă umplut, atunci continuă cu următorul byte.

```
200      RET
```

Program BASIC:

```
260 LET ADR9=49400:LET X=31:
LET S=2214:RESTORE 280
270 LET ADR=ADR9:GOSUB 9997
280 DATA 33,0,64,17,80,195,26,254,0,40,5,119,35,
19,24,9,19,26,71,54,0,35,16,251,19,62,88,188,32,
232,201
```

6.4 Memorarea unei ferestre

Programul permite reținerea în memoria calculatorului a imaginii existente într-o porțiune rectangulară de ecran, pentru a putea fi prelucrată și adusă eventual înapoi.

Algoritmul este următorul:

- ★ calculează adresa unde începe, în memoria ecran, linia curentă.
- ★ transferă în memorie un număr de octeți dat de lungimea ferestrei.
- ★ trece la linia următoare.
- ★ repetă de la primul pas pentru un număr de linii dat de lățimea ferestrei.

Parcurgerea ferestrei începe din colțul din stânga sus. În primul pas, calculează

adresa acestui colț, plecând de la coordonatele lui în puncte (X și Y sunt memorate în registrul C, respectiv B) și folosind o subrutină deja existentă în memoria ROM.

Această subrutină, aflată la adresa 8933, este apelată cu o instrucțiune de CALL și va realiza calculul adresei octetului care conține, în memoria ecran, punctul de coordonatele date în B și C. La reîntoarcerea din subrutină, această adresă este memorată în registrul HL.

Având astfel adresa de început a unei linii, putem parcurge ușor toată linia, deoarece octeții din care este formată au adrese consecutive. De exemplu, octetul numărul 3 din linie va avea adresa egală cu adresa începutului de linie + 3 (primul octet din linie este considerat octetul numărul 0). Pentru a parcurge o linie, este necesar să parcurgem un număr de octeți dat de lungimea ei.

Trecerea la linia următoare nu va fi grea dacă mai avem încă memorate coordonatele, în puncte, ale începutului liniei anterioare. Având aceste coordonate, nu rămâne decât să micșorăm coordonata Y (registrul B) cu 1, pentru a comunica trecerea la linia aflată dedesubtul celei pe care tocmai am terminat-o de parcurs.

Repetând acești pași de un număr de ori egal cu numărul de linii al ferestrei, vom termina de parcurs toată fereastra.

În programul care urmează, fereastra este parcursă ca mai sus, realizându-se în același timp și transferul octeților care o formează la adrese consecutive de memorie.

Programul în limbaj de asamblare este:

5

ORG 49440

★adresa la care compilăm programul în memorie (o notăm ADR10).

10	LD IX, 51000	★adresa la care este salvată fereastra în memorie.
20	LD BC, 0	★colțul stânga jos al ferestrei (punctul de coordonate 0,0 - B=Y=0,C=X=0).
30	LD DE, #0A0A	★lungimea ferestrei (în caractere)=10 și înălțimea ferestrei (în pixeli)=10.
50	L1 PUSH BC	
60	PUSH DE	
70	CALL #22AA	★calculează adresa la care începe fereastra și o depune în HL.
90	POP DE	
100	LD B, D	★încarcă în B lungimea unei linii.
110	L2 LD A, (HL)	
120	LD (IX+00), A	★salvează byte-ul curent din linie la adresa IX.
130	INC HL	★trece la următorul byte.
140	INC IX	★incrementează adresa la care se salvează.
150	DJNZ L2	★repetă pentru toată linia curentă.
160	POP BC	
170	INC B	★trece la următoarea linie.
180	DEC E	★decrementează contorul de linii.
190	LD A, E	
200	CP 0	
210	RET Z	★dacă a ajuns la 0, atunci se iese din program.
220	JR L1	★altfel repetă pentru linia următoare.

Program BASIC:

```
290 LET ADR10=49440:LET X=35:.
```

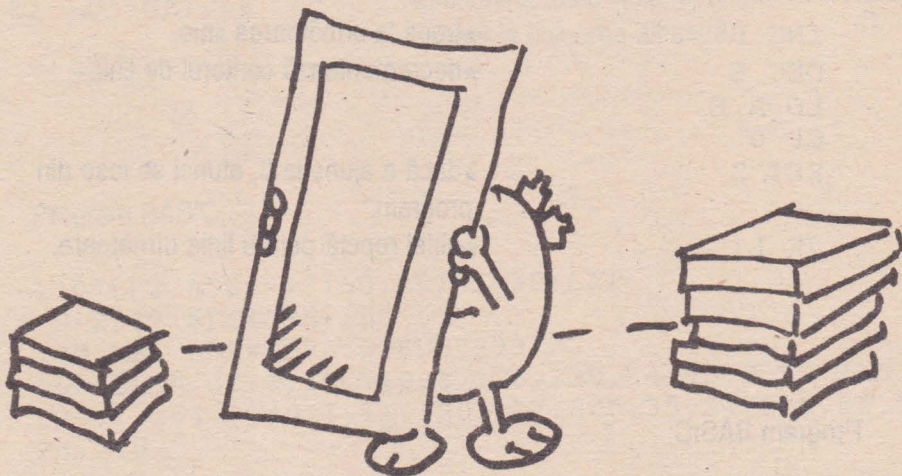
```
LET S=3719:RESTORE 310
300 LET ADR=ADR10:GOSUB 9997
310 DATA 221,33,56,199,1,0,0,17,10,10,19,7,213,
205,170,34,209,66,126,221,119,0,221,35,35,16,247,
193,4,29,123,254,0,200,24,231
```

POKE-uri:

- ★ ADR10+5, coordonata x a colțului stânga jos al ferestrei.
- ★ ADR10+6, coordonata y a colțului stânga jos al ferestrei.
- ★ ADR10+8, înălțimea ferestrei în pixeli (puncte).
- ★ ADR10+9, lungimea ferestrei în caractere (grupe de câte 8 puncte).

Pentru schimbarea adresei ferestrei se folosește următorul program:

```
LET ADR=adresa: LET ADRH=INT (ADR/256): LET
ADRL=ADR-256*ADRH: POKE ADR10+2,ADRL: POKE
ADR10+3,ADRH
```



6.5 Readucerea unei ferestre pe ecran

Programul este același, cu excepția faptului că în loc să trecem fiecare octet de pe ecran în memorie, vom trece din memorie pe ecran, adică schimbăm liniile

```
110 L2 LD A, (HL) în 110 L2 LD A, (IX+00) și
120 LD (IX+00), A în 120 LD (HL), A
```

De asemenea, dacă se dorește și pastrarea programului anterior în memorie, se schimbă adresa de ORG (notăm noua adresă ADR27).

Program BASIC:

```
320 LET ADR11=49480:LET X=35:
LET S=3719:RESTORE 340
330 LET ADR=ADR11:GOSUB 9997
340 DATA 221,33,56,199,1,0,0,17,10,10,19,7,213,
205,170,34,209,66,221,126,0,119,221,35,35,16,247,
193,4,29,123,254,0,200,24,231
```

POKE-uri:

- ★ ADR11+5, coordonata x a colțului stânga jos al ferestrei.
- ★ ADR11+6, coordonata y a colțului stânga jos al ferestrei.

Observație: coordonatele pot fi diferite de cele la care se afla fereastra când a fost salvată (poate fi mutată), dar dimensiunile și adresa ei trebuie să coincidă.

- ★ ADR11+8, înălțimea ferestrei în pixeli (puncte).

★ ADR11+9, lungimea ferestrei în caractere (grupe de câte 8 puncte).

Pentru schimbarea adresei ferestrei se folosește următorul program:

```
LET ADR=adresa: LET ADRH=INT (ADR/256): LET
ADRL=ADR-256*ADRH: POKE ADR11+2,ADRL: POKE
ADR11+3,ADRH
```

6.6 Negarea (inversarea) unei ferestre

Acest program realizează negativul unei imagini, inversând spațiile libere cu puncte. De exemplu, pentru un octet 10011011 negativul lui va fi 01100100. Acesta se poate obține cu ajutorul unui operator logic, XOR (eXclusive OR), a cărui tabelă de adevăr este următoarea:

	0	1
0	0	1
1	1	0

După cum se observă, oricare ar fi valoarea x a unui bit, avem

$$x \text{ XOR } 0 = x$$

adică rămâne neschimbat, și

$$x \text{ XOR } 1 = \text{NOT } x$$

(prin XOR 1 obținem inversarea bitului). Aceasta duce la concluzia că putem inversa toți biții unui octet prin XOR cu un octet cu toți biții poziționați pe 1, deci 11111111 = 255. De asemenea, se poate obține inversarea a numai jumătate din

fiecare octet, cu XOR 11110000, etc.

Programul permite, practic, efectuarea oricărei operații logice asupra tuturor octeților dintr-o fereastră. Poate fi folosit și operatorul AND, cu ajutorul căruia am putea "păstra" numai o parte din fiecare octet, și putem realiza, de exemplu, un efect de "topire" a conținutului unei ferestre, în mai mulți pași. Dacă operăm cu AND 10111111, bitul 6 al fiecărui octet dispăre, după aceea putem face să dispără bitul 4, cu AND 11101111, și așa mai departe până "topim" tot conținutul.

Parcurgerea ferestrei se face ca în programele de mai sus, numai că acum, în loc să fie transferat în memorie, fiecare octet din cei care o formează este supus transformării cu XOR 255.

Programul în limbaj de asamblare este următorul:

10	ORG 49520	★adresa la care compilăm programul în memorie (o notăm ADR12).
20	LD BC, 0	★colțul stânga jos al ferestrei (punctul de coordonate 0,0 - B=Y=0,C=X=0).
30	LD DE, #0A0A	★lungimea ferestrei (în caractere)=10 și înălțimea ferestrei (în pixeli)=10.
50	L1 PUSH BC	
60	PUSH DE	
70	CALL #22AA	★calculează adresa la care începe fereastra și o depune în HL.
90	POP DE	
100	LD B,D	★încarcă în B lungimea unei linii.
110	L2 LD A, (HL)	
120	XOR 255	★se execută operația logică dorită asupra octetului curent.
125	LD (HL), A	
130	INC HL	★trece la următorul byte.
150	DJNZ L2	★repetă pentru toată linia curentă.

160	POP BC	
170	INC B	★trece la următoarea linie.
180	DEC E	★decrementează contorul de linii.
190	LD A, E	
200	CP 0	
210	RET Z	★dacă a ajuns la 0, atunci se iese din program.
220	JR L1	★altfel repetă pentru linia următoare.

Program BASIC:

```

350 LET ADR12=49520:LET X=31:
LET S=3493:RESTORE 370
360 LET ADR=ADR12:GOSUB 9997
370 DATA 1,0,0,17,10,10,197,213,205,170,34,209,
66,126,238,255,119,35,16,249,193,4,29,123,254,0,
200,24,233,24,239
    
```

POKE-uri:

- ★ ADR12+1, coordonata x a colțului stânga jos al ferestrei.
- ★ ADR12+2, coordonata y a colțului stânga jos al ferestrei.
- ★ ADR12+4, înălțimea ferestrei în pixeli (puncte).
- ★ ADR12+5, lungimea ferestrei în caractere (grupe de câte 8 puncte).
- ★ ADR12+15, valoarea pentru XOR (aici 255).

6.7 Ștergerea unei ferestre

Este cea mai simplă subrutină. De data aceasta, parcurgerea ferestrei se face odată cu ștergerea tuturor octeților întâlniți. Aceștia sunt puși pe valoarea 0, adică cu toți biții 0, adică "toate punctele stinse".

Programul în limbaj de asamblare este:

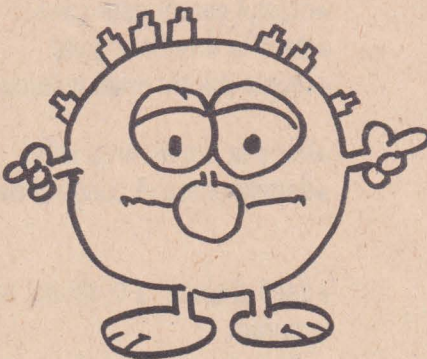
10	ORG 49555	★adresa la care compilăm programul în memorie (o notăm ADR13).
20	LD BC, 0	★colțul stânga jos al ferestrei (punctul de coordonate 0,0 - B=Y=0,C=X=0).
30	LD DE, #0A0A	★lungimea ferestrei (în caractere)=10 și înălțimea ferestrei (în pixeli)=10.
50	L1 PUSH BC	
60	PUSH DE	
70	CALL #22AA	★calculează adresa la care începe fereastra și o depune în HL.
90	POP DE	
100	LD B,D	★încarcă în B lungimea unei linii.
110	L2 LD (HL), 0	★octetul curent este șters.
130	INC HL	★trece la următorul byte.
150	DJNZ L2	★repetă pentru toată linia curentă.
160	POP BC	
170	INC B	★trece la următoarea linie.
180	DEC E	★decrementează contorul de linii.
190	LD A,E	
200	CP 0	
210	RET Z	★dacă a ajuns la 0, atunci se iese din program.
220	JR L1	★altfel repetă pentru linia următoare.

Program BASIC:

```
380 LET ADR13=49555:LET X=27:  
LET S=2550:RESTORE 400  
390 LET ADR=ADR13:GOSUB 9997  
400 DATA 1,0,0,17,10,10,197,213,205,170,34,209,66,  
54,0,35,16,251,193,4,29,123,254,0,200,24,235
```

POKE-uri:

- ★ ADR13+1, coordonata x a colțului stânga jos al ferestrei.
- ★ ADR13+2, coordonata y a colțului stânga jos al ferestrei.
- ★ ADR13+4, înălțimea ferestrei în pixeli (puncte).
- ★ ADR13+5, lungimea ferestrei în caractere (grupe de câte 8 puncte).



Mișcare pe ecran

7.1 Prezentare

Capitolul care urmează este cel mai sugestiv din punct de vedere al avantajelor codului mașină. În realizarea efectului de mișcare pe ecran, viteza interpretorului BASIC lasă mult de dorit, așa încât asemenea programe sunt aproape imposibile.

★ *Cum se realizează mișcarea ?*

Ce se întâmplă, de fapt ? Evident că nu există nimic care să se poată "mișca" în sensul propriu-zis al cuvântului, în interiorul ecranului. Acolo nu este decât vid și fascicule de electroni. Mișcarea de produce, ca și în cinematografie, prin succesiunea foarte rapidă de imagini. Un ecran de televizor schimbă imaginea de 50 de ori pe secundă, un monitor poate să ajungă la 100 de ori pe secundă.

Succedând două imagini care reprezintă de fapt două ipostaze diferite ale aceluiași obiect, se creează impresia de mișcare. De exemplu, dacă pe ecran se succed imagini ale unui om cu piciorul drept pe sol, apoi cu piciorul drept puțin ridicat, apoi ridicat mai mult și așa mai departe, se va crea impresia de mișcare; omul respectiv își va ridica piciorul, cu atât mai repede cu cât imaginile se succed mai rapid.

