

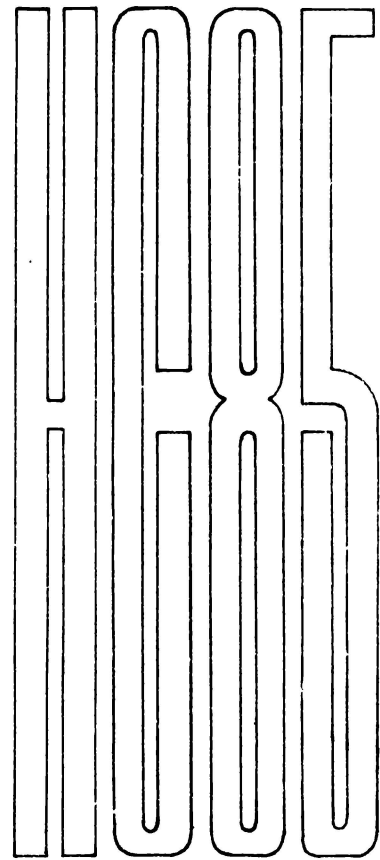
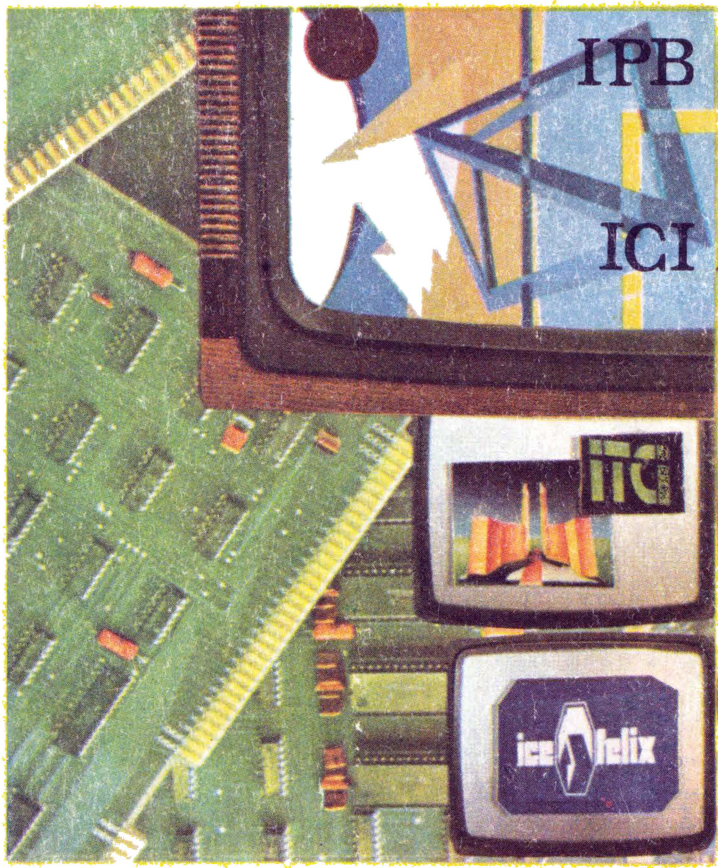
N. Țăpuș
V. Hărăbor

A. Petrescu
T. Moisa
M. Mârșanu

Gh. Rizescu
Tr. Mișu

abc de **CALCULATOARE PERSONALE** și... nu doar atât...

1



EDITURA TEHNICĂ



**AUTOMATICĂ
INFORMATICĂ
ELECTRONICĂ
MANAGEMENT**



BIBLIOTECA DE

Automatică — Informatică — Electronică — Management

SERIA ÎNȚIERE

E. VASILIU

ÎNȚIERE ÎN DISPOZITIVELE SEMICONDUCTOARE

D. STANOMIR

ÎNȚIERE ÎN ELECTROACUSTICA

W. TRUSZ

ABC-UL REPARĂRII RADIORECEPTOARELOR

Traducere din lb. polonă (Ciclul ABC-uri)

A. POPA

ABC DE PROTECȚIA MUNCII (ciclul ABC-uri)

MARGARETA DRĂGHICI

ÎNȚIERE ÎN COBOL

STELIAN NICULESCU

ÎNȚIERE ÎN FORTRAN

PAUL CONSTANTINESCU ȘI ZAHARIA NICOLAE

ÎNȚIERE ÎN ORGANIZAREA ȘI PROIECTAREA SISTEMELOR DE CONDUCERE

I. CREȚU

ÎNȚIERE ÎN ESTETICA PRODUSELOR (ciclul ABC-uri)

E. AISBERG

ABC DE RADIO ȘI TELEVIȚIUNE

Traducere din limba franceză

J. D. WARNIER, B. MI FLANGAN

ÎNȚIERE ÎN PROGRAMARE

Traducere din limba franceză

I. H. BERNHARD, B. KNUPPERTZ

ÎNȚIERE ÎN TRISTOARE

Traducere din limba germană

W. DEPPEERT, K. STOLL

ÎNȚIERE ÎN PNEUMOAUTOMATICA

Traducere din limba germană

ÎNȚIERE ÎN RADIOELECTRONICA CUANTICA

V. POPESCU

ÎNȚIERE ÎN CIBERNETICA SISTEMELOR INDUSTRIALE

I. PAPADACHE

AUTOMATIZĂRI INDUSTRIALE, ÎNȚIERE, APLICAȚII

ȘT. NICULESCU

FORTRAN, ÎNȚIERE ÎN PROGRAMARE STRUCTURATA

J. FORRESTER

PRINCIPILE SISTEMELOR: TEORIE ȘI AUTOÎNȚIERE

PROGRAMATA

Traducere din lb. engleză — S.U.A.

P. DRANSFIELD, D. F. HABER

ÎNȚIERE ÎN PROGRAMATA ÎN METODA LOCULUI RĂDĂCINILOR

Traducere din lb. engleză — S.U.A.

D. RODDY

ÎNȚIERE ÎN MICROELECTRONICA

Traducere din lb. engleză

NICULESCU CI., IOSIF M.

ÎNȚIERE ÎN COMUNICAȚIILE PRIN FIBRE OPTICE

CSABAI DĂNIEL

TEHNICA SONORIZĂRII (traducere din lb. maghiară)

MITROPAN GH., PFLANZER G.

ÎNȚIERE ÎN TELEVIȚIUNEA ÎN CULORI

RADU NEGOESCU

ÎNȚIERE ÎN ELECTRONICA BIOMEDICALA

RADU NEGOESCU

INSTRUMENTAȚIA ELECTRONICA BIOMEDICALA

A. PETRESCU ș.a.

TOTUL DESPRE... CALCULATORUL PERSONAL aMIC

I. DUMITRAȘCU ș.a.

ÎNVĂȚĂM FORTRAN... CONVERSIND CU CALCULATORUL

I. DUMITRAȘCU

ÎNVĂȚĂM COBOL... CONVERSIND CU CALCULATORUL

I. DUMITRAȘCU

ÎNVĂȚĂM MICROELECTRONICA INTERACTIVA

M. PATRUBANY

TOTUL DESPRE... MICROPROCESORUL Z 80

Prof. dr. Ing.
Adrian Petrescu

25.09.1994
Shorbu

șef catedră dr. Ing.
Nicolae Țăpuș
matematician
Viorica Hărăbor

șef lucrări dr. Ing.
Trandafir Moisa
drd. Ing.
Mihai Mârșanu

prof. emerit liceu mat.
Gheorghe Rizescu
drd. Ing.
Traian Mihu

abc de **CALCULATOARE PERSONALE** și... nu doar atit...

Coordonare generală: Prof. dr. Ing. **ADRIAN PETRESCU**

Studiu introductiv: Academician Prof. dr. **NICOLAE TEODORESCU**

Volumul 1

EDITURA TEHNICĂ
București, 1990

CONTRIBUȚIA AUTORILOR

A. Petrescu: 3, 4, 5, 6, 7
N. Tăpuș: 11, 12 p.
T. Moisa: 10, 12 p
Gh. Rîzescu: P-D-E p
V. Hărăbtor: 8, 9.
M. Mărșanu: 1.1, 1.2, 2.1, 2.2, 2.3
D. Mihu 1.3, 1.4, 2.4, p

RECENZII: Dr. ing. **PAUL CONSTANTINESCU**, dr. ing. **LIVIU DUMITRAȘCU**
dr. ec. **GHEORGHE SABAU**, ing. **CONSTANTIN MINDRULEANU**

REDACTOR: ing. **PAUL ZAMFIRESCU**

Mulțumiri întregului colectiv al Întreprinderii poligrafice Sibiu (Director: ing. Petre Pascu; ing. șef: Gheorghe Tămaș; producție: ing. Ioan Șandru (șef producție), Eugenia Oleksik; maiștri în diverse compartimente: Ioan Simtion (m. princ.), Ioan Stroie, Vasile Ienciu, Iancu Popescu, Ioan Rodean, Mircea Pascu, dar mai ales, celor care — cu deosebire — au contribuit direct la realizarea lucrării: (Tastere monotyp: Marioara Suciuc, Emilia Lungu, Livia Mihu; monoturnare: Mihai Schuster, Ioan Gîndilă, Ilie Lupu, Ionel Bica; linotip: Florentina Murărescu, paginare: Dan Koss, ajutat de Mioara Mașca, Ana Berea, Ana Străjan; pregătire offset: Nagy Sibile, Fănica Popovici, Ileana Pădureanu, Marcela Hiesch, Daniela Boantă, Cornelia Spinei, Mariana Băilă, Victoria Barbulat, Mirela Popa; tipar offset: Nicolae Prișcă, Nicolae Vinersar, ajutați de Ioan Funariu, Octavian Moga; corectură: Marcela Tamaș, Alina Iliu, Maria Cheleț, Maria Stoisor; legătorie: Jenica Matei, Elena Lăcătuș, Ana Szylaghi, Paraschiva Giuscă, Ștefan Nanași, Cornel Banea Cu suzele de rigoare pentru omisiunile nedorite, reparabile în al doilea tiraj.

ISBN 973-31-0013-7

ISBN 973-31-0014-5

COPERTE: Arh. **SILVIA PINTEA**
DESENE: Arh. **IULIAN MIHAESCU**

TEHNOREDACTARE: (inițială) **VICTORIA UNGUREANU;**
adaptări și completări 1990 — redacția

Bun de tipar 27.04.1990; Coli de tipar 21,5.

Cărțile s-au realizat la Întreprinderea poligrafică Sibiu

Studiu introductiv

Un titlu atractiv, îmbietor, un colectiv de autori specialiști de concepție și proiectare, de profesori competenți și inimoși este cartea de vizită cu care se prezintă modest și ispititor „ABC de calculatoare personale ... și nu doar atât“, lucrare de mare atractivitate, de o amploare care corespunde în cea mai largă măsură adagiului „... și nu doar atât“, fără a se diminua cu nimic ponderea primei părți a titlului.

Startul — dar și finalul cărții — sînt date într-un original PROLOG—DIALOG—EPILOG, care este o elaborare din 1990, menită de a reflecta probleme cardinale ale concepțiilor autorilor și editorului, în lumina gândirii revoluționare ce va anima, în mod sigur, informatizarea amplă în învățămîntul și educația societății românești.

Concepută cu o structură bogată și complexă, cuprinsă în două volume (cu trei casete magnetice asociate unei părți a tirajului), lucrarea beneficiază de contribuția unor autori prestigioși: trei dintre aceștia: prof. Adrian Petrescu și șefii de lucrări Nicolae Țăpuș și Trandafir Moisa, din Catedra de Calculatoare a Institutului Politehnic București au obținut în 1987 premiul I la Concursul național de creație științifică și tehnică pentru calculatorul personal-profesional Felix-PC, prof. Adrian Petrescu, colectiv IPB și ICE obținind în plus premiul II pentru calculatorul personal HC 85. Adăugăm că prof. emerit Gh. Rizescu a obținut, în 1987, în cadrul aceluiași concurs, premiul I pentru comunicarea privind folosirea calculatorului în învățămîntul liceal. Și ceilalți autori au merite distincte în informatică și tehnică de calcul: Mihai Mârșanu — de la Institutul de Tehnică de Calcul, Viorela Hărăbor și Nicolae Badea de la Institutul de Cercetări pentru Informatică; Tr. Mișu, Eugen Dobrovie și Viorel Cososchi — Întreprinderea de Calculatoare Electronice; C. Hărăbor este profesor la liceul D. Bolintineanu.