Mișcarea produsă pe calculator presupune doi pași:

- ★ desenarea imaginii pentru ipostaza curentă
- ★ ștergerea imaginii pentru ipostaza anterioară

Urmând acești doi pași se creează efectul. Dacă de exemplu, dorim să realizăm "mișcarea" unui punct pe ecran, de la coordonata 1 la coordonata 200, vom proceda astfel:

- 1) desenăm punctul la coordonata 0
- 2) punem punctul la coordonata curentă
- 3) ștergem punctul de la coordonata anterioară
- 4) trece la coordonata următoare
- 5) dacă nu am ajuns la coordonata 200, repetă de la pasul 2).

Algoritmul ar fi următorul:

1. coordonata=0 (coordonata de start)
2. pune(coordonata,0) (pune punctul la coordonata curentă)
3. șterge(coordonata-1,0) (șterge de la coordonata anterioară)
4. coordonata=coordonata+1 (trece la coordonata următoare)
5. dacă coordonata < 200 atunci mergi la 2)
6. stop

Cu ajutorul unui ciclu FOR.. NEXT am putea scurta programul, astfel:

```
FOR N=1 TO 200  
PLOT N,0  
PLOT INVERSE 1;N-1,0
```

NEXT N

Rulând programul, veți observa că are o viteză destul de mică. Și n-am încercat decât pentru un punct! Vă imaginați cât va ține pentru o figură mai complicată?

De exemplu, dacă dorim să realizăm mișcarea unui pătrat:

```

10 FOR N=1 TO 200
20 REM DE LA COORDONATA 1 LA 200
30 REM DESENEAZA PATRATUL
40 PLOT N,0: DRAW 10,0: DRAW 0,10
50 DRAW -10,0: DRAW 0,-10
60 REM STERGE PATRATUL ANTERIOR
70 INVERSE 1: PLOT N-1,0
80 DRAW 10,0: DRAW 0,10
90 DRAW -10,0: DRAW 0,-10
100 NEXT N

```

Viteza este suficient de mică pentru a descuraja pe oricine să mai încerce programe de mișcare în BASIC.

În cod mașină, se urmează aceiași pași, însă totul se petrece cu o viteză mult mai mare. Astfel, se pot "mișca" fără probleme, cu viteze ce ne cer chiar un sistem de "frânare", atât caractere cât și desene mai complexe, după cum veți vedea rulând programele care urmează.

★ *Structura ecranului*

Calculatoarele de tip Spectrum au o structură deosebit de complicată a ecranului, care face destul de dificilă manipularea lui de la nivelul de cod mașină.

Memoria ecran este împărțită în trei părți, fiecare reprezentând 8 linii orizontale de pe ecran. Fiecare linie orizontală este compusă din 8 linii mai mici (cu

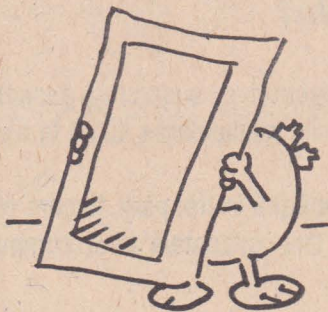
Înălțimea de 1 punct fiecare și lungimea de 256 puncte-biți, deci 32 bytes). În total o astfel de parte din ecran are lungimea de 2048 bytes.

Cele trei părți se succed în memorie una după alta, astfel încât prima va fi găsită la adresa 16384 (unde începe memoria ecran), a doua la adresa $16384+2048=18432$, iar a treia la $16384+2*2048=20480$. La adresa 22528, unde se sfârșește ultima parte, încep atributele (768 bytes), despre care am vorbit la capitolul 3 (efecte de cortină).

Fiecare parte reprezintă 8 linii de caractere a câte 8 linii de puncte, începând de sus în jos, împărțite astfel:

- ★ primii 32 bytes sunt pentru linia 1 de caractere, linia 1 de puncte
- ★ următorii 32 pentru linia 2 de caractere, linia 1 de puncte
- ★ câte 32 de bytes pentru fiecare linie de caractere până la linia 8, și linia 1 de puncte
- ★ 32 bytes pentru linia 1 de caractere, linia 2 de puncte

și așa mai departe, până la linia 8 de caractere, linia 8 de puncte. Astfel puteți observa că într-o linie de puncte și cea de sub ea este o diferență de $8*32=256$ octeți. Dacă am reprezenta schematic o linie de caractere (de exemplu prima), ea ar arăta astfel:



Linia 1 de caractere:

Caracterul:

	1	2	3	4	5	6	8-31	32
linia								
1	16384	16385	16386	16387	16388	16389	...	16415
2	16640	16641	16642	16643	16644	16645	...	16671
3-7
8	18176	18177	18178	18179	18180	18181	...	18207

Linia 2 de caractere:

Caracterul:

	1	2	3	4	5	6	8-31	32
linia								
1	16416	16417	16418	16419	16420	16421	...	16447
2	16672	16673	16674	16675	16676	16677	...	16703
...								

și așa mai departe, pentru 24 de linii de caractere.

Observații:

★ dacă avem adresa unui octet din memoria ecran, pentru a trece la octetul care reprezintă punctele de sub punctele reprezentate de primul, trebuie să adunăm la adresa acestuia valoarea 256.

★ dacă adresa unui octet din memoria ecran este într-un registru pereche, atunci trecerea la octetul de dedesubt se face prin mărirea cu 1 (incrementarea) celui mai semnificativ byte al registrului pereche, operație echivalentă cu adunarea valorii 256. De exemplu, dacă în registrul HL avem valoarea 16384, trecerea la

octetul care se află, pe ecran, sub 16384 se face prin incrementarea lui H. Astfel, HL conține valoarea 16640.

$$\begin{array}{rcccc} \text{HL} & = & 16384 & = & 64 * 256 & + & 0 \\ & & & & \uparrow & & \uparrow \\ & & & & \text{H} & & \text{L} \end{array}$$

$$\begin{array}{rcccc} \text{HL} & = & 16440 & = & 65 * 256 & + & 0 \\ & & & & \uparrow & & \uparrow \\ & & & & \text{H}+1 & & \text{L} \end{array}$$

★ Setul de caractere

În programele din acest capitol se vor face referiri și la o zonă de memorie unde se află ceea ce se numește **set de caractere**. Vom încerca să prezentăm pe scurt această noțiune.

Prin **caracter** înțelegem orice literă, cifră sau semn de punctuație tipărită pe ecran de interpretorul BASIC. Un caracter este definit ca un desen pe un careu grafic de 8 linii și 8 coloane, deci 64 de puncte. Fiecare linie este reprezentată printr-un octet, cei 8 biți ai săi oferind informația despre cele 8 puncte din linia respectivă: 1 pentru punct aprins, 0 pentru punct stins. Tipărind punctele conform acestor informații, calculatorul realizează apariția caracterului respectiv pe ecran.

Prin convenție internațională, caracterele sunt numerotate într-o anumită ordine. În calculatoarele de tip Spectrum sunt reprezentate numai caracterele de bază, ele începând la codul 32 și terminându-se la codul 127. Caracterele grafice au codurile între 128 și 164.

Pentru a putea verifica acestea, putem rula următorul program:

```
10 FOR N=32 TO 164
20 PRINT "CODUL CARACTERULUI ";CHR$(N);" ESTE ";N
```

30 NEXT N

Pe ecran vor fi afișate toate caracterele de bază folosite de calculator. Veți observa că între codurile 144 și 164 se află caracterele pe care le puteți defini din BASIC.

Pentru a ști cum să pună pe ecran aceste caractere, interpretorul le are memorate într-o zonă de memorie, numită zona setului de caractere. Adresa ei se află în cuvântul de memorie de la adresa 23606 (deci va fi dată de formula $256 * \text{PEEK}(23607) + \text{PEEK}(23606)$), deci o putem încărca într-un registru pereche (de exemplu HL) cu `LD HL,(23606)`.

Fiecare caracter necesită, pentru memorare, 8 octeți (64 biți), deci adresa unui caracter o putem afla înmulțind codul lui cu 8 și adunând rezultatul la adresa de început a setului de caractere. De exemplu, cei 8 bytes care determină cum apare litera "a" se află la adresa

$$256 * \text{PEEK}(23607) + \text{PEEK}(23606) + 8 * \text{CODE}("a")$$

În cod mașină, adresa unui caracter o putem obține astfel:

<code>LD DE,(23606)</code>	★ încărcăm în DE adresa de început a setului de caractere.
<code>LD HL,65</code>	★ încărcăm în HL codul caracterului (aici "A").
<code>ADD HL,HL</code>	★ HL devine $65 * 2$.
<code>ADD HL,HL</code>	★ HL devine $65 * 4$.
<code>ADD HL,HL</code>	★ HL devine $65 * 8$.
<code>ADD HL,DE</code>	★ adunăm $65 * 8$ la adresa setului de caractere; în HL se obține adresa la care începe reprezentarea caracterului "A" în memorie.

Dacă afișăm acești 8 bytes unul sub altul pe ecran, vom obține chiar caracterul. Ținând cont de structura ecranului, "unul sub altul" înseamnă o diferență de 256

bytes între adresele la care trebuie afișați.

Dacă, de exemplu, în DE avem adresa unui caracter, afișarea lui în colțul din stânga sus al ecranului se va face cu programul următor:

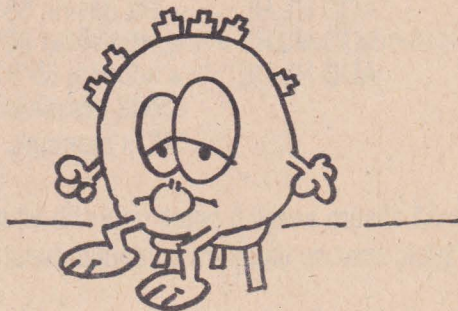
	LD HL, 16384	- adresa colțului din stânga sus al ecranului.
	LD B,8	- vom repeta afișarea pentru 8 cei bytes, care constituie cele 8 linii ale caracterului.
LIN	LD A,(DE)	- încarcă linia curentă a caracterului în acumulator.
	LD (HL),A	- o transferă la linia curentă de pe ecran.
	INC DE	- DE trece la adresa liniei următoare a caracterului.
	INC H	- HL trece (prin adunare cu 256 sau mărirea cu 1 a MSB) la adresa liniei de dedesubtul liniei curente de pe ecran, la care trebuie să fie pus următorul byte al caracterului.
	DJNZ LIN	- repetă pentru toate cele 8 linii ale caracterului (micșorează B și repetă de la LIN dacănu a ajuns la 0).

În BASIC, programul care ar executa același algoritm ar fi:

```

10 LET HL=16384
20 LET DE=256*PEEK 23607+PEEK 23606 + 8*CODE "a"
30 REM DE CONTINE ADRESA CARACTERULUI "a"
40 FOR N=1 TO 8
50 POKE HL,PEEK (DE)
60 LET DE=DE+1
70 LET HL=HL+256
80 NEXT N

```

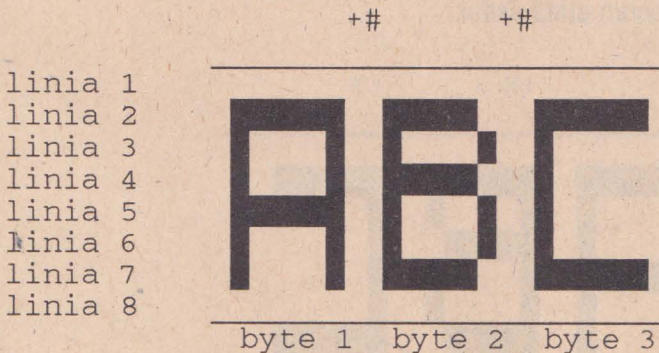


7.2 Deplasare orizontală

Aceasta este o subrutină care va fi folosită mai târziu în programul de "defilare" a unui text pe o bandă orizontală; ea poate fi apelată și separat, efectuând mișcarea "pixel cu pixel" pe banda respectivă.

Algoritmul este următorul: pentru fiecare linie de caractere din bandă (formată pe ecran din 8 linii subțiri de puncte) se analizează locațiile, de la sfârșit la început, deplasându-se câte un pixel (bit) la stânga sau la dreapta iar în locul rămas liber punând un punct dacă din locația care urma trebuia, prin deplasare, să dispară un punct aprins.

Mai concret : presupunem o bandă formată din trei caractere, deci 8 linii cu trei locații pe fiecare linie (un caracter este reprezentat pe 8 linii și 8 coloane). Pe ecran, arată astfel (presupunem că pe ea se află deja trei caractere, A, B și C):



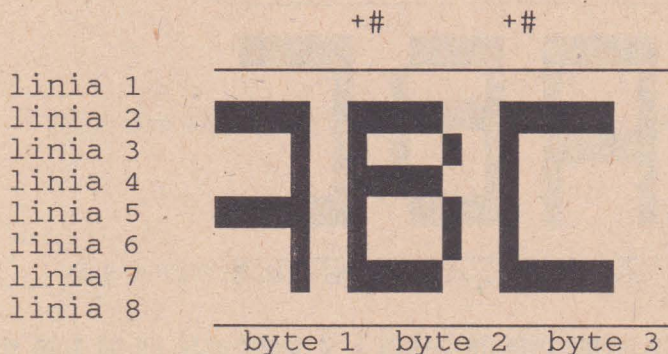
Analiza începe cu primul byte de la linia 1. Îl deplasează un bit spre stânga, stingând astfel bitul + dacă este aprins și analizează bitul notat cu #, cel care urmează să dispară din locația următoare. Dacă acesta este aprins, atunci îl aprinde pe cel notat cu + de la primul byte. Repetă această operație pentru toți octeții din linie până la ultimul inclusiv, după care trece la linia următoare.

În cazul nostru, la primul pas, deoarece nu sunt biți noi de aprins, deplasarea se face simplu. Acum configurația este următoarea:



Pentru prima linie, totul se desfășoară la fel, pe locațiile notate cu # neexistând biți aprinși. La liniile 2-7 însă, bitul # este aprins, ceea ce determină aprinderea bitului +.

Acum banda de ecran arată astfel:



Repetând de încă 6 ori, în byte 1 va fi litera "B", în byte 2 va fi litera "C", iar byte 3 va fi liber. Și așa mai departe, procesul poate fi repetat de oricâte ori se dorește

acest lucru.