Cartea beneficiază din punct de vedere editorial și de competența ca autor a inginerului Paul Zamfirescu, din redacția de informatică și tehnică de calcul a Editurii Tehnice, el însuși autor cunoscut în domeniul automatizării și informaticii.

Lucrarea este împărțită în 11 părți, repartizate 5 în primul volum și 6 în cel de-al doilea. Aceste părți se impart în 25 de capitole, subdivizate la rîndul lor, marea majoritate, în paragrafe care sînt fiecare consacrate unei teme importante atît prin ea însăși cit și prin integrarea sa în contextul general al întregii lucrări; un amplu PROLOG—DIALOG—EPILOG, o bibliografie bogată și anexe de actualizare completează fericit ansamblul.

Partea I descrie atrăgător și substanțial „Evoluția maxi/mini/micro calculatoarelor și a calculatoarelor personale în țara noastră și pe plan mondial.“ Este o temă care cîștigă imediat pe orice cititor, nu numai pe elevii care ar voi să se inițieze în tehnicile de calcul, cu vedeta actuală a calculatoarelor electronice, calculatorul personal, care trebuie să se răspîndească, în grabă, și în învățămîntul nostru primar, gimnazial, liceal și superior.

Lectura celor două capitole ale acestei părți ne întoarce în tunelul timpului, la anii apariției calculatoarelor electronice, la minunile pe care ni le evocau presa și imaginația noastră, la vertiginoasa lor evoluție care trimite în preistoria acestei tulburătoare și covârșitoare invenții.

Contribuții și priorități românești în matematică, cibernetică, electronică, tehnică de calcul și informatică, evocate în primul capitol, ne amintesc de anii în care se frământa pe plan mondial ideea realizată prin inventarea și punerea în aplicație a calculatorului electronic, prin conjugarea eforturilor în aceste domenii.

Azi avem o industrie proprie de calculatoare, iar ideea autorilor de a releva stadiul acesteia și al informaticii în țara noastră este binevenită, ca și cea de a prezenta succint unele aplicații prestigioase ale calculatoarelor în societatea modernă.

Capitolul 2 ne aduce pe planul actualității în domeniul invocat de însuși titlul lucrării, „Evoluția calculatoarelor personale și a microcalculatoarelor personale”, începând cu sublinierea a două elemente de referință. Primul este că această carte este printre primele consacrată învățării utilizării calculatorului personal în țara noastră, și că este prima dedicată integral primului calculator românesc HC-85 compatibil cu un model internațional, Sinclair Spectrum (Marea Britanie).

Al doilea este faptul că, dintru-început, cititorul este inițiat în structura și funcționalitatea calculatorului personal, pus la curent cu rolul și importanța microelectronicii pe plan mondial, cu locul ocupat în prezent de calculatoarele personale și de microcalculatoare în societatea contemporană, precum și cu dezvoltarea actuală a industriei românești în aceste sectoare.

Partea a II-a se ocupă de „Calculatoare numerice. Realizare fizică. Baze aritmetice și logice” și este o introducere matematică, împlinită prin realizările fizice ale funcțiilor și circuitelor logice combinaționale și secvențiale. Aici găsim un capitol privitor la bazele aritmetice și altul la logica matematică și circuitele logice ale calculatorului numeric.

O dată cu partea a III-a este abordat studiul calculatorului personal HC85, din punctul de vedere al structurii și componentelor, al operării și programării. Sînt comparate structura și modul de operare ale calculatorului numeric cu cele ale lui HC 85, dîndu-se și elemente de programare în limbajul algoritmic.

Pe măsură ce se înaintează în contextul primului volum se accentuează obiectivele studiului calculatorului personal HC 85. Astfel, în partea a IV-a este abordată problema programării în limbajul BASIC, legat intim de acest calculator, dîndu-se caracteristicile și elementele acestui limbaj. Prezentarea este sistematică și gradată, cu grija deliberată de a fi accesibilă unor începători fără experiență, dar cu acea curiozitate care caracterizează pe copii și adolescenți. Găsim aici, după noțiunile introductive, paragrafe privilegiate la tastatură, mod de lucru și alfabet.

Capitolul 9 este destinat prezentării detaliate a instrucțiunilor limbajului BASIC. Cititorului i se atrage atenția că folosirea calculatorului este posibilă numai prin intermediul instrucțiunilor pentru care a fost proiectat, exemplificîndu-i-se semnificația noțiunii de instrucțiune, care cuprinde cuvîntul cheie și unul sau mai multe argumente. Prin aceasta se abordează cerința esențială a acordării treptate și sistematice a modului de gîndire umană cu cel al calculatorului.

Se ajunge astfel la partea a V-a unde cititorul este inițiat în „Programarea în limbajul LOGO pe calculatorul HC 85”. Capitolul 10 își asumă sarcina de a dezvălui caracteristicile și elementele limbajului LOGO, debutînd printr-o scurtă introducere în care se discută locul și importanța cîștigate de calculatoare în viața societății de azi, subliniindu-se cele ale calculatorului personal. Accesibilitatea deose-

bită a acestuia este comparată, foarte sugestiv, cu apariția tiparului.

Evoluția limbajelor de programare, al căror număr a trecut de o sută, a condus și la limbajele BASIC și LOGO. Primul permite ca programatorul să „vorbească”, să converseze cu calculatorul, iar cel de-al doilea se prezintă ca un limbaj conceput pentru învățare, dar el are și o semnificație mai profundă, aceea de a fi un limbaj de învățare a unui mod de gândire. El își are originea în cercetările privitoare la știința calculatoarelor, care a promovat gândirea secvențială, algoritmică, în special în cele ce adâncesc gândirea artificială, iar din punct de vedere uman în cercetările lui Piaget cu privire la dezvoltarea gândirii la copii. Autorii răspund la întrebarea „Ce este LOGO?”, punând în evidență principalele caracteristici ale acestui limbaj, ca limbaj de programare. Apoi se consacră un paragraf special locului lui LOGO în școli, în care se atrage atenția asupra inițierii începătorilor și sarcinilor profesorilor de LOGO, arătându-se că programarea este pentru mulți oameni o modalitate importantă de a învăța să învețe, un mod de explorare intelectuală. Paragraful, care este o frumoasă pledoarie pentru LOGO, se încheie cu observația că microcalculatoarele produse în țara noastră, HC 85 și FELIX PC, ca și cele personale compatibile cu APPLE, oferă posibilitatea de a se folosi limbajul LOGO.

Urmează folosirea lui LOGO pe HC-85, cu aspecte privitoare la utilizarea tastaturii, apoi reguli gramaticale ale acestui limbaj și obiectele sale: numere, cuvinte, liste, delimitatori, variabile. În continuare se dau reguli privind lucrul cu proceduri, proceduri cu intrări, tipuri de proceduri, introducerea și editarea acestora. Găsim, de asemenea, un paragraf pentru expresii condiționate, altul pentru linii complexe și în final un glosar de termeni LOGO. Remarcăm și înținderea deosebită a acestui capitol, foarte instructiv și convingător.

Procedurile implicite, denumite primitive, sînt la baza definirii procedurilor de descriere a algoritmilor diverselor programe. Autorii consacră un amplu capitol, 11, primitivelor limbajului LOGO, dînd exemplificări variate și binevenite.

Se începe cu caracteristicile generale ale primitivelor, urmate de primitivele LOGO pentru controlul penelului și ecranului în regim grafic. Apoi sînt prezentate, rînd pe rînd, primitive pentru schimbarea stării penelului, cu specificarea că în limbajul LOGO există două tipuri de primitive: comenzi și operații.

Cu aceste accepțiuni, urmează primitive care specifică starea penelului, apoi cele pentru utilizarea peniței electronice și a ecranului, cele pentru specificarea stării peniței și a ecranului, cele pentru controlul ecranului în regim alfa-numeric. Se trece apoi la primitive LOGO, care specifică operații matematice și logice, unde se disting operatori aritmetici și logici, primitive pentru operații aritmetice și pentru operații logice.

Alte tipuri de primitive LOGO sînt cele pentru lucrul cu cuvinte și liste, pentru preluarea unor elemente din cuvinte sau liste, pentru concatenarea și pentru examinarea acestora. Apoi, găsim primitive pentru comunicația cu echipamentele de intrare/ieșire, pentru citirea informației din exterior și altele pentru ecran.

Alte categorii sînt cele pentru generarea de sunete. Urmează primitive de asigurare a ramificației în program, cum sînt cele pentru ramificarea condiționată, pentru întreruperea procedurilor, pentru execuția și repelarea unei liste de instrucțiuni, pentru lucrul cu fișiere, pentru încărcarea de pe casetă, pentru salvarea pe casetă, pentru controlul imprimantei și tipărirea informației de pe ecran, pentru lucrul cu microdriverul, pentru examinarea spațiului de lucru și pentru specificarea obiectelor și primitivelor LOGO. Din această prezentare de primitive și din specificarea amănunțită a obiectivelor lor ca operații și comenzi, se desprinde destul de pregnant caracterul de limbaj de învățare, pentru cine vrea să învețe cum se poate

învăța cu calculatorul, prin transpunerea conversațională a gândirii umane în cea a calculatorului, prin folosirea limbajului LOGO.

Desigur că mințile cele mai susceptibile de a folosi această modalitate de dezvoltare a propriei lor gândiri sînt cele ale copiilor la vîrstele cînd școala își ia sarcina de a le deștepta și întemeia gândirea logic-deductivă, la care se ajunge pe la vîrsta de 10—11 ani, desăvîrșirea începînd pe la cea de 15—16 ani și continuîndu-se cu eforturi susținute toată viața.

În capitolul 12, „Tehnici de programare în LOGO“ vom găsi procedurile prezentate în capitolul 10, cu accentul pus pe algoritmi, cu precizarea că elaborarea unui program implică atît dezvoltarea algoritmului cît și implementarea acestuia într-un limbaj de programare, operațiile fiind structurate pe două nivele, de care se vorbește permanent în informatică: nivelul logic și nivelul fizic. Vom găsi în capitol proiectarea programelor, ilustrată prin exemple, fiind apoi subliniată proprietatea de recursivitate în limbajul LOGO, care este o modalitate de a scrie proceduri care se apelează pe ele însele.

Capitolul este completat cu numeroase aplicații atractive ca exemple: grafice, sumarea a două numere aleatoare, ordonarea alfabetică, conversia numerelor naturale dintr-o bază în alta, sau desenarea interactivă. Acestea sînt, fără excepție, menite totdeauna să călzească pe începători, care își verifică în mod fericit puterile și se simt atrași spre însușirea cît mai temeinică a tehnicilor, chiar dificile, de programare.

O mențiune specială în încheierea acestui prim volum trebuie făcută cu privire la faptul că lucrarea prezintă pentru prima dată în literatura noastră de specialitate limbajul LOGO, ceea ce aduce un serviciu deosebit de prețios celor ce vor avea sarcina sau plăcerea de a preda folosirea calculatoarelor personale în procesul instructiv-educativ.

Arhitectura lucrării consacră cel de-al doilea volum cu precădere folosirii calculatorului personal HC 85 în învățămînt și educație.

Ministerul de specialitate (în prezent Ministerul Învățămîntului), pornind de la necesitățile obiective de a se introduce informatica în perfecționarea procesului de învățămînt, a aprobat — foarte timid — introducerea în cadrul programei de matematică a unor capitole de informatică și programarea calculatoarelor la clasele IX—XII, cu începere din anul școlar 1987—1988. Și în cărțile de clasa a VII-a a apărut, în 1989, informatica.

Editura Tehnică, publicînd în 1985, în cadrul aceleiași redacții (după cărțile din 1984 privind microcalculatoarele M118) cărțile de mare tiraj „Totul despre... calculatorul personal aMIC“, s-a corelat cu apariția primelor calculatoare personale românești (aMIC și Prae); continuînd în 1989 cu „Totul despre... microprocesorul Z 80, 2 vol. +casetă pe aMIC și PRAE), cu „Totul despre... BASIC, 2 vol.“, în tiraje de zeci de mii de exemplare a recunoscut implicit amploarea instruirii în informatică și tehnică de calcul, la toate nivelele.

În condițiile Revoluției din decembrie '89 necesitatea lucrării a devenit stringentă. De aceea, vedem în realizarea ei un răspuns menit să se înscrie printre acțiunile care marchează o cotitură în pregătirea și educația elevilor noștri.