Algoritmul scris în limbaj de asamblare este dat de programul următor:

10	ORG 52000	★adresa la care este compilat.
20	SDS LD HL, 16384	★adresa primului pătrat din bandă (inițializată aici la prima linie din ecran).
30	PUSH HL	★păstrează adresa în stivă.
40	LD C, 8	★numărul de linii subțiri dintr-o bandă.
50	LD B, 31	★numărul de caractere al benzii.
60	L0 PUSH BC	★păstrează parametrii benzii.
70	L1 SLA (HL)	★deplasează la stânga octetul de la adresa din HL.
80	INC HL	★trece la octetul următor.
90	BIT 7, (HL)	★analizează bitul * din acesta.
100	JR NZ, L4	★dacă este aprins, sare la eticheta L4, definită mai jos.
110	L2 DJNZ L1	★dacă linia nu s-a terminat, atunci repetă pentru următorul octet.
120	POP BC	★reface parametrii inițiali ai benzii.
130	POP HL	★reface adresa primei linii din bandă.
140	DEC C	★semnalizează adresa următoare, prin decrementarea contorului de linii.
150	INC H	★actualizează adresa pentru linia următoare.
160	PUSH HL	★păstrează adresa.
170	LD A, C	★analizează dacă au fost deplasate toate cele 8 linii, comparând contorul de linii cu 0.
180	CP 0	
190	JR NZ, L0	★dacă nu este 0, reia pentru linia următoare.
200	POP HL	★dacă da, atunci reface stiva dinainte de apelarea programului.
210	RET	★revenire în programul apelant.
220	L4 DEC HL	★această subrutină poziționează bitul +. HL conține adresa cu bitul # aprins, deci este decrementat la

		adresa anterioară.
230	SET 0, (HL)	★este aprins bitul +.
240	INC HL	★HL conține adresa locației curente.
250	JR L2	★procesul se reia de unde a fost întrerupt.

Observații:

★ Programul asamblat va putea fi apelat cu RANDOMIZE USR 52000 (sau adresa care a fost definită la ORG). Dacă se dorește repetarea lui de x ori, va putea fi pus într-un ciclu FOR . . NEXT de genul:

```
10 FOR n=1 to x:RANDOMIZE USR 52000:NEXT n
```

repetarea făcându-se cu o viteză destul de mică, deoarece este apelat din BASIC.

★ Dacă nu se dispune de un asamblor, programul în cod mașină poate fi plasat direct în memorie, astfel:

```
410 RESTORE 440:LET ADR14=52000:LET ADR=ADR14
420 LET S=4000:LET X=36
430 GOSUB 9997
440 DATA 33,0,64,229,14,8,6,31,197,203,38,35,
203,126,32,14,16,247,193,225,13,36,229,121,
254,0,32,236,225,201,43,203,198,35,24,236
```

★ Pot fi schimbate locațiile ce conțin parametrii benzii, putând astfel obține deplasarea unei fâșii în orice parte a ecranului. Adresa de început a benzii se modifică cu ajutorul subrutinei următoare:

```
9990 INPUT "COORDONATELE:LINIE=";X:INPUT
"COLOANA=";Y
9991 LET BADR=16384+2048*INT(X/8)+Y+32*(X
-8*INT(X/8))
```

```

9992 LET BH=INT (BADR/256)
9993 POKE ADR+1,BADR-256*BH:POKE ADR+2,BH
9994 RETURN

```

care este apelată cu:

```
LET ADR=ADR14:GOSUB 9990
```

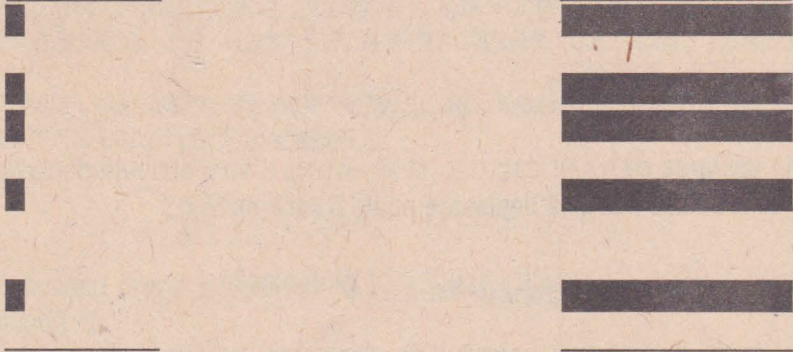
Parametrii începutului de linie se dau ca și la instrucțiunea "PRINT AT". Dacă sunt deja inițializați, pot fi plasați în subrutina de cod cu GOSUB 9991 (se evită întrebările).

★ Lungimea liniei se schimbă cu

```
POKE ADR14+7, lungime
```

★ Programul în cod mașină oferă și posibilitatea controlului modelului cu care este umplut spațiul rămas liber prin deplasarea benzii, prin scrierea în dreapta acesteia a unui caracter definitor. De exemplu,

caracterul _____ va determina umplerea liniei cu



Dacă pe prima coloană a caracterului nu este nici un bit aprins (cum este cazul la toate caracterele din setul definit în calculatoarele compatibile SPECTRUM),

atunci linia rămâne liberă.

★ Pentru o deplasare mai rapidă, subrutina poate fi apelată din cod mașină cu următorul program:

```

260          ORG 52040
270          LD B,70      -numărul de deplasări.
280 LL0     PUSH BC
290          CALL SDS    -cheamă rutina de deplasare.
300          POP BC
310          DJNZ LL0    -micșorează B cu 1 și repetă dacă nu a
                        ajuns la 0.
320          RET

```

echivalent cu programul BASIC:

```

450 RESTORE 480:LET ADR15=52040:LET ADR=ADR15:LET
X=10
460     LET     A1=INT(ADR14/256):LET     A2=ADR14-
256*A1:GOSUB 9980
470 REM  A1 SI A2 CONTIN ADRESA PROGRAMULUI
CHEMAT PENTRU REPETARE,AICI CEL DE SCROLL
DREAPTA-STÂNGA
480 DATA 6,70,197,205,A2,A1,193,16,249,201

```

Prin apelarea cu RANDOMIZE USR ADR15 va fi obținută deplasarea cu 70 pixeli a benzii. Această deplasare poate fi schimbată cu

POKE ADR15+1,nr.deplasări.

★ Pentru schimbarea sensului de deplasare (stânga-dreapta), linia 20 din programul în limbaj de asamblare se modifică în

```
20 SDS     LD HL,16415
```

16415 reprezentând adresa de sfârșit (!!) a benzii (obținută adunând lungimea benzii la adresa de început), precum și liniile:

```
70 L1      SRA (HL)
75         RES 7, (HL)
```

★ în locul liniilor 70 și 75 poate fi folosită instrucțiunea SRL (HL) care, după ce deplasează conținutul celulei de memorie adresate de HL resetează automat bitul 7.

```
80         DEC HL
90         BIT 0, (HL)
220 L4     INC HL
230         SET 7, (HL)
240         DEC HL
```

★ Programul BASIC este următorul:

```
490 RESTORE 510:LET ADR16=52050:LET ADR=ADR16:LET
X=38
500 LET S=4436:GOSUB 9997
510 DATA 33,31,64,229,14,8,6,31,197,203,46,203,
190,43,203,70,32,14,16,245,193,225,13,36,229,
121,254,0,32,234,225,201,35,203,254,43,24,236
```

Subrutina de deplasare este activată cu RANDOMIZE USR ADR16.
Parametrii benzii pot fi modificați cu

```
LET ADR=ADR16:GOSUB 9000
```

introducând drept coordonate pe cele ale **sfârșitului** de bandă. Lungimea se schimbă cu

```
POKE ADR16+7, lungime
```

★ Pentru repetare, se poate folosi linia de date de la repetarea mișcării în sens

invers, astfel:

```
520 LET ADR17=52090:LET X=10:LET
ADR=ADR17:RESTORE 480
530 LET A1=INT(ADR16/256):LET A2=ADR16-256*A1
540 GOSUB 9980
```

Numărul de repetări se modifică cu

```
POKE ADR17+1,nr
```

7.3 Rotirea unei benzi horizontale pe ecran

Acest program este doar o trecere la nivelul următor, unde se va realiza "defilarea" unui text pe una sau mai multe benzi de pe ecran. El se deosebește de cel anterior doar prin faptul că, după deplasarea benzii, ceea ce dispăre într-o parte reapare în cealaltă, obținându-se astfel o "rotire" a conținutului.

Algoritmul conține în plus o testare a primului bit din bandă, înainte de rotire, și o subrutină care, în caz că acest bit este aprins, îl poziționează în locul ce rămâne liber după deplasare. În rest, programul este identic cu cel anterior, de aceea vor fi explicate doar liniile care apar în plus:

10	ORG 52100	★ adresa; o notăm ADR18
20	RDS LD HL,16384	
30	PUSH HL	
40	LD C,8	
50	LD B,32	
60	L0 PUSH BC	
70	BIT 7,(HL)	★testează primul bit din bandă.
80	JR NZ,P1	★dacă este aprins, îl pune la sfârșit.
90	L1 SLA (HL)	


```

100      INC HL
110      BIT 7, (HL)
120      JR NZ, L4
130 L2    DJNZ L1
135      RES 7, (HL)

```

★după ce a deplasat linia, stinge bitul care a fost aprins de subrutina de rotire (ca să nu fie luat ca aprins și la următoarea deplasare).

```

140      POP BC
150      POP HL
160      DEC C
170      INC H
180      PUSH HL
190      LD A, C
200      CP 0
210      JR NZ, L0
220      POP HL
230      RET
240 L4    DEC HL
250      SET 0, (HL)
260      INC HL
270      JR L2
280 P1    PUSH HL
290      PUSH BC
300      LD C, B
310      LD B, 0
320      ADD HL, BC

330      SET 7, (HL)
340      POP BC
350      POP HL
360      JR L1

```

★formează în BC lungimea benzii.
★adună lungimea benzii la adresa de început.
★aprinde ultimul bit din bandă.
★reia procesul de unde a fost întrerupt.

★ Apelarea programului se face cu

```
RANDOMIZE USR ADR18
```

★ Programul BASIC echivalent pentru rotirea de la dreapta la stânga este următorul:

```
550 LET ADR18=52100:RESTORE 570:LET S=6706
560 LET X=55:LET ADR=ADR18:GOSUB 9997
570 DATA 33,0,64,229,14,8,6,32,197,203,126,32,
29,203,38,35,203,126,32,16,16,247,203,190,193,
225,13,36,229,121,254,0,32,230,225,201,43,203,
198,35,24,234,229,197,72,6,0,237,74,203,254,193,
225,24,214
```

★ Pentru schimbarea parametrilor se folosește tot subrutina definită la 9000, astfel:

```
LET ADR=ADR18:GOSUB 9000
```

unde se introduc coordonatele începutului de bandă. Lungimea se schimbă cu

```
POKE ADR18+7, lung
```

★ Pentru repetare, se adaugă liniile:

```
370      ORG 52160
380      LD B,70
390 L5    PUSH BC
400      CALL RDS
410      POP BC
420      DJNZ L5
430      RET
```

sau programul BASIC corespunzător:

```

580 LET ADR19=52160:LET ADR=ADR19:RESTORE 480:LET
X=10
590 LET A1=INT(ADR18/256):LET A2=ADR18-256*A1
600 LET A41=A1:LET A42=A2:GOSUB 9980

```

★ Dacă viteza pare prea mare, se poate folosi un ciclu FOR . .NEXT care să conțină RANDOMIZE USR ADR18 și eventual o pauză, sau ca o alternativă se poate introduce în programul scris în limbaj de asamblare instrucțiunea HALT la linia 405.

În cazul acesta programul BASIC devine:

```

610 LET ADR29=52170:RESTORE 630:LET ADR=ADR29:LET
X=11
620 GOSUB 9980
630 DATA 6,70,197,205,A42,A41,118,193,16,248,201

```

★ Programele de repetare construite până acum au un dezavantaj: nu acceptă întreruperea în cazul apăsării unei taste, nici măcar în cazul apăsării lui CAPS SHIFT+BREAK. Pentru o viteză lentă sau chiar o viteză rapidă cu un număr mare de repetări, este destul de incomod, de aceea se poate folosi o variantă care realizează întoarcerea în BASIC la apăsarea unei taste de pe claviatură:

```

640 LET ADR21=52181:RESTORE 660:LET X=18:LET
ADR=ADR21
650 GOSUB 9980
660 DATA 6,70,197,205,A42,A41,118,205,142,2,123,
254,255,193,192,16,241,201

```

Programul în cod mașină este următorul:

```

379          ORG 52181
380          LD B,70      -nr. de repetări.
390 L5       PUSH BC
400          CALL RDS

```

410	HALT	-dacă se înlocuiește cu NOP, viteza crește.
420	CALL 654	-apelează subrutina de citire a tastaturii.
430	LD A, E	
440	CP 255	-verifică dacă a fost apăsată vreo tastă.
450	POP BC	
460	RET NZ	-dacă da, revine in BASIC.
470	DJNZ L5	-dacă nu, continuă până la numărul total de repetări.
480	RET	-dacă este înlocuit cu JR L5, rotirea continuă până la apăsarea unei taste, fără să mai țină cont de numărul de repetări.

★ Cu

POKE ADR21+17,24:POKE ADR21+18,239

rotirea continuă până la apăsarea unei taste (indiferent dacă s-au terminat repetările cerute). Mai avem și

POKE ADR21+1,nr.repetări

precum și locațiile ADR21+4 și ADR21+5 , unde se află adresa programului care este repetat (aici se poate afla adresa oricărui program care poate fi repetat, de exemplu programul de scroll).

★ Pentru a mări viteza, se schimbă octetul care reprezintă instrucțiunea HALT:

POKE ADR21+6,0

★ Pentru rotirea benzii în sens invers, este necesar următorul program:

670 LET ADR22=50200:RESTORE 690:LET ADR=ADR22:LET X=59

680 LET S=7324:GOSUB 9997

```
690 DATA 33,31,64,229,14,8,6,31,197,203,70,32,
33,203,46,203,190,43,203,70,32,18,16,245,203,
46,203,190,193,225,13,36,229,121,254,0,32,226,
225,201,35,203,254,43,24,232,229,197,72,6,0,237,
66,203,254,193,225,24,210
```

★ Pentru schimbarea parametrilor benzii se folosește subrutina de la 9000, astfel:

```
LET ADR=ADR22:GOSUB 9000
```

introducând coordonatele sfârșitului benzii.

Lungimea se schimbă cu

```
POKE ADR22+7, lungime
```

★ Pot fi folosite aceleași programe de repetare, însă introducând adresa ADR8 la instrucțiunea de CALL, sau schimbând în BASIC:

```
LET A81=INT(ADR22/256):LET A82=ADR22-256*A81
și, respectiv:
```

★ pentru programul de repetare rapidă:

```
POKE ADR19+4,A82:POKE ADR19+5,A81
```

★ pentru programul de repetare cu pauză:

```
POKE ADR20+4,A82:POKE ADR20+5,A81
```

★ pentru programul de repetare cu verificarea tastaturii:

```
POKE ADR21+4,A82:POKE ADR21+5,A81
```

★ pentru revenirea la sensul de rotație inițial se dau aceleași POKE, dar cu A41 și A42.