Volumul al II-lea al lucrării are acest scop, dar ca și primul spune ceva mai mult, fiind deosebit de prețios nu numai elevilor, dar și profesorilor lor, și chiar oricui își dă seama că informatica cere și oferă un nou mod de gândire, tot atît de indispensabil ca și abecedarul, cartea de citire și aritmetica primilor pași în viață ai copiilor întregii omeniri. Este un volum la fel de dens ca primul.

Partea a VI-a tratează „Microcalculatorul HC 85 în procese industriale.

Pachete de programe de aplicații pentru HC 85". Vom găsi aici un capitol privind la proiectarea interfeșelor pentru echipamente nestandard, cu aplicații în măsurători și conducerea microroboșilor industriali. Următorul capitol oferă pachete de programe aplicative: Grafica 3—D, Baze de date, Tabelare electronică, Prelucrarea textelor, realizate în străinătate și larg aplicate și în țara noastră.

Partea a VII-a abordează problemele puse de calculatorul personal în procesul de învățămînt, ca locul informaticii în învățămîntul liceal și asistarea procesului de învățămînt cu calculatorul, ridicînd și pe cea a tipurilor de laboratoare pentru procesul de învățămînt, unde se prezintă în detaliu organizarea spațiului școlar respectiv. Ceea ce nu vedem nici aici, deși are reprezentanți remarcabili și la noi în țară, este laboratorul de matematică. După părerea noastră, laboratorul de informatică, în curs de edificare și înzestrare, ar trebui cuplat în același spațiu, sau în încăperi vecine și deschise una alteia, cu laboratorul de matematică.

În finalul cap. 15 se discută metoda elaborării programelor, dîndu-se apoi un sumar al introducerii calculatoarelor în învățămîntul din unele țări.

Această problemă este reluată pe un plan superior conceptual și aplicativ în cap. 16 „Educație și Informatică” reelaborat integral în 1990, cu includerea conceptelor esențiale din primul congres UNESCO „Educație și Informatică”, ce a avut loc la Paris la sfîrșitul lunii aprilie 1989. Sînt cuprinse de asemenea programe analitice pentru pregătirea elevilor și profesorilor din SU A, URSS, Franța, Japonia, soluțiile problemelor de la concursurile naționale românești din 1987—89, temele recomandate (și rezolvate) pentru cluburile de Informatică. Capitolul are legături evidente cu textul PROLOG—DIALOG—EPILOG și continuării în ANEXE.

Partea a VIII-a prezintă „Programe educaționale în BASIC, pe calculatorul HC 85”.

Găsim programe pentru clasele I—VIII folosite în rezolvarea unor anumite clase de probleme, dar și altele adaptate unui singur tip de probleme, cum ar fi trasarea graficelor unor funcții elementare.

Programele pentru clasele IX—XII cuprind noțiuni delicate, ca funcția modul, grafice de funcții trigonometrice directe și inverse, funcții exponențiale și logaritmice, trasări de conice și de alte locuri geometrice. Trezîndu-se apoi la alte discipline, avem programe de teoria cinetico-moleculară, de optică, de motoare termice, de osciloscopie etc. Această parte are o extindere suficientă pentru a se constitui singură ca o broșură cu caracter tehnic de programare.

Partea a IX-a, (capitolul 19), prezintă un caracter original, fiind însoțită de trei casete magnetice audio cu înregistrări de programe pentru calculatorul HC 85. Acestea au ca obiectiv învățarea interactivă a operării, a deprinderii de a lucra la tastatură, a instruirii în BASIC și LOGO, a proiectării circuitelor logice etc. Programele au fost realizate la catedra de calculatoare a Facultății de Automatică a Institutului Politehnic București, sub coordonarea-autorilor, prof. dr. Adrian Petrescu și șef de lucrări dr. Nicolae Țăpuș, de cîtiva studenți plîni în 1987, astăzi ingineri, citați în lucrare, fiecărui pentru programul elaborat. La acestea se adaugă programul de prezentare a calculatorului personal HC 85, elaborat la Întreprinderea de Calculatoare Electronice București sub coordonarea directorului tehnic, ing. Traian Mihu, el însuși coautor al lucrării.

Pe casele mai sînt înregistrate programe ale unor elevi din clasele VI—XII de la școli generale și licee din București, Arad, Buzău, Iași, realizate în cluburile de informatică ale liceelor, în unele cazuri cu aportul IPB, ITC, ICI, centre de calcul. Vedem, deci, cîtă creativitate poate stimula la copii cunoașterea și folosirea calculatorului personal, în speță a calculatorului HC 85.

Complementelor de matematică, indispensabile folosirii calculatorului le este consacrată partea a X-a. În afară de cîteva noțiuni introductive privind siste-

mele de numerație, cu punerea în evidență a operațiilor elementare în bazele binară, zecimală, octală și hexazecimală, strict legate de operațiile pe calculator, se prezintă operatorii logici cu proprietățile respective, funcțiile logice și problema deciziei, un accent deosebit fiind pus pe noțiunile de algebră booleană. După o scurtă prezentare pe baza unor definiții generale, se trece la simbolurile și regulile algebrei și se evidențiază unele modele utile ale acesteia, ca cele din logica propozițiilor, din teoria mulțimilor, din teoria schemelor cu contacte și relee etc.

Partea a XI-a revine unor „Complemente informatice“ unde se prezintă succint instrucțiunile microprocesorului Z 80, apoi instrucțiunile de utilizare a codului mașină la calculatorul HC-85, cu ajutorul cărora se pot scrie unele programe eficiente, cu viteze convenabile de operare și cu ocuparea unui spațiu redus în memorie. În sfârșit, calculatorul personal-profesional FELIX PC și familia IBM-PS/2 formează obiectul ultimului capitol, 25, primul dintre acestea fiind realizat în țara noastră cu un grad ridicat de integrare tehnologică, cu o structură compactă și un sistem de programare susceptibil de numeroase aplicații. Cît privește familia de calculatoare personale IBM-PS/2, acestea i se face o prezentare pornind de la faptul că a fost produsă în prima parte a anului 1987, deci prezintă o noutate stimulatorie, de interes larg.

Lucrarea este completată printr-o bibliografie care acoperă întreaga arie de cunoștințe prezentată în cele două volume. O serie de actualizări și completări importante (instrucțiunile interpretorului BETA BASIC — un BASIC mai performant, cu care cititorii și cei ce utilizează casetele pot lucra, continuări ale soluțiilor problemelor de la concursurile naționale sau ale rezolvării temelor propuse în cap. 16, cum și multe altele fac obiectul celor 6 ANEXE, de excepție. Cartea (inclusiv casetele) este rodul unei munci intense, desfășurate de întregul colectiv de autori cu competență și devotament.

Sfârșitul consistent al materialului PROLOG—DIALOG—EPILOG este un „happy end,“ fiind consacrat calculatorului HC 85 extins și calculatorului HC-88, produse adaptate lucrului cu dischete și lucrului în rețele.

Se cuvine să menționăm aici aportul tuturor membrilor colectivului, care au reușit să ofere cititorilor o lectură atractivă într-un domeniu care, de obicei, se caracterizează printr-o tehnicitate obositoare. Credincioși titlului ales pentru lucrare, ei au lăsat impresia unei excursii agreabile într-un teritoriu plin de atracții, dar și de obstacole, în realitate conducînd pe cititor să-l cunoască și să-l fructifice. Este indiscutabil un merit deosebit faptul că lectura lucrării interesează nu numai pe începători, dar și pe specialiști, care au ce să rețină din ea, în special din ceea ce autorii au numit „... și nu doar alt!“.

Ceea ce surprinde este faptul că porțiunile realizate de fiecare dintre aceștia se sudează fără discontinuități, dînd lucrării o unitate remarcabilă, ceea ce pune în valoare coordonarea ei.

Reușita acestei lucrări dovedește că lucrul în echipă poate realiza performanțe pe care individual nu le-ar putea obține fiecare dintre membrii echipei, calitate pentru care întreg colectivul de autori merită felicitări deosebite. O mențiune specială se cuvine coordonatorului lucrării, profesorul A. Petrescu, ca și redactorului P. Zamfirescu, pentru unitatea și coeziunea întregii lucrări. Putem prevedea dispariția ei de pe piața editorială într-un timp prea scurt pentru ca cei interesați, în număr foarte mare, să aibă toți șansa de a o procura și felicităm Editura Tehnică pentru inițiativa de a o realiza.

ACADEMICIAN NICOLAE TEODORESCU

Cuprins general

Volumul 1

<i>Studiu introductiv</i> (Acad. N. Teodorescu)	V
Cuprins general vol. 1 și vol. 2	XI
PROLOG—DIALOG—EPILOG (continuă în vol. 2)	XIII
Cuprins detaliat vol. 1	XXVIII
Partea I — Calculatoare, microcalculatoare și calculatoare personale în țara noastră și pe plan mondial	1
Partea II — Calculatoare numerice. Realizare fizică. Baze aritmetice și logice	37
Partea III — Calculatorul personal HC-85. Structură, componente, operare, programare	66
Partea IV — Programarea în limbajul BASIC pe calculatorul HC-85	103
Partea V — Programarea în limbajul LOGO pe calculatorul HC-85	175
	-- 312

Volumul 2

Cuprins general vol. 1 și vol. 2	V
PROLOG — DIALOG — EPILOG — (continuare din volumul 1)	VI
Cuprins detaliat vol. 2	XIV
Partea VI — Microcalculatorul HC-85 în procese industriale. Pachete de programe de aplicații pentru HC-85	1
Partea VII — Calculatorul personal în învățământ și educație	44
Partea VIII — Programe educaționale în BASIC pe calculatorul HC-85	111
Partea IX — Microbiblioteca de programe pe casete	182
Partea X — Complemente matematice	192
Partea XI — Complemente informatice	236
BIBLIOGRAFIE GENERALĂ, CONSULTATĂ ȘI RECOMANDATĂ	275
ANEXE — 1. Metodica algoritmi; 2. Programe educaționale; 3. Interpretorul BETA BASIC; 4. Concursuri informatice (probleme rezolvate) (continuă din vol. 1) 5. Probleme în pseudocod; 6. Soluții ale temelor din 16.6	279
PROLOG — DIALOG — EPILOG (continuare din vol. 2); HC-85 extins, HC-88 disc, rețea	320—
	— 360

În atenția cititorilor:

Cărțile pe care le consultați fac parte din primul tiraj al lucrării „ABC de calculatoare personale”. Ele au fost tipărite parțial în decembrie 1989 și parțial în aprilie 1990. În primele 4 luni ale anului 1990 au fost elaborate cea 200 pagini noi, care completează sau înlocuiesc elaborările 1989. În acest fel, cărțile sînt pe deplin îmbunătățite și actualizate la nivelul primului trimestru al anului 1990, ținîndu-se seamă de documentația străină la zi, de realizările recente ale industriei românești, de noile structuri și programe ale învățămîntului, cum și de stadiul și perspectivele informatizării în lume și în țară.

Sîntem conștienți, însă, de două tipuri de inadvertențe:

- a) au rămas unele greșeli de tipar, ușor detectabile;
- b) nu am putut elimina absolut toate cuvintele de tipul: „pionier”, „utecist”, țări socialiste.

Mai menționăm încă odată înființarea Comisiei Naționale pentru Informatică, dependentă direct de Consiliul de Miniștri; totodată ministerul nou înființat a fost redenumit Ministerul Industriei Electrotehnice și Electronice.

Vă promitem că în următoarele tiraje din 1990, aceste erori nu vor mai exista.

PROLOG — DIALOG — EPILOG

O școală în care profesorul nu învață și el, e o absurditate. Cred că am găsit un motto pentru școala mea. E vorba aceasta extraordinară...: „Nu se știe cine dă și cine primește“.

CONSTANTIN NOICA

Cîte ceva despre calculatorul personal românesc

Calculatorul personal reprezintă una din importanțele realizări ale industriei de tehnică de calcul din ultimii ani, cu un impact deosebit în numeroase domenii ale activității sociale, între care un loc important îl ocupă domeniul educației.

În contextul revoluției tehnico-științifice actuale *informația* capătă un rol de prim ordin, reprezentînd o resursă cu o pondere mereu crescîndă în avuția națională a oricărei țări.