★ Să prezentăm în final încă un program de repetare, în care fiecare deplasare este însoțită de un sunet:

```

375      ORG 52260
380      LD B,70
390 L5   PUSH BC
400      CALL RDS
410      HALT          ★se înlocuiește cu NOP pentru mărirea vitezei.
420      LD DE,1      ★durata sunetului.
430      LD HL,2000   ★frecvența sunetului.
440      CALL 949     ★produce sunetul
450      CALL 654     ★verifică tastatura
460      LD A,E
470      CP 255
480      POP BC
490      RET NZ
500      DJNZ L5
510      RET          ★sau JR L5 pentru repetare până la apăsarea unei taste.
    
```

echivalent cu programul BASIC:

```

700 LET ADR23=52260:RESTORE 720:LET X=27:LET
ADR=ADR23
710 GOSUB 9980
720 DATA 6,70,197,205,A82,A81,118,17,1,0,33,208,
7,205,181,3,205,142,2,123,254,255,193,192,16,
232,201
    
```

★ Variante:

★ pentru repetare până la apăsarea unei taste:

```
POKE ADR23+26,24:POKE ADR23+27,230
```

★ durata sunetului se schimbă prin:

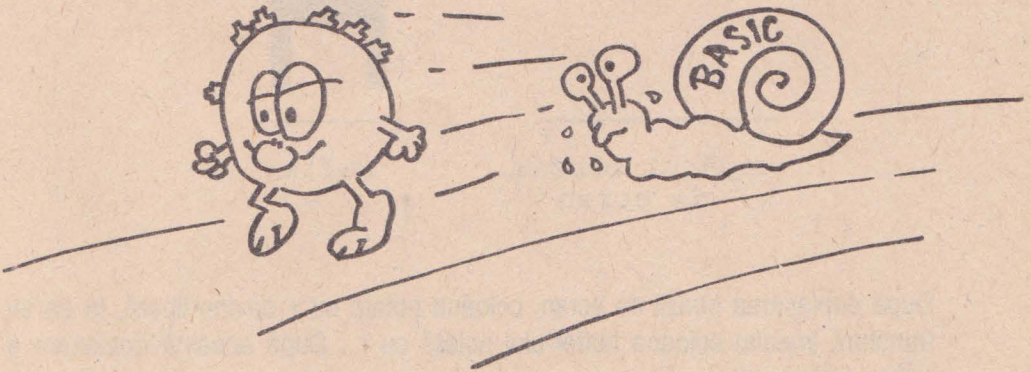
```
LET L=lungime:LET LH=INT(L/256):LET LL=L-256*LH:  
POKE ADR23+8,LL:POKE ADR23+9,LH
```

★ frecvența sunetului se schimbă astfel:

```
LET F=frecventa:LET FH=INT(F/256):LET FL=F-256*FH:  
POKE ADR23+11,FL:POKE ADR23+12,FH
```

★ viteza de repetare se poate mări cu

```
POKE ADR23+6,0
```

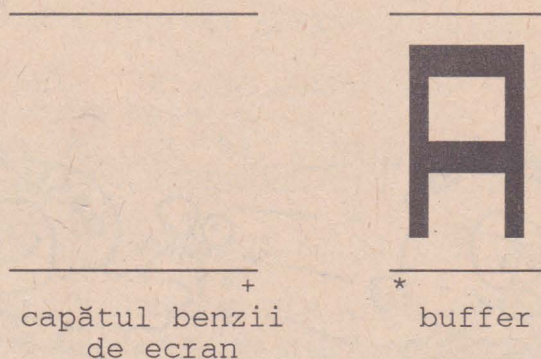


7.4 Defilarea orizontală a unui text pe ecran

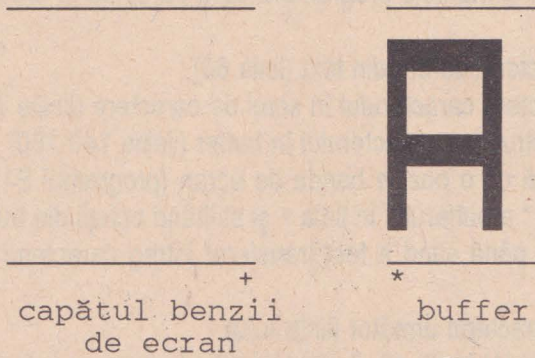
Algoritmul programului este următorul: fiecărui caracter din text i se preia structura de 8 octeți (bytes) din setul de caractere și este pusă într-o zonă tampon (buffer), în cazul programului de mai jos, la adresa 50290; apoi se efectuează 8 deplasări spre stânga ale benzii de ecran, la fiecare deplasare umplându-se locul rămas liber cu cei 8 biți din stânga ai caracterului din buffer.

Înainte de a trece la următoarea deplasare, se shiftează byte cu byte caracterul din buffer, pentru a i se prelua următorii 8 biți.

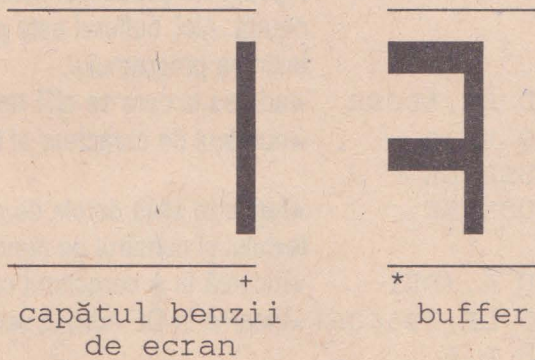
De exemplu, în buffer avem caracterul "A". Înainte de primul pas, situația este următoarea:



După deplasarea benzii de ecran, coloana notată cu + devine liberă; în ea se transferă imediat coloana buffer-ului notată cu *. După această deplasare a bufferului, avem:



Datorită faptului că pe linia marcată * nu există nici un bit poziționat (nici un punct aprins), și linia + a benzii de ecran a rămas vidă. După deplasarea octeților din buffer cu câte un bit la stânga, pe coloana notată cu * există 6 biți poziționați, iar aceștia vor fi trecuți la următorul pas în coloana +, shiftând imediat din nou bufferul, pentru a reînnoi coloana * :



și așa mai departe, până când tot caracterul a fost transferat.

În esență, componentele programului sunt parcurse în ordinea următoare:

- 1) preia caracterul curent din text (linia 60)
- 2) caută structura caracterului în setul de caractere (liniile 70-130)
- 3) transferă structura caracterului în buffer (liniile 140-180)
- 4) deplasează cu o poziție banda de ecran (programul P1), mutând în același timp coloana * a bufferului în linia + și shiftând octeții din buffer (liniile 220-410)
- 5) repetă (4) până când a fost transferat întreg caracterul din buffer în bandă (linia 440)
- 6) trece la caracterul următor (linia 470)
- 7) repetă de la (1) la (6) până când textul a fost transmis în întregime (linia 480)
- 8) STOP (revenire în BASIC-linia 490).

Programul în limbaj de asamblare este următoarul:

10	ORG 50300	★adresa la care este asamblat programul. ATENȚIE ! La schimbarea adresei, se va re poziționa și bufferul (linia 140) într-o zonă neutră. Aici, bufferul este plasat chiar înaintea programului.
20	LD DE, 55000	★adresa la care se află textul.
30	LD B, 32	★numărul de caractere al textului.
40	L9 PUSH DE	
50	PUSH BC	★reține în stivă datele de mai sus (adresa textului și numărul de caractere).
60	LD A, (DE)	★încarcă în A caracterul curent.
70	SCAR LD DE, (23606)	★încarcă în DE adresa setului de caractere.
80	LD L, A	
90	LD H, 0	★încarcă HL cu valoarea din A.
100	ADD HL, HL	
110	ADD HL, HL	
120	ADD HL, HL	★înmulțește HL cu 8.
130	ADD HL, DE	★adună adresa setului de caractere; în HL

140		LD DE, 50290	s-a obținut adresa caracterului din A. ★adresa bufferului (adresa de ORG -10).
150		PUSH DE	★reține adresa bufferului în stivă.
160		LD BC, 8	★anunță câți bytes trebuie transferați din setul de caractere în buffer (lungimea unui caracter este de 8 bytes).
170		LDIR	★transferă cei 8 bytes în buffer.
180		POP DE	★DE conține din nou adresa de început a bufferului.
190	SC8	LD B, 8	★numărul de linii ce vor fi transferate din buffer în banda de ecran.
200	L3	PUSH BC	
210		PUSH DE	
220		LD HL, 16385	★încarcă în HL adresa de început a benzii.
230		PUSH HL	
240		LD C, 8	★numărul de linii subțiri ale benzii.
250		LD B, 31	★lungimea benzii în caractere.
260	L0	PUSH BC	
270	L1	SLA (HL)	★deplasează spre stânga conținutul adresei HL.
280		INC HL	★trece la următorul byte al liniei.
290		BIT 7, (HL)	★verifică dacă bitul din stânga este aprins.
300		JR NZ, L4	★dacă da, atunci sare la L4 (și transferă în bitul din dreapta al adresei curente).
310	L2	DJNZ L1	★repetă pentru întreaga lungime a benzii.
320		JR L7	★sare la L7 (aprinde ultimul bit al liniei dacă bitul corespunzător din buffer este aprins și deplasează octetul respectiv din buffer).
330	L8	POP BC	
340		POP HL	
350		DEC C	★trece la următoarea linie subțire a benzii.
360		INC H	★formează adresa ei în HL.
370		PUSH HL	
380		LD A, C	

390	CP 0	★verifică dacă linia deplasată înainte a fost ultima linie a benzii.
400	JR NZ, L0	★dacă nu, atunci reia de la L0.
410	POP HL	
420	POP DE	
430	POP BC	
440	DJNZ L3	★repetă până la transferul complet al caracterului din buffer.
450	POP BC	
460	POP DE	
470	INC DE	★trece la următorul caracter.
480	DJNZ L9	★dacă nu a fost terminat textul, atunci reia de la L9.
490	RET	★reîntoarcere în BASIC.
500 L4	DEC HL	★subrutină care poziționează bitul din dreapta al unei adrese; este chemată dacă locația alăturată are bitul ce trebuie transferat aprins. Mai întâi, formează în HL adresa locației care conține bitul ce trebuie aprins.
510	SET 0, (HL)	★aprinde bitul
520	INC HL	★reface în HL adresa locației curente.
530	DJNZ L1	
540 L7	EX DE, HL	★subrutina transferă bitul * al liniei din buffer în bitul + al adresei din DE. întâi inversează pointerii, deoarece nu există instrucțiuni de lucru pe biți cu adrese din DE.
550	DEC DE	
560	SLA (HL)	★rotește linia din buffer.
570	JR C, L5	★dacă în poziția * a fost un bit aprins, atunci sare la L5.
580 L6	INC HL	
590	EX DE, HL	
600	JR L8	★dacă nu, atunci reface adresele inițiale și

		revine la L8.
610	L5 INC HL	
620	EX DE, HL	★reface pointerii.
630	SET 0, (HL)	★poziționează ultimul bit (+) al benzii.
640	JR L8	★reia de la L8.

Observații:

★ Programul BASIC este următorul:

```

730 LET ADR24=50300:LET ADR=ADR24:LET X=100
740 LET S=11776:RESTORE 750:GOSUB 9997
750 DATA 17,216,214,6,32,213,197,26,237,91,54,
92,111,38,0,237,106,237,106,237,106,237,90,
17,114,196,213,1,8,0,237,176,209,6,8,197,213,
33,1,64,229,14,8,6,31,197,203,38,35,203,126,32,
27,16,247,24,29,193,225,13,36,229,121,254,0,32,
234,225,209,193,16,219,193,209,19,16,184,201,43,
203,198,35,16,218,235,27,203,38,56,4,35,235,24,
219,35,235,203,198,24,213

```

★ Apelarea se face cu

```
RANDOMIZE USR ADR24
```

după ce a fost introdus textul în memorie.

★ Textul care urmează să fie scris poate fi introdus din BASIC, printr-un program de forma:

```

LET AŞ="....." (textul care trebuie introdus)
LET AT=55000 (adresa textului)
FOR N=1 TO LEN AŞ:POKE AT+N-1,CODE AŞ(N):NEXT N

```

După rularea acestei subrutine, se va inițializa adresa textului și lungimea lui în

programul în cod mașină, după cum urmează:

★ adresa textului este conținută în locațiile ADR10+1 și ADR10+2 , putând fi modificată astfel:

```
LET AT=adresa text : LET ATH=INT(AT/256) : LET
ATL=AT-256*ATH
POKE ADR24+1,ATL : POKE ADR24+2,ATH
```

★ lungimea textului poate fi modificată cu

```
POKE ADR24+4 , lungime (în cazul nostru , LEN A$).
```

★ Pot fi puse în memorie mai multe texte, care să fie chemate prin simpla schimbare a datelor conținute în programul în cod mașină.

★ Adresa de început a benzii de ecran se află la locațiile ADR24+38 și ADR24+39 , iar lungimea ei la ADR24+44.

★ **Diverse efecte pot fi obținute introducând în programul scris în limbaj de asamblare linia**

```
405 JR L10
```

și modificând linia 410 în

```
410 L11 POP HL
```

Acum, se pot pune la sfârșitul programului, după declararea etichetei L10, toate efectele dorite, având grijă ca ele să se încheie cu instrucțiunea JR L11.

OBSERVAȚIE: se poate folosi și CALL , dar programul ar deveni mai greu relocabil.

EXEMPLU:

650	L10	HALT	★încetinire; dacă este înlocuit cu NOP, viteza revine la normal.
660		LD DE, 1	★durata sunetului.
670		LD HL, 2000	★frecvența sunetului.
680		CALL 949	★produce sunetul.
690		CALL 654	★citește tastatura.
700		LD A, E	
710		CP 255	★verifică dacă s-a apăsă o tastă.
720		JR NZ, RT	★dacă da, atunci sare la rutina de revenire în BASIC.
730		JR L11	★dacă nu, atunci reia programul de unde a fost întrerupt pentru producerea efectelor.

Subrutina care urmează este necesară doar dacă s-a optat pentru reînțoarcerea în cazul apăsării unei taste. Ne aflăm însă în mijlocul programului, și ca să putem întrerupe trebuie să refacem stiva, în care se află 5 regiștri pereche:

```

740 RT POP DE
750     POP DE
760     POP DE
770     POP DE
780     POP DE
790     RET

```

Programul rezultat poate fi introdus din BASIC astfel:

```

760 LET ADR25=50410:LET X=102:LET
S=11934:RESTORE 780
770 LET ADR=ADR25:GOSUB 9997
780 DATA 17,216,214,6,32,213,197,26,237,91,54,
92,111,38,0,237,106,237,106,237,106,237,90,
17,225,196,213,1,8,0,237,176,209,6,8,197,213,
33,1,64,229,14,8,6,31,197,203,38,35,203,126,32,

```

27, 16, 247, 24, 29, 193, 225, 13, 36, 229, 121, 254, 0, 32,
234, 24, 33, 225, 209, 193, 16, 217, 193, 209, 19, 16, 182,
201, 43, 203, 198, 35, 16, 216, 235, 27, 203, 38, 56, 4, 35,
235, 24, 217, 35, 235, 203, 198, 24, 211

NOTA: adresa de buffer a fost schimbată în 50401.