Este evident faptul că numeroase profesii actuale sau viitoare au și vor avea ca obiect informația. În așa-numita „sferă de activitate informațională“ sînt deja cuprinși conducătorii de la toate nivelurile, oamenii de știință și specialiștii, funcționarii din instituții etc. Ponderea acestor categorii de oameni este în continuă creștere în toate țările industrializate.

Puternica proliferare a calculatoarelor personale aduce o contribuție importantă la facilitarea accesului la mijloacele electronice de prelucrare a datelor a unor largi categorii de utilizatori: ingineri, economiști, tehnologi, fizicieni, chimiști, matematicieni, medici, protecționiști, profesori, muncitori, oameni de artă, agricultori, elevi ș. a.

Consecințele utilizării calculatorului personal sînt legate de accelerarea procesului de luare a deciziilor, pe baza unui volum mare de informații prelucrate în mod corespunzător, creșterea calității activității de proiectare de echipamente, procese și dispozitive, folosirea mai eficientă a resurselor materiale și umane la diverse niveluri, realizarea unor importante economii de materii prime și combustibil, creșterea calității produselor, perfecționarea și îmbunătățirea activităților din sfera serviciilor etc.

Nivelul „mediu“ atins de industria noastră de tehnică de calcul a permis ca, pe baza proiectelor realizate de specialiști din învățămînt-cercetare și producție, să se treacă la producția de serie a mai multor tipuri de calculatoare personale, bazate pe microprocesoare.

După cum este cunoscut, un calculator personal are gabarit și greutate reduse, este portabil, încorporează în structura sa unul sau mai multe microprocesoare, o memorie internă cu o capacitate suficient de mare (peste 64 Ko) și dispune de: o tastatură alfanumerică, un sistem de afișare pe ecran TV sau ecran plat (cu cristale lichide sau plasmă), o memorie externă pe casete magnetice, discuri flexibile sau rigide, posedă unul sau mai multe limbaje de programare de nivel înalt și are un preț accesibil (deocamdată pentru colectivități moderate).

Industria noastră de tehnică de calcul produce calculatoare personale și personal-profesionale bazate pe microprocesoare de 8 și 16 biți.

În categoria calculatoarelor personale au fost incluse cele care folosesc un microprocesor de 8 biți, o memorie internă cu capacitate de pînă la 64 Ko, un echipament de vizualizare de tip TV alb / negru sau color, o memorie externă avînd ca suport caseta magnetică (AMIC, Prae, HC-85, TIM-S, Cobra).

Categoria calculatoarelor personal-profesionale este ilustrată de sistemele bazate pe microprocesoare de 8/16 biți, memorii interne cu capacitate de minimum 64 Ko, dispozitive de vizualizare de tip monitor-profesional, echipamente periferice performante, unități de discuri flexibile / rigide, imprimantă etc. (FELIX M118, HC-88, TPD-Junior, FELIX-PC, XT-Junior).

Limita între cele două categorii nu este fixă. Ea se modifică continuu, o dovadă simplă fiind calculatoarele HC-85 extins, cu extensii de disc flexibil, monitoare profesionale etc.

Revenind la calculatoarele personale, specialiștii noștri au ajuns la un consens luînd ca model de referință calculatorul de tip Sinclair Spectrum, larg răspîndit între utilizatorii de calculatoare personale din Marea Britanie și din alte țări.

Modelele realizate, HC-85¹, TIM-S², Cobra³ și, de curând, CIP 4⁴, JET 5⁵, HC-88⁶, emulează aceeași arhitectură, în sensul că operează cu porturi de intrare / ieșire, registre generale și set de instrucțiuni identice, în condițiile unor structuri fizice diferite.

Cite ceva despre informatizarea învățământului românesc

Privind domeniul învățământului, necesitatea cunoașterii și folosirii tehnicii de calcul în economie, în cercetarea științifică și tehnologică, în medicină..., pe scurt — în toate domeniile permeabile algoritimizării, a condus la „coborrea”, unor noțiuni științifice din domeniul informaticii, de la nivelul cursurilor și practicii universitare (sau cel al liceelor de specialitate), la nivelul învățământului public general. Astfel, au apărut — mai întâi în licee — cereurile privind informatica și calculatorul. Ele s-au extins repede în cadrul activităților extracolare din gimnazii, pătrunzând apoi în viața unor școli primare și chiar mai jos. Toate aceste preocupări s-au constituit mai întâi ca activități facultative, acolo unde perfecționarea și stăruința oamenilor s-au ridicat la nivelul cerințelor unor astfel de preocupări. Pregătirea universitară și, într-o anumită măsură, cea postuniversitară a personalului din învățământ, inițiativa acestuia la nivelul școlii, al inspectoratelor școlare și al Ministerului Învățământului, sprijinul acordat de către facultățile, întreprinderile și instituțiile de profil informatic, toate acestea au făcut să crească interesul școlii pentru informatică și calculator.

Ca urmare, în anul școlar 1987—88 au fost inserate, în cadrul programei de matematică a tuturor liceelor țării, citeva noțiuni de bază privind informatica și programarea calculatoarelor. S-a resimțit însă lipsa dotării cu tehnica de calcul a școlilor, cabinetele de matematică sau laboratoarele școlare pentru folosirea calculatorului ca mijloc de învățământ fiind destul de puține și, mai ales, destul de slab înzestrate. Se simte nevoia unor întreprinderi care să asigure tehnica de calcul necesară școlilor. Cu sprijinul Ministerului Învățământului, Învățământului Politehnic și al celui Universitar, ori al unor licee devenite „pilot” în acest sens; cu sprijinul I.T.C.I. și al filialelor sale, al centrelor teritoriale de calcul electronic, a fost lansată și susținută acțiunea de organizare a taberelor centrale sau județene de informatică pentru copii și pentru profesori. Pe baza experienței dobândite s-a reușit definitivarea de către Ministerul Învățământului a unei programe tematice anuale pentru taberele de vacanță, destinate pregătirii copiilor din învățământul primar sau gimnazial în programarea și utilizarea calculatoarelor. Pot fi astfel menționate taberele de calculatoare organizate cu sprijinul întreprinderii de Calculatoare Electronice, al Facultății de Automatică a I.P.B., al Institutului de Tehnică de Calcul și Informatică, al unor facultăți de profil din Cluj, Iași și Timișoara, al unor centre de calcul și licee din țară. În același cadru se înscriu și taberele locale, organizate la nivelul unor centre județene sau ministere, cu sprijinul unor licee în care s-au organizat laboratoare dispunând de tehnica de calcul compatibilă procesului de învățământ. Cursurile acestor tabere de vacanță se încheie prin lucrări teoretice și practice, care stabilesc o ierarhie a competențelor, stimulând astfel gândirea și acțiunea creatoare a copiilor.