7.5 Deplasarea unei benzi verticale pe ecran

Dacă la deplasarea unei benzi orizontale algoritmul era relativ simplu, fiecare bit trecând în locul celui care urma, la deplasarea pe verticală lucrurile se complică mult. În primul rând, pe ecran un byte este format din 8 biți dispuși orizontal, nu vertical, deci dacă vrem să mutăm un bit mai jos sau mai sus nu mai putem rezolva acest lucru printr-o simplă rotire în cadrul octetului care îl conține, ci trebuie să mutăm tot octetul în cel care se află dedesubt sau deasupra.

Structura întreprinsă a memoriei ecran la calculatoarele compatibile SPECTRUM face și mai dificilă această operație, obligându-ne să calculăm separat, pentru fiecare octet, adresa lui și a celui care urmează să îi preia conținutul. Pentru aceasta avem două soluții: să construim noi un program care să realizeze această operație, sau să folosim o subrutină deja existentă în memoria ROM a calculatoarelor de tip SPECTRUM, și anume cea care, primind în registrul pereche BC coordonatele unui punct de pe ecran, întoarce în HL adresa lui.

Putem optimiza acest algoritm calculând adresa numai din caracter în caracter pe verticală, în interiorul unui caracter putând trece de la o linie la alta prin incrementarea sau decrementarea byte-ului cel mai semnificativ al pointerilor (adică adunarea sau scăderea valorii 256 - vezi 7.1).

Programul în limbaj de asamblare este următorul:

10	ORG 50600	★adresa de început a programului (o notăm ADR26).
20	I1 LD BC, 0	★marginea de jos a benzii de ecran (x=0,y=0).
30	L0 PUSH BC	★salvează coordonatele în stivă.
40	CALL 8874	★obține în HL adresa care conține punctul din BC.
50	PUSH HL	
60	POP DE	★copiază adresa în DE.
70	DEC H	★HL devine pointer la adresa de deasupra celei conținute în DE.
80	LD B, 7	★B conține numărul de repetări pentru deplasarea caracterului pe verticală.
90	L1 LD A, (HL)	
100	LD (DE), A	★conținutul adresei din HL trece în DE.
110	DEC D	★DE trece la adresa de deasupra.
120	DEC H	★HL trece la adresa de deasupra.
130	DJNZ L1	★repetă pentru tot caracterul.
140	POP BC	
150	PUSH BC	★readuce adresele din stivă.
160	PUSH DE	★înmagazinează ultimul pointer.
170	LD A, B	
180	ADD A, 8	★poziționează coordonatele la începutul caracterului care urmează.
190	LD B, A	
200	CP 175	★verifică dacă s-a atins marginea superioară a benzii (aici, aceasta este și marginea superioară a ecranului).
210	JR NC, RET	★dacă da, atunci sare la subrutina care realizează eliberarea stivei și întoarcerea în programul apelant.
220	CALL 8874	★dacă nu, atunci calculează adresa de început a următorului caracter.
230	POP DE	★reface pointerul anterior.

240	LD A, (HL)	
250	LD (DE), A	★transferă ultima linie a caracterului care urmează să fie deplasat în prima linie a caracterului care a fost deplasat anterior.
260	POP BC	★reface coordonatele.
270	LD A, B	
280	ADD A, 8	
290	LD B, A	★modifică coordonatele pentru trecerea la caracterul următor.
300	JR L0	★sare la început.
310	RET POP DE	★subrutina de revenire în BASIC ; mai întâi reface stiva, în care se află doi regiștri pereche.
320	POP DE	
330	RET	★reânțoarcere în programul apelant (sau în interpretorul BASIC).

★ Apelarea se face cu

RANDOMIZE USR 50600

★ Programul BASIC echivalent este următorul:

```

790 LET ADR26=50600:LET X=45:LET
S=5265:RESTORE 810
800 LET ADR=ADR26:GOSUB 9997
810 DATA 1,0,0,197,205,170,34,229,209,37,6,7,
126,18,21,37,16,250,193,197,213,120,198,8,71,
254,175,48,13,205,170,34,209,126,18,193,120,
198,8,71,24,217,209,209,201
    
```

★ Se pot opera următoarele schimbări:

★ la adresele ADR26+1 și ADR26+2 se află coordonatele unui punct din partea de jos a benzii de ecran (aici cel de coordonate $x=0, y=0$):

```
POKE ADR26+2, coordonata x
POKE ADR26+3, coordonata y (divizibilă cu 8)
```

★ la adresa ADR26+27 se află coordonata capătului superior al benzii (aici $y=175$).

★ Pot fi folosite programele de repetare obișnuite, definite în cadrul subcapitolului 7.3, schimbând adresa programului apelat cu adresa acestui program, aici 50600.

★ Pentru inversarea sensului de deplasare, se modifică următoarele linii din programul în limbaj de asamblare:

```
10      ORG 50650      ★adresa programului în cod mașină (o notăm
                    ADR27).
20 SJ   LD BC,44800   ★coordonatele unui punct de deasupra benzii
                    (y=175,x=0).
70      INC H
110     INC H
120     INC D
180     SBC A,8
        Linia 200 dispăre !
210     JR C,RET
280     SUB 8
```

Astfel, operațiile sunt schimbate pentru sensul invers de parcurgere a benzii.

★ Programul BASIC este următorul:

```
830 LET ADR27=50650:LET X=43:LET
S=5058:RESTORE 850
```

```

840 LET ADR=ADR27:GOSUB 9997
850 DATA 1,0,175,197,205,170,34,229,209,36,6,7,
126,18,36,20,16,250,193,197,213,120,222,8,71,56,
13,205,170,34,209,126,18,193,120,214,8,71,24,
219,209,209,201

```

★ POKE-uri:

★ la adresele ADR27+1 și ADR27+2 se află coordonatele unui punct din partea de sus a benzii de ecran (aici cel de coordonate x=0, y=175):

```
POKE ADR27+2, coordonata x
```

```
POKE ADR27+3, coordonata y
```

★ Pentru rotire, se adaugă următoarele linii (la oricare din programele de mai sus; în loc să realizeze doar deplasarea, vor realiza rotirea pe aceeași direcție):

```

11 MEM   DEFB 0
12 SCR   LD BC,0      ★sau LD BC,44800 pentru rotirea inversă (ce
dispare sus apare jos).
13       CALL 8874
14       LD A,(HL)
15       LD (MEM),A
305 RET  LD A,(MEM)
306       LD (DE),A

```

- LINIA 310 SE SCHIMBĂ ÎN 310 POP DE - (se șterge eticheta)

Efectul de rotire are la bază un algoritm simplu: înainte de a realiza deplasarea coloanei cu o iterație în sus sau în jos, linia care trebuie să dispară din coloană este înmagazinată în memorie, într-un octet pe care îl numim MEM, și pe care îl punem chiar înaintea programului în cod mașină. După ce s-a realizat deplasarea benzii, linia care a dispărut în direcția de mișcare este citită din octetul MEM, unde a fost memorată, și pusă pe ecran în partea opusă a

coloanei.

ATENȚIE !! Programul se compilează cu opțiunea 16 (la întrebarea "Options:" a compilatorului nu se mai tastează ENTER ci numărul 16), pentru a realiza punerea tabelii de simboluri înaintea programului asamblat. Atunci, programul va fi apelat cu RANDOMIZE USR 50651

★ Programul BASIC echivalent este:

```
820 LET ADR27=50650:LET X=58:LET
S=6725:RESTORE 840
830 LET ADR=ADR27:GOSUB 9997
840 DATA 0,1,0,175,205,170,34,126,50,218,197,1,
0,175,197,205,170,34,229,209,36,6,7,126,18,36,
20,16,250,193,197,213,120,222,8,71,56,13,205,
170,34,209,126,18,193,120,214,8,71,24,219,58,
218,197,18,209,209,201
```

★ Avem următoarele opțiuni:

★ la adresele ADR27+2 și ADR27+3 se află coordonatele unui punct din partea de sus a benzii de ecran (aici cel de coordonate x=0, y=175), însă aceste coordonate trebuie să se afle și la adresele ADR27+12 și ADR27+13:

POKE ADR27+2, coordonata x

POKE ADR27+12, coordonata x

POKE ADR27+3, coordonata y

POKE ADR27+13, coordonata y

7.6 Defilarea unui text pe verticală

Dacă la defilarea unui text pe o bandă orizontală din ecran am folosit programul de deplasare orizontală a unei benzi de ecran, aici vom folosi programul scris la capitolul anterior, de deplasare a unei benzi verticale din ecran.

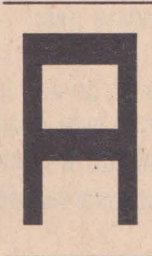
După cum am mai văzut, în memoria calculatorului fiecare caracter are o structură definită pe 8 bytes. Dacă însă la deplasarea orizontală trebuia să calculăm coloana formată de cei 8 biți dintr-o parte a caracterului, aici problema va fi mai simplă, dat fiind că în bandă caracterul nu trebuie transferat prin coloane, ci prin linii, deci practic fiecare octet în parte.

În esență, componentele programului sunt parcurse în ordinea următoare:


- 1) preia caracterul curent din text (linia 60) ;
- 2) caută structura caracterului în setul de caractere (liniile 70-130) ;
- 3) reține unde se află caracterul în memorie. Pentru aceasta se folosește registrul HL (liniile 140-180) ;
- 4) deplasează cu o poziție banda de ecran (programul P1), mutând în același timp linia (octet) * de la adresa din HL în linia + a benzii și trece la următoarea linie a caracterului, adică mărește HL cu 1. (liniile 220-410) ;
- 5) repetă (4) până când a fost copiat întreg caracterul din memorie în bandă (linia 440) ;
- 6) trece la caracterul următor (linia 470) ;
- 7) repetă de la (1) la (6) până când textul a fost transmis în întregime (linia 480);

8) STOP (revenire în BASIC-linia 490).

Pentru o înțelegere mai ușoară, vom reprezenta câțiva pași ai programului. La început, situația este următoarea:

HL →	byte 0	
	byte 1	
	byte 2	
	byte 3	
	byte 4	
	byte 5	
	byte 6	
	byte 7	

Programul deplasează banda de ecran o iterație, mutând permanent octetul care are adresa în HL pe locul rămas liber din bandă, după care mărește HL cu 1, pentru a ajunge la următorul octet.

HL →	byte 0	
	byte 1	
	byte 2	
	byte 3	
	byte 4	
	byte 5	
	byte 6	
	byte 7	

Repetând de 8 ori, se obține copierea completă a caracterului din memorie în banda de ecran.

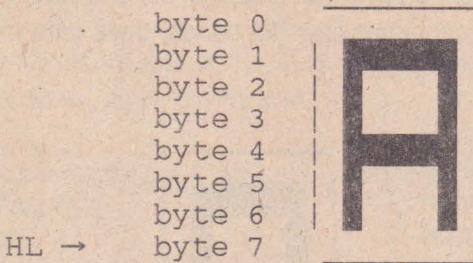
Observație: în exemplul dat, am transmis caracterul de sus în jos, ceea ce înseamnă că partea de sus a caracterului a apărut prima în banda de ecran. Dacă însă acest lucru este însoțit de o mișcare a benzii de sus în jos, vom avea

surpriza să constatăm că litera apare cu capul în jos ! Pare deci evident că ordinea de transmitere a octeților care compun caracterul trebuie coordonată cu mișcarea benzii de ecran, și anume:

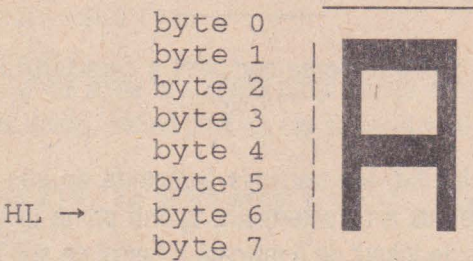
★ pentru mișcarea de jos în sus, ordinea este cea din exemplul anterior, respectiv HL este inițializat la adresa de început a caracterului și este mărit cu 1 la fiecare pas;

★ pentru mișcarea de sus în jos, ordinea va fi inversă, și anume vor fi transmiși mai întâi byte 7, apoi byte 6 și așa mai departe până la byte 0, deci HL va fi inițializat la sfârșitul caracterului și va fi micșorat cu 1 la fiecare pas, așa cum se vede mai jos:

pasul 1



pasul 2



Programul în limbaj de asamblare este următorul:

10	ORG 50750	
20	MEM DEFB, 0	
30	LD DE, 55000	★adresa la care se află textul.
40	LD B, 32	★numărul de caractere al textului.
50	L9 PUSH DE	
60	PUSH BC	★reține în stivă datele de mai sus (adresa textului și numărul de caractere).
70	LD A, (DE)	★încarcă în A caracterul curent.
80	SCAR LD DE, (23606)	★încarcă în DE adresa setului de caractere.
90	LD L, A	
100	LD H, 0	★încarcă HL cu valoarea din A.
110	ADD HL, HL	
120	ADD HL, HL	
130	ADD HL, HL	★înmulțește HL cu 8.
140	ADD HL, DE	★adună adresa setului de caractere; în HL s-a obținut adresa caracterului din A.
150	LD DE, 8	★următoarele două linii sunt necesare dacă trebuie să transferăm caracterul curent de la sfârșit la început (pentru mișcarea de sus în jos). Pentru aceasta, trebuie să îl inițializăm pe HL la adresa de sfârșit a caracterului, adică să adunăm 8 la adresa de început.
160	ADD HL, DE	★efectuează adunarea.
170	LD B, 8	★pentru fiecare caracter repetă transferul a 8 linii (bytes) din memorie în banda de ecran.
180	L10 LD A, (HL)	★încarcă în A byte-ul de la adresa indicată de HL.
190	LD (MEM), A	★păstrează acest byte într-o locație de memorie special definită în acest scop.

200		DEC HL	★fixează HL la adresa anterioară (linia de deasupra celei care a fost transmise). Pentru mișcarea în sens invers, HL va trebui să fie mărit în loc să fie micșorat, deci vom înlocui această instrucțiune cu INC HL.
210		PUSH HL	★urmează programul de mișcare propriu-zisă a benzii, în cursul căreia regiștrii HL și BC pot fi alterați. Pentru a nu pierde însă informațiile importante conținute în ei, îi salvăm în stiva calculatorului.
220		PUSH BC	
230	I1	LD BC, 0	★marginea de jos a benzii de ecran ($x=0, y=0$).
240	L0	PUSH BC	★salvează coordonatele în stivă.
250		CALL 8874	★obține în HL adresa care conține punctul dih BC.
260		PUSH HL	★copiază adresa în DE. ★HL devine pointer la adresa de deasupra celei conținute în DE. ★B conține numărul de repetări pentru deplasarea caracterului pe verticală.
270		POP DE	
280		DEC H	
290		LD B, 7	★B conține numărul de repetări pentru deplasarea caracterului pe verticală.
300	L1	LD A, (HL)	★conținutul adresei din HL trece în DE.
310		LD (DE), A	
320		DEC D	★DE trece la adresa de deasupra.
330		DEC H	★HL trece la adresa de deasupra.
340		DJNZ L1	★repetă pentru tot caracterul.
350		POP BC	★readuce adresele din stivă.
360		PUSH BC	
370		PUSH DE	★înmagazinează ultimul pointer.
380		LD A, B	★poziționează coordonatele la începutul caracterului care urmează.
390		ADD A, 8	
400		LD B, A	

410	CP 175	★verifică dacă s-a atins marginea superioară a benzii (aici, aceasta este și marginea superioară a ecranului).
420	JR NC, RET	★dacă da, atunci sare la subrutina care realizează eliberarea stivei și întoarcerea în programul apelant.
430	CALL 8874	★dacă nu, atunci calculează adresa de început a următorului caracter.
440	POP DE	★reface pointerul anterior.
450	LD A, (HL)	
460	LD (DE), A	★transferă ultima linie a caracterului care urmează să fie deplasat în prima linie a caracterului care a fost deplasat anterior.
470	POP BC	★reface coordonatele.
480	LD A, B	
490	ADD A, 8	
500	LD B, A	★modifică coordonatele pentru trecerea la caracterul următor.
510	JR L0	★sare la început.
520	RET LD A, (MEM)	★deplasarea benzii a fost terminată, deci putem transfera la capătul rămas liber linia de caracter reținută în memorie. Transferul se face prin registrul A.
530	LD (DE), A	★octetul din A este transferat la capătul benzii (al cărui adresă se află în DE).
540	POP DE	
550	POP DE	★eliberează stiva de cei doi regiștri pereche care au rămas acolo din programul de mișcare a benzii.
560	CALL 654	★cheamă subrutina de citire a tastaturii. Programul este realizat astfel încât să permită ieșirea în BASIC la apăsarea oricărei taste de pe claviatură.

570	LD A, E	
580	CP 255	★subrutina de citire a tastaturii se află în memoria ROM și este concepută astfel încât să întoarcă valoarea 255 în registrul E dacă nu a fost apăsată nici o tastă. Verificăm această posibilitate comparând registrul E cu 255.
590	JR NZ, RR	★dacă este diferit (deci a fost apăsată o tastă) atunci sare la subrutina de revenire în BASIC.
600	POP BC	
610	POP HL	★reface, din stiva calculatorului, valorile regiștrilor HL și DE, salvați înainte de a deplasa banda de ecran (BC conține numărul de linii rămase de transmis din caracterul curent pe ecran, iar HL reprezintă adresa liniei la care s-a ajuns).
620	HALT	★această instrucțiune poate fi înlocuită cu NOP, în care caz procesul va decurge mult mai rapid (mișcarea va fi accelerată). Această linie este pusă pentru a oferi posibilitatea de schimbare a vitezei cu care apare textul pe ecran.
630	DJNZ L10	★reia de la eticheta L10 pentru următoarea linie a caracterului (deplasează banda de ecran și transmite în spațiul rămas liber această linie) până când toate liniile au fost transmise.
640	POP BC	
650	POP DE	★reface din stiva calculatorului regiștrii BC și DE, care au fost salvați înainte de a se începe tratarea caracterului curent din text. BC conține numărul de caractere care au

mai rămas de transmis pe ecran, iar DE conține adresa caracterului care tocmai a fost scris.

660 INC DE

★mărește DE cu 1, pentru a conține adresa următorului caracter ce trebuie transmis pe ecran.

670 DJNZ L9

★repetă de la eticheta L9 până când au fost transmise toate caracterele.

690 RET

★când a fost terminat tot textul, se întoarce în interpretorul BASIC.

700 RR POP BC

★această subrutină realizează întoarcerea în BASIC în cazul în care a fost apăsată o tastă, deci programul nu a fost lăsat să se deruleze până la capăt. Mai întâi se eliberează stivă de cei 4 regiștri pereche care au rămas acolo în urmă întreruperii forțate a programului, deoarece nu se poate realiza întoarcerea în BASIC decât cu stiva calculatorului "curată".

710 POP BC

720 POP BC

730 POP BC

740 RET

★ Programul este apelat cu

RANDOMIZE USR 50751

★ Program BASIC:

850 LET ADR28=50750:LET X=110:LET
S=13104:RESTORE 870

860 LET ADR=ADR28:GOSUB 9997

870 DATA 0,17,216,214,6,32,213,197,26,237,91,

54,92,111,38,0,237,106,237,106,237,106,237,90,
 17,8,0,237,90,6,8,126,50,62,198,43,229,197,1,
 0,0,197,205,170,34,229,209,37,6,7,126,18,21,
 37,16,250,193,197,213,120,198,8,71,254,175,
 48,13,205,170,34,209,126,18,193,120,198,8,71,
 24,217,58,62,198,18,209,209,205,142,2,123,254,
 255,32,11,193,225,118,16,188,193,209,19,16,
 158,201,193,193,193,193,201

★ Textul este pus în memorie cu ajutorul unui program de forma:

```
LET A$=" . . . " (textul care trebuie să apară pe ecran)
FOR N=1 TO LEN A$
POKE 55000+N-1, CODE A$(N) (textul este pus la adresa
                           55000)
NEXT N
```

★ Modificări posibile:

★ locațiile 50752 și 50753 conțin adresa textului în memorie (aici 55000).

★ locația 50755 conține numărul de caractere al textului ce urmează să fie scris.

★ locația 50846 conține 118 pentru mișcare lentă și 0 pentru mișcare rapidă a benzii de ecran pe care se derulează textul.

★ locațiile 50789 și 50790 inițializează coordonatele benzii de ecran, coordonate date printr-un punct, ca la programul de mișcare simplă a benzii pe ecran (X, apoi Y, care trebuie să fie divizibil cu 8).

★ Programul BASIC pentru mișcarea de jos în sus este:

```

880 LET ADR29=50900:LET X=103:LET
S=12728:RESTORE 900
890 LET ADR=ADR29:GOSUB 9997
900 DATA 0,17,216,214,6,32,213,197,26,237,91,54,
92,111,38,0,237,106,237,106,237,106,237,90,6,8,
126,50,212,198,35,229,197,1,0,175,197,205,170,34,
229,209,36,6,7,126,18,20,36,16,250,193,197,
213,120,222,8,71,56,13,205,170,34,209,126,18,
193,120,214,8,71,24,219,58,212,198,18,209,209,
205,142,2,123,254,255,32,11,193,225,0,16,190,
193,209,19,16,165,201,193,193,193,193,201

```

★ Programul se apelează cu

RANDOMIZE USR 50901

★ Textul se inițializează cu același program ca mai sus.

★ Modificări posibile:

★ locațiile 50902 și 50903 conțin adresa textului în memorie (aici 55000).

★ locația 50905 conține numărul de caractere al textului ce urmează să fie scris.

★ locația 50989 conține 118 pentru mișcare lentă și 0 pentru mișcare rapidă a benzii de ecran pe care se derulează textul.

★ locațiile 50934 și 50935 inițializează coordonatele benzii de ecran, coordonate date printr-un punct, ca la programul de mișcare simplă a benzii pe ecran (X, apoi Y, care trebuie să fie divizibil cu 8, cu deosebirea față de programul anterior că aici ele trebuie să reprezinte un punct de deasupra benzii, nu de dedesubtul ei).

7.7 Mișcarea unui cursor pe ecran

Acesta este un efect folosit în multe programe, atât în jocuri, precum și în utilitare (de exemplu ARTSTUDIO). Programul permite definirea unei forme pixel cu pixel și mișcarea ei pe ecran în concordanță cu tastele apășate. În exemplul de mai jos, programul va lua în considerație tastele "5", "6", "7", "8" (cursorul) pentru deplasare și tasta "0" pentru revenire în BASIC.

Sucesiunea pașilor în program este următoarea:

★ desenează cursorul pe ecran, la poziția inițială. Coordonatele punctelor din care este format cursorul se găsesc permanent la adresa 60000, iar numărul lor este în registrul B. Prin "desenează" înțelegem desenarea cu OVER (dacă este desenat dinainte, îl șterge, iar dacă nu, îl desenează);

★ citește tastatura;

★ dacă nu a fost apășată nici o tastă, reia de la b;

★ dacă tasta apășată este "0" atunci trimite la rutina de ieșire în BASIC;

★ dacă tasta apășată a fost "5", "6", "7", "8" trimite respectiv la rutinele de tratare a mișcării la stânga, în jos, în sus și la dreapta;

★ reia de la pasul 2;

Rutinele de mișcare sunt construite toate pe același model:

★ șterge cursorul de la poziția curentă;

★ citește coordonatele punctelor din care este format cursorul;

★ mărește sau micșorează coordonata corespunzătoare mișcării pe care o

tratează (de exemplu, pentru sus mărește coordonata y a tuturor punctelor);

★ dacă se depășește limita ferestrei, atunci inițializează coordonata în partea opusă;

★ repetă pentru toate punctele;

★ sare la primul punct din programul principal;

Rutina de ieșire în BASIC:

★ șterge cursorul de la poziția curentă;

★ eliberează stiva calculatorului;

★ iese în BASIC.

Programul în limbaj de asamblare este următorul:

10	ORG 50000	★adresa unde este asamblat programul (atenție, nu este relocabil, așa că aceasta trebuie să fie adresa definitivă).
20	LD HL, 23697	★adresa atributelor pe care interpretorul BASIC le ia în seamă pentru a efectua instrucțiunea PLOT.
30	SET 0, (HL)	★poziționează atributul "OVER". De acum înainte, toate punctele vor fi puse complementate cu cele de pe ecran (vor fi stinse dacă sunt aprinse și aprinse dacă sunt stinse). Poate să fie poziționat orice alt bit din octetul de atribute, respectiv INVERSE, FLASH, etc., sau poate fi lăsat fără atribute.

40	LD HL, 60000	★adresa la care se află memorate coordonatele punctelor din care este format cursorul. în caz că există mai multe, pot fi schimbate prin schimbarea acestei adrese.
50	LD B, 20	★numărul de puncte din care este format cursorul.
60	CALL PLT	★desenează cursorul pe ecran în poziția inițială, chiar dacă nu s-a apăsat nici o tastă.
70	LD HL, 60000	★reinițializează adresa cursorului.
80	LD B, 20	★lungimea cursorului.
90	SCAN PUSH HL	★păstrează adresa.
100	PUSH BC	★păstrează lungimea.
110	CALL 654	★cheamă subrutina de citire a tastaturii.
120	POP BC	
130	POP HL	★reface lungimea și adresa.
140	LD A, E	★încarcă în A codul tastei apăstate.
150	CP 255	★verifică dacă s-a apăsat vreo tastă.
160	JR Z, SCAN	★dacă nu, atunci reia citirea tastaturii.
170	PUSH AF	
180	PUSH DE	
190	PUSH BC	★păstrează regiștrii.
200	NOP	★acest octet este păstrat pentru o opțiune particulară, aceea de a lăsa urmă pe ecran (în care caz , este înlocuit cu instrucțiunea DEC B).
210	CALL PLT	★șterge cursorul de pe poziția inițială (era aprins, deci prin scrierea peste, cu OVER, a fost șters).
220	POP BC	
230	POP DE	
240	POP AF	★reface regiștrii.

250	PUSH BC	
260	PUSH HL	★păstrează adresa și lungimea.
270	CP 3	★verifică dacă tasta apăsată a fost "6".
280	JR Z, JOS	★dacă da, atunci sare la subrutina de deplasare în jos a cursorului.
290	CP 11	★verifică dacă a fost apăsată tasta "7".
300	JR Z, SUS	★dacă da, atunci sare la subrutina de deplasare în sus a cursorului.
310	CP 19	
320	JR Z, DRE	
330	CP 4	
340	JR Z, STI	★similar pentru tastele "5" și "8" (deplasare la stânga, respectiv la dreapta a cursorului).
350	CP 35	★verifică dacă a fost apăsată tasta "0".
360	POP HL	
370	POP BC	★reface stiva.
380	RET Z	★dacă a fost apăsat "0", atunci se reîntoarce în BASIC.
390	JR SCAN	★reia de la citirea tastaturii.
400	DRE LD A, (HL)	★această subrutină realizează modificarea coordonatelor punctelor care formează cursorul pentru deplasarea la dreapta. Mai întâi, încarcă în A coordonata X a punctului curent,
410	CP 255	o compară cu marginea din dreapta a ferestrei (aici 255, dar poate fi schimbată).
420	CALL Z, ZE1	★dacă a ajuns la margine, atunci rezoziționeaza la marginea opusă.
430	INC (HL)	★mărește coordonata X a punctului.
440	INC HL	
450	INC HL	★trece la coordonata X a punctului următor.
460	DJNZ DRE	★repetă pentru toate punctele din cursor.
470	POP HL	

480		POP BC	★reface adresa și lungimea.
490		CALL PLT	★desenează cursorul la noile coordonate.
500		JR L0	★reia de la L0.
510	SUS	INC HL	★aceasta subrutină deplasează în stânga cursorul. Pentru aceasta trece întâi la coordonata Y a punctului curent.
520		LD A, (HL)	★încarcă coordonata Y în A.
530		CP 174	★o compară cu marginea de sus a ferestrei.
540		CALL Z, ZEO	★dacă a atins-o, atunci readuce la 0.
550		INC (HL)	★mărește coordonata Y.
560		INC HL	★trece la punctul următor.
570		DJNZ SUS	★repetă pentru toate punctele.
580		POP HL	
590		POP BC	★reface adresa și lungimea.
600		CALL PLT	★desenează cursorul.
610		JR L0	★reia de la L0.
620	JOS	INC HL	★subrutina de deplasare în jos.
630		LD A, (HL)	
640		CP 1	
650		CALL Z, S175	
660		DEC (HL)	
670		INC HL	
680		DJNZ JOS	
690		POP HL	
700		POP BC	
710		CALL PLT	
720		JR L0	
730	STI	LD A, (HL)	★subrutina de deplasare la stânga.
740		CP 1	
750		CALL Z, S255	
760		DEC (HL)	
770		INC HL	
780		INC HL	
790		DJNZ STI	
800		POP HL	

810	POP BC		
820	CALL PLT		
830	JR L0		
840	ZE0 LD (HL), 0	★	subrutina de repunere a coordonatei Y la marginea de jos a ferestrei.
850	RET	★	întoarcere în punctul din care s-a făcut apelarea.
860	ZE1 LD (HL), 0	★	subrutina de repunere a coordonatei X la marginea din stânga a ferestrei.
870	RET	★	întoarcere la punctul din care a fost apelată.
880	S175 LD (HL), 175	★	poziționarea coordonatei Y la marginea de sus a ferestrei.
890	RET	★	întoarcere.
900	S255 LD (HL), 255	★	marginea din dreapta a ferestrei.
910	RET	★	întoarcere.
920	PLT PUSH HL	★	această subrutină desenează cursorul la coordonatele memorate începând la adresa din HL. întâi salvează adresa.
930	LPLT PUSH BC	★	păstrează coordonatele.
940	LD C, (HL)	★	încarcă coordonata X în C.
950	INC HL	★	HL conține acum adresa coordonatei Y a punctului curent.
960	LD B, (HL)	★	încarcă coordonata Y în B.
970	INC HL	★	trece la punctul următor.
980	PUSH HL	★	ii păstrează adresa.
990	CALL 8933	★	pune punctul pe ecran.
1000	POP HL	★	reface adresa punctului.
1010	POP BC	★	reface B, care conține numărul de puncte rămase.
1020	DJNZ LPLT	★	repetă pentru toate punctele.
1030	POP HL	★	reface adresa primului punct al cursorului.
1040	RET	★	reîntoarcere de unde a fost apelat.

★ Programul în cod mașină doar preia datele despre cursor, deci acestea trebuie puse dinainte în memorie, cu un program de forma:

```
LET ADC=60000 (adresa la care este pus în memorie)
FOR N=X1 TO X2:FOR G=Y1 TO Y2
IF POINT (N,G)=1 THEN POKE ADC,N:POKE
ADC+1,G:LET ADC=ADC+2
NEXT G:NEXT N
```

În acest program, am presupus că desenul cursorului se află între coordonatele X1 și X2, respectiv Y1 și Y2 (vor fi inițializate înainte de apelarea acestui program). Numărul de puncte ce formează cursorul se află cu $PRINT (ADC - 60000) / 2$, și trebuie să fie maxim 255. Odată pus în memorie în felul acesta, cursorul poate fi salvat ca bloc de cod, fie introdus cu o linie de DATA.

Codurile tastelor citite pot fi schimbate conform tabelului de coduri prezentat la capitolul 3.6 - "Rutine din memoria ROM".

★ Program BASIC:

```
910 LET ADR30=50000:LET X=170:LET
S=24459:RESTORE 920
920 LET ADR=ADR30:GOSUB 9997
930 DATA 33,145,92,203,198,33,96,234,6,20,205,
167,195,33,96,234,6,20,229,197,205,142,2,193,
225,123,254,255,40,244,245,213,197,5,205,167,
195,193,209,241,197,229,254,3,40,93,254,11,40,
71,254,19,40,15,254,4,40,45,254,35,40,4,225,193,
24,208,209,209,201,126,254,255,204,238,195,52,
35,35,16,245,225,193,205,167,195,24,182,229,
197,78,35,70,35,229,205,229,34,225,193,16,243,
225,201,126,254,0,204,244,195,53,35,35,16,245,
225,193,205,167,195,24,148,35,126,254,174,204,
247,195,52,35,16,245,225,193,205,167,195,24,130,
35,126,254,51,204,241,195,53,35,16,245,225,193,
```

205, 167, 195, 195, 93, 195, 54, 0, 201, 54, 175, 201, 54,
255, 201, 54, 50, 201

★ Modificări posibile:

POKE 50033, 0 - cursorul nu lasă urmă;

POKE 50043, codul tastei la apăsarea căreia cursorul să se deplaseze în jos (aici 3, pentru tasta "6");

POKE 50047, codul tastei la apăsarea căreia cursorul să se deplaseze în sus (aici 11, pentru tasta "7");

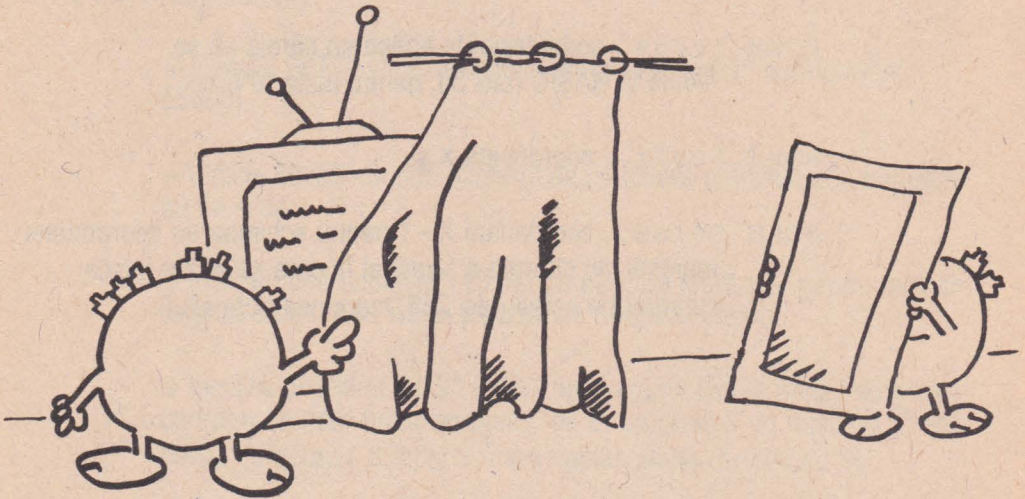
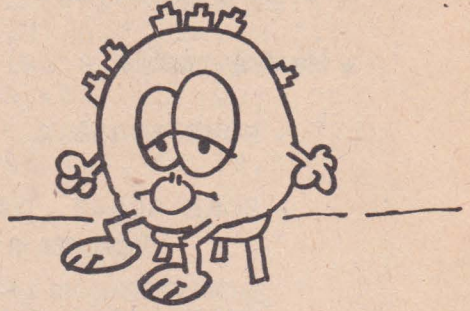
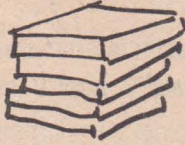
POKE 50051, codul tastei la apăsarea căreia cursorul să se deplaseze în stânga (aici 19, pentru tasta "5");

POKE 50055, codul tastei la apăsarea căreia cursorul să se deplaseze în dreapta (aici 4, pentru tasta "8");

POKE 50059, codul tastei la apăsarea căreia să se revină în BASIC (aici 35, pentru tasta "0").

POKE 50071, coordonată X și

POKE 50165, coordonată X - 1 pentru schimbarea coordonatei marginii din dreapta a ferestrei în care se poate mișca cursorul pe ecran (aici 255, marginea ecranului).



ANEXA A

Instrucțiunile microprocesorului Z80

În această anexă vom prezenta lista completă, dată de proiectanți, a mnemonicelor instrucțiunilor în cod mașină recunoscute de Z80.

Vom folosi următoarele notații:

- **N** = un număr cuprins între 0 și 255 inclusiv
- **NN** = un număr cuprins între 0 și 65535 inclusiv
- **X** = orice registru dintre { A,B,C,D,E,H,L }
- **Y** = orice locație dintre { (HL), (IX+n), (IY+n) }
- **Z** = orice registru dintre { BC, DE, HL, SP }
- **cond** = orice condiție dată de indicatori (C,M,NC,NZ,P,PE,PO,Z)
- **dis** = un număr cuprins între -127 și 127 , reprezentând deplasamentul (distanța, în octeți, la care de face saltul).

Atenție ! A nu se uita diferența dintre HL și (HL) : primul reprezintă registrul, care poate conține un număr sau o adresă în intervalul 0-65535, iar al doilea este conținutul acelei adrese, adică numărul care se află memorat în byte-ul de la adresa din HL. La fel, este o diferență între NN și (NN). Primul este o adresă, al doilea este conținutul acelei adrese.

ADC A, Y ; ADC A, N ; ADC A, X

★ adună la conținutul acumulatorului numărul specificat de al doilea operand (registru, conținut de adresă sau număr). Dacă rezultatul depășește 255, flag-ul C (carry) este poziționat și în acumulator se depune rezultatul - 256.

ADC HL, Z

★ similar ca mai sus pentru adunări de numere mai mari decât 255. Aici se folosesc regiștrii dubli.

ADD A, Y ; ADD A, N ; ADD A, X ;
ADD HL, Z ; ADD IX, Z ; ADD IY, Z

★ la fel ca mai sus, numai că indicatorul C nu mai este poziționat în cazul depășirii.

AND Y ; AND X ; AND N

★ execută operația ȘI logic între conținutul acumulatorului și al doilea operand (registru, locație adresată de un registru sau număr) Rezultatul este întors în acumulator.

BIT 0,Y ; BIT 0,X ; BIT 1,Y ; BIT 1,X ; BIT 2,Y ; BIT 2,X ; BIT 3,Y ; BIT 3,X ;
BIT 4,Y ; BIT 4,X ; BIT 5,Y ; BIT 5,X ; BIT 6,Y ; BIT 6,X ; BIT 7,Y ; BIT 7,X

★ poziționează indicatorul Z dacă bitul respectiv din cel de-al doilea operand este stins (0).

CALL NN

★ cheamă subrutina de la adresa NN.

CALL cond, NN

- ★ cheamă subrutina de la adresa NN dacă indicatorii satisfac condiția cond.

CCF

- ★ completează indicatorul carry (dacă e 0 îl face 1 și invers).

CP X ; CP Y ; CP N

- ★ compară conținutul operandului cu acumulatorul și poziționează indicatorii în funcție de rezultat.

CPD

- ★ compară conținutul adresei (DE) cu al adresei (HL) și micșorează DE și HL.

CPDR

- ★ același afect, numai că repetă de câte ori arată registrul BC.

CPI ; CPIR

- ★ la fel, numai că mărește conținutul regiștrilor în loc să-l micșoreze.

DAA

- ★ folosește la operațiile aritmetice în sistem zecimal.

CPL

- ★ completează conținutul acumulatorului.

DEC X ; DEC Y ; DEC IX ; DEC IY ; DEC Z

★ micșorează conținutul operandului cu 1.

DI

★ folosește pentru sistemul de întreruperi (dezactivare).

DJNZ dis

★ micșorează B cu 1 și repetă de la deplasamentul dis până când B este 0.

EI

★ folosește pentru sistemul de întreruperi (activare).

EX (SP), HL ; EX (SP), IX ; EX (SP), IY ; EX DE, HL

★ schimbă operanzii între ei.

EX AF,AF' ; EX BC,BC' ; EX DE,DE' ; EX HL,HL' ; EXX

★ schimbă conținuturile regiștrilor activi cu cei de rezervă.

HALT ; IM 0 ; IM 1 ; IM 2

★ folosesc pentru sistemul de întreruperi. HALT poate fi folosit și pentru a crea o întârziere de 20 ms.

IN A, N ; IN X, (C)

★ încarcă de la portul reprezentat de al doilea operand un octet în primul operand.

INC X ; INC Y ; INC Z ; INC IX ; INC IY

★ mărește valoarea operandului cu 1.

IND ; INDR ; INI ; INIR

★ instrucțiuni pentru preluări repetate de octeți de la porturi.

JP (HL) ; JP (IX) ; JP (IY)

★ salturi lungi la adresele reprezentate de conținutul unor regiștri pereche.

JP NN

★ salt necondiționat la adresa NN.

JP cond, NN

★ salturi lungi condiționate.

JR dis

★ salt scurt (la o adresă aflată la o distanță cuprinsă între -127..127 octeți față de adresa instrucțiunii de salt).

JR cond, dis

★ salt scurt condiționat.

LD (NN), A ; LD (NN), Z ; LD (NN), IX ; LD (NN), IY

★ încărcare a locației sau cuvântului de la adresa NN cu valoarea specificată de al doilea operand (registru simplu sau pereche).

LD (BC), A ; LD (DE), A ; LD Y, A ; LD (HL), N ; LD (HL), X

★ încarcă la locația specificată de valoarea primului operand valoarea celui de al doilea.

LD A, (BC) ; LD A, (DE) ; LD X, Y ; LD X, X ; LD A, I ; LD A, R ; LD R, A ; LD I, A

★ transferuri de valori între regiștri.

LD X, N ; LD BC, NN ; LD BC, (NN) ; LD DE, (NN) ; LD DE, NN ; LD HL, NN
LD HL, (NN) ; LD IX, NN ; LD IX, (NN) ; LD IY, NN ; LD IY, (NN)

★ transfer de valori (specificate sau din memorie) în regiștri.

LD SP, HL ; LD SP, IX ; LD SP, IY

★ transferuri în pointerul de stivă.

LDD ; LDDR ; LDI ; LDIR

★ încărcări ale conținutului adresei (DE) cu conținutul adresei (HL), cu mărirea sau micșorarea regiștrilor DE și HL și cu sau fără repetare de BC ori.

NEG

★ neagă valoarea lui A.

NOP

★ anunță procesorul să nu execute nici o operație timp de un tact.

OR Y ; OR X ; OR N

★ execută operația SAU logic între acumulator și operand.

OUT (C), X ; OUT (N), A

★ transmite la portul specificat de primul operand valoarea din cel de-al doilea.

OUTD ; OUTI

★ transmisii cu modificarea regiștrilor DE și HL (micșorare, respectiv mărire).

OTDR ; OTIR

★ transmisii repetate (Out, Increment/Decrement and Repeat).

POP Z ; POP IX ; POP IY ; POP AF

★ extrage valori din stivă și le pune în regiștrii specificați.

PUSH Z ; PUSH IX ; PUSH IY ; PUSH AF

★ depune valorile regiștrilor specificați în stivă.

RES 0, X ; RES 0, Y ; RES 1, X ; RES 1, Y ; RES 2, X ; RES 2, Y ; RES 3, X ;
RES 3, Y ; RES 4, X ; RES 4, Y ; RES 5, X ; RES 5, Y ; RES 6, X ; RES 6, Y ;
RES 7, X ; RES 7, Y

★ stinge bitul respectiv din registrul specificat ca al doilea operand.

RET

★ salt la adresa găsită în stivă (ân modul cel mai general, adresa de după ultima instrucțiune de CALL).

RET cond

★ la fel, condiționat.

RL X ; RL Y

★ rotește la stânga conținutul operandului.

RLC X ; RLC Y

★ la fel, dar prin indicatorul carry (care este considerat acum ca un al 9-lea bit).

RR X ; RR Y

★ rotire la dreapta.

RRC X ; RRC Y

★ la fel, dar prin carry.

SBC A, Y ; SBC A, X ; SBC A, N

★ scade valoarea celui de al doilea operand din acumulator și poziționează carry dacă rezultatul este negativ.

SBC HL, Z

★ scade numere mai mari decât 255, poziționând indicatorii.

SCF

★ poziționează carry.

SET 0, X ; SET 0, Y ; SET 1, X ; SET 1, Y ; SET 2, X ; SET 2, Y ; SET 3, X ;
SET 3, Y ; SET 4, X ; SET 4, Y ; SET 5, X ; SET 5, Y ; SET 6, X ; SET 6, Y ;
SET 7, X ; SET 7, Y

★ aprinde bitul respectiv din registrul specificat ca al doilea operand.

SLA X ; SLA Y

★ deplasează conținutul operandului la stânga.

SRA X ; SRA Y

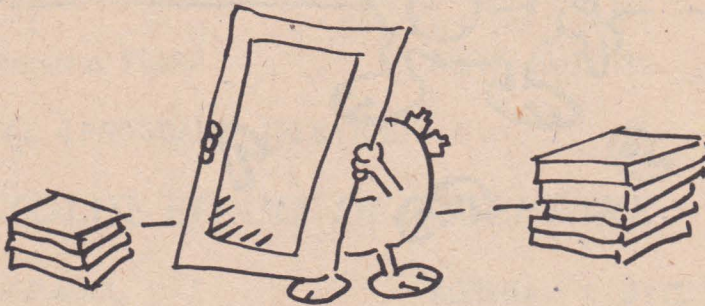
★ la fel, spre dreapta.

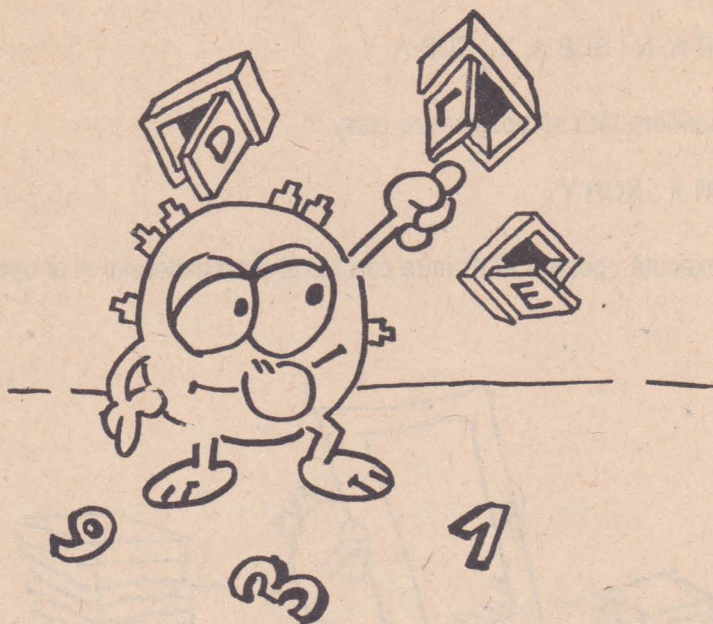
SUB A, N ; SUB A, X ; SUB A, Y

★ scădere fără să poziționeze carry.

XOR X ; XOR Y

★ execută operația XOR între conținutul acumulatorului și al operandului.





ANEXA B

Lista programelor

În paginile următoare vom prezenta lista programelor care au fost comentate în această carte.

★ *CAPITOLUL EFECTE DE CORTINĂ*

PROGRAMUL	ADRESA	BYTES	SUMA
★ Acoperirea ecranului de sus în jos	ADR1 - 49000	25	2592
★ Acoperirea ecranului de jos în sus	ADR2 - 49025	25	2817
★ Acoperirea ecranului stânga - dreapta	ADR3 - 49050	25	2336
★ Stingere cu FLASH	ADR4 - 49075	31	2446

TOTAL: 4 PROGRAME ; 106 BYTES

★ *CAPITOLUL EFECTE DE AȘTEPTARE*

PROGRAMUL	ADRESA	BYTES	SUMA
★ FLASH colorat	ADR5-49150	30	2507

PROGRAMUL	ADRESA	BYTES	SUMA
★ Efecte pe BORDER	ADR6-49180	18	2417
variantă cu întoarcere la tastare	ADR7-49200	28	-

TOTAL: 3 PROGRAME ; 76 BYTES

★ *LUCRUL CU IMAGINI*

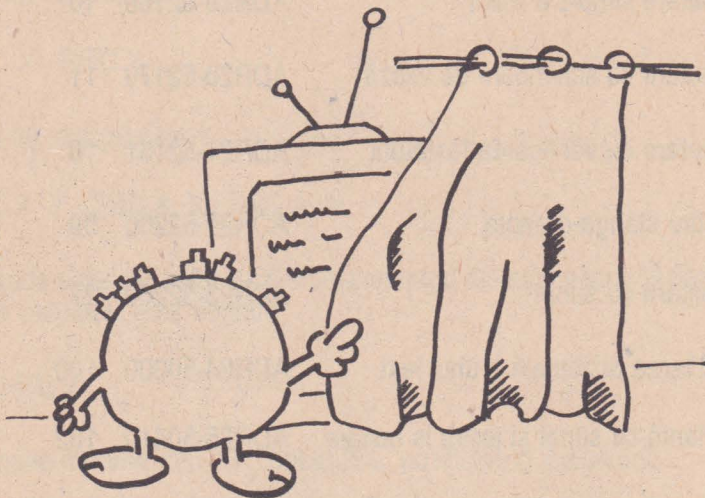
PROGRAMUL	ADRESA	BYTES	SUMA
★ Compactare ecran	ADR8-49300	99	3846
★ Refacere ecran compactat	ADR9-49400	31	2214
★ Memorare fereastră	ADR10-49440	35	3719
★ Desenare fereastră memorată	ADR11-49480	35	3719
★ Negare fereastră	ADR12-49520	31	3493
★ Stergere fereastră de pe ecran	ADR13-49555	27	2550

TOTAL: 6 PROGRAME ; 258 BYTES

★ MIȘCARE PE ECRAN

PROGRAMUL	ADRESA	BYTES	SUMA
★ Deplasare de la stânga la dreapta	ADR14-52000	36	4000
repetare simplă a deplasării	ADR15-52040	10	-
★ Deplasare de la dreapta la stânga	ADR16-52050	38	4436
repetare simplă a deplasării	ADR17-52090	10	-
★ Rotire dreapta-stânga	ADR18-52100	55	6706
repetare simplă a rotirii	ADR19-52160	10	-
repetare cu schimbare de viteză	ADR20-52170	11	-
repetare cu verificarea tastaturii	ADR21-52181	18	-
★ Rotire stânga-dreapta	ADR22-52200	59	7324
repetare cu sunet	ADR23-52260	27	-
★ Defilarea orizontală a unui text	ADR24-50300	100	11776
variantă cu sunet și ieșire la tastare	ADR25-50410	102	11934
★ Deplasare verticală de jos în sus	ADR26-50600	45	5265
★ Rotire jos-sus-jos	ADR27-50650	58	6725
★ Defilarea de sus în jos a unui text	ADR28-50750	110	13104

PROGRAMUL	ADRESA	BYTES	SUMA
★ Defilarea de jos în sus a unui text	ADR29-50900	103	12728
★ Mișcarea unui cursor pe ecran	ADR30-50000	170	24459



BIBLIOGRAFIE

Steven Vickers

ZX Spectrum, BASIC programming
Sinclair Research Ltd., 1982

Jan Logan, Frank O'Hara

The Complete Spectrum ROM Disassembly
Melbourne House Publisher, 1983



FABRICA DE CALCULATOARE
ELECTRONICE S.A.

ICE FELIX S.A.

78009 BUCUREȘTI ROMÂNIA
str. Ing. G. Constantinescu 2, sector 2

Tel: 688.38.00 688.26.89 688.38.40 688.22.95
Fax: 312.87.50 687.62.20 Telex: 11 626 felix r

Bucurându-se de o experiență în
domeniul tehnicii de calcul de peste 20 de ani,

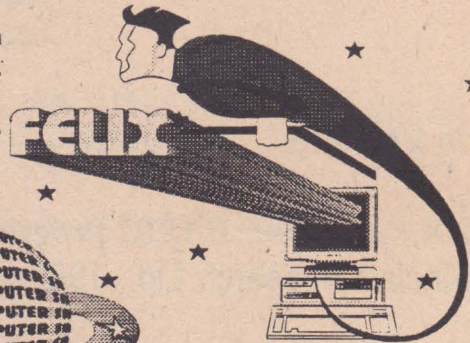
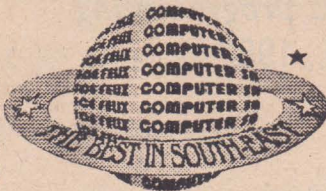
ICE FELIX COMPUTER S.A.

asigură în orice relație, cu orice partener,
garanția unei afaceri sigure și eficiente

ICE FELIX COMPUTER S.A.

o marcă pe care trebuie s-o **FOLOSITI!**

În consens cu
cererea pieței, **FELIX**
COMPUTER S.A. vă
recomandă câteva din
produsele sale:



CALCULATOARE:

- Microcalculatoare compatibile IBM, rețele de calculatoare
- Calculatoare de birou cu afișaj și imprimantă
- Calculator de proces SPOT, software aferent
- Familia de minicalculatoare CORAL

PERIFERICE:

- Monitor monocrom VGA
- Imprimantă matricială
- Imprimantă laser

SISTEME DE SECURITATE:

SISTEME DE RECEPȚIE TV SATELIT

PROIECTARE ȘI REALIZARE DE MATRIȚE

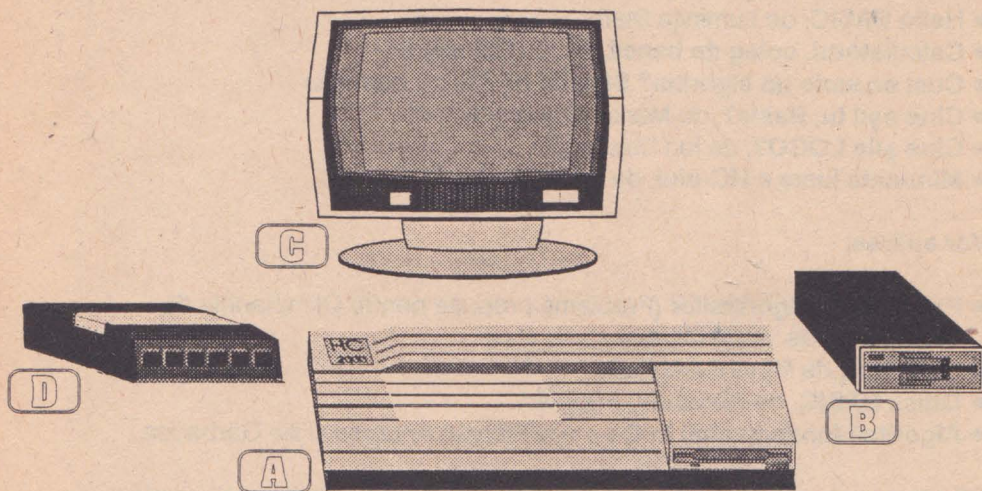
PROIECTARE ȘI REALIZARE DE CIRCUITE IMPRIMATE

APLICAȚII SOFTWARE

ALTE PRODUSE ELECTRONICE DE UZ SPECIAL ȘI GENERAL

Pentru toate produsele ICE FELIX COMPUTER S.A. se asigură garanție, service, piese de schimb, intervenții cu personal specializat.

HC 2000



Caracteristici:

A Unitate centrală HC2000

- microprocesor Z80A/3,5MHz
- memorie RAM 64K din care 48K disponibili în mod BASIC SINCLAIR, respectiv 56K disponibili în mod CP/M.
- memorie EPROM 48K din care 16K pentru interpretorul BASIC SINCLAIR, 16K pentru funcțiile BIOS CP/M și 16K pentru funcțiile INTERFACE1 (disc, interfață serială, interfață rețea).
- tastatură extinsă de 50 de taste care include și tastele speciale CP/M (CTRL, ESC, săgeți, TAB, LARGE BLANK etc.)
- afișare pe televizor alb-negru/color PAL sau pe monitor RGB monocrom/color; rezoluție grafică: 256x192 pixeli; rezoluție alfanumerică: 32 coloane X 24 linii (mod BASIC) și 64 coloane X 24 linii (mod CP/M)
- unitate de disc flexibil de 3,5"DD încorporată cu capacitate de 720K (CP/M) sau 640K (BASIC)

B Unitate floppy-disc externă (opțional)

- unitate tip 5,25"DD cu capacitate de 720K (CP/M) sau 640K (BASIC)
- alimentare din calculator

C Monitor RGB monocrom (opțional)

- intrare de tip RGB/TTL (mufă 9 pini)
- diagonala 31 cm (12")
- luminofor verde

D Casetofon de date (opțional)

- viteza de deplasare a benzii: 4,75 cm/s
- intrare/ieșire date prin mufă DIN standard de 5 pini
- alimentare la priza de 220V/50Hz

Toate produsele prezentate mai sus pot fi achiziționate atât de la distribuitorii ICE FELIX din țară, cât și de la magazinul societății situat la sediul acesteia (program: zilnic 8:00 - 17:00). Pentru oricare din produsele achiziționate se asigură service în garanție (13 luni de la data achiziționării) și post-garanție.

Editura AGNI - Tel: 615.55.59/633.45.31 Fax:312.93.33

În seria "Biblioteca de informatică" destinată elevilor au apărut:

- **Cum să realizăm jocuri pe calculator**, de Ion Diamandi
- **Hello BASIC**; de Luminița State
- **Calculatorul, coleg de bancă**, de Ion Diamandi
- **Cum se scrie un algoritm? Simplu**, de Adrian Atanasiu
- **Cine ești tu, Basic?**, de Marian Gheorghe
- **Cine știe LOGO?**, de Ion Diamandi
- **Minunata lume a HC-ului**, de Vlad Atanasiu

Vor apărea:

- **Provocarea algoritmilor** (Probleme propuse pentru Olimpiadele de informatică), de Victor Mitrana
- **CLIPPER**, de Mihai Cerchizan
- **Quick BASIC**, de Alexandru Popovici
- **Algoritmi fundamentali in C++**, de Răzvan Andonie și Ilie Gârbacea

Cărțile noastre se pot procura și prin sistemul "Cartea prin poșta" cu plata ramburs (la primirea coletului). Pentru aceasta este suficientă trimiterea unei scrisori simple după modelul de mai jos:

Numele _____ Localitatea _____

Str _____ Nr _____ Bl _____ Ap _____ Judet _____ Cod _____

Vă rog să-mi expediți prin colet poștal cu ramburs cartea
nr. exemplare _____

Semnătura,

Adresa noastră poștală: Editura AGNI CP 30-107, BUCUREȘTI

Pentru difuzarea cărților noastre în școli, cluburi ale copiilor, cercuri de informatică etc., Editura AGNI oferă reduceri de prețuri. Astfel, pentru comenzi între 5 și 20 exemplare, reducerea va fi de 10%. Pentru comenzi de peste 20 exemplare, reducerea va fi de 15%. Cheltuielile de expediție vor fi suportate de Editura AGNI.

MINUNATA LUME A HC-ULUI



este o excelentă introducere în lumea programării în cod mașină, ajutînd la o mai bună înțelegere a calculatorului, a legăturilor dintre hardware și software, dintre “ce se cere” și “cum se face”. Cartea prezintă o serie de efecte de calitate, prin folosirea cărora se pot îmbunătăți cu mult designul și eficacitatea programelor.

VLAD ATANASIU

Un foarte tînăr autor care a descoperit acest univers ne face o invitație în **lumea minunată a HC-ului**. E normal. Această lume a fost a tinerilor. Ea însă poate deveni și a copiilor iar cartea de față reprezintă una din cheile cu care se poate deschide poarta acestei lumi.