Stimularea și evaluarea activității cercurilor de informatică din școli s-au realizat prin intermediul sesiunilor științifice la nivel de școală, interșcoli sau la nivel județean și republican. Începând cu anul școlar 1985—86, pentru stimularea și evaluarea activității cercurilor de informatică s-au introdus concursurile școlare locale, județene și republicane sub titlul

¹ HC-85 a fost proiectat ca model de laborator la Catedra de Calculatoare din IPB (prof. dr. ing. A. Petrescu și asist. ing. F. Iacob) și re-proiectat tehnologic, ca model industrial, în vederea introducerii în fabricația de serie de ing. E. Dobrovie și ing. S. Anghel de la ICE.

² TIM-S a fost proiectat de către specialiștii de la Catedra de Calculatoare din IPT (colectiv condus de prof. dr. ing. C. Strugaru) și de la FMECTC Timișoara.

³ Cobra este proiectat și produs de Filiala ITCI Brașov (dr. ing. Gh. Toacșe și colectiv).

⁴ CIP (Calculator de Instruire programabil) proiectat și produs de Într. Electronica, din 1989/90, cu tiraj de masă, cu interpretorul BASIC pe casetă și cu un preț de desfacere de 9850 lei.

Menționăm că în volumele 56 și 57 din seria AMC a Editurii Tehnice, ce apar în trim. II 1990, este publicat limbajul BASIC pe CIP.

⁵ JET (Jocuri electronice pe televizor) este proiectat și produs de Într. Electromagnetice din 1989/90, cu interpretor încorporat și preț în jur de 11 500 lei, pentru desfacere către public.

⁶ Spre deosebire de CIP și JET (produse de larg consum) HC-88 proiectat la Întreprinderea de Calculatoare Electronice din 1989 (proiectanți: ing. T. Mihu, ing. E. Dobrovie și ing. V. Cososchi) este prevăzut cu unitate de disc flexibil și asigură o dublă compatibilitate, HC-85 și CUB-Z (CP/M).

„Informatica pentru utilizatori“. Această mișcare a cuprins, pe de o parte, reprezentanții cercurilor din gimnazii, iar, pe de alta, pe cei din licee, exceptând liceele sau clasele cu profil informatic, al căror concurs a fost organizat cu mai bine de 15 ani în urmă.

În cadrul tuturor acestor concursuri ale inteligenței și muncii s-au evidențiat mulți dintre cei mai talentați elevi, programele lor devenind în unele cazuri produse-program pentru biblioteca școlară. Este, desigur, necesară stimularea activității școlilor pentru crearea de produse-program necesare procesului de învățămînt și înființarea unor instituții pentru realizarea și sprijinirea realizării acestor programe la nivelul parametrilor de performanță didactice mondiale.

Ca o confirmare a acestor preocupări și a eficienței activității de instruire în cercurile de informatică, vom evidenția rezultatul obținut de echipa României la Concursul Internațional de Programare* de la Sofia, Bulgaria, din perioada 17—20 mai 1987, în cadrul celei de-a II-a Conferință internațională „Copiii în era informaticii — facilități pentru creativitate. Inovație și noi activități“.

În același cadru se înscriu și taberele de calculatoare organizate anual, în timpul vacanțelor de vară pentru studenți, precum și cele pentru profesorii din gimnazii sau din licee. Menționăm aici faptul că, pe baza unor cursuri de cîte zece zile, organizate consecutiv, de-a lungul a trei vacanțe școlare, Liceul „Dimitrie Cantemir“ din București a contribuit la înțelegerea în domeniul cunoașterii și folosirii calculatorului în procesul de învățămînt de către profesorii de matematică și fizică de la liceele patronate de Ministerul Industriei Ușoare. Această activitate de sprijin a fost extinsă, prin colaborare cu unele ministere, care au organizat astfel de cursuri pentru profesorii de matematică, fizică, chimie și biologie.

Ca urmare a acestor preocupări, impulsionate și de introducerea, în anul școlar 1987—88, în programa de matematică a claselor, IX—XII, a unor noțiuni de informatică și programarea calculatoarelor, s-a accentuat cerința dezvoltării producției de calculatoare personale, a documentației tehnice și a cărților de informatică educaționale. De bun augur, în acest sens, au fost și manualele privind tehnica de calcul și informatica — existente prin planul de învățămînt al liceelor de matematică și fizică, sau pe profilul pur informatic. Cartea de față încearcă să răspundă și ea acestui deziderat. Ea s-a constituit ca rod al colaborării mai multor specialiști din învățămînt și cercetare, care au participat efectiv la proiectarea calculatoarelor personale, în introducerea în fabricație și la experimentarea acestora în cadrul cercurilor de elevi din școli, la organizarea și îndrumarea taberelor de calculatoare și informatică, la reciclarea unor cadre didactice și la dezvoltarea prin publicații a schimbului de idei și de experiență. În consecință, în această carte au fost incluse și cîteva din contribuțiile unor elevi care s-au distins la sesiunile de comunicări ale elevilor, la concursuri, sau în cadrul cercurilor de informatică ale unor școli.

Lucrarea de față apare în cadrul Editurii Tehnice, avînd și scopul de a se constitui ca ghid pentru activitățile privind informatica din școli. De remarcat că prin restructurarea sa, la inițiativa redacției de specialitate, cadrul inițial al acestei lucrări a fost lărgit (ca și colectivul său) iar cartea rezultată își lărgiște aria cititorilor cărora se adresează. Ea are astfel un caracter de masă, multe dintre noțiunile introduse și tratate în diversele sale capitole fiind accesibile și copiilor în clasele primare. Unele capitole din lucrare presupun cunoașterea unor noțiuni de bază din domeniile ale fizicii și chimiei, ale electronicii și electrotehnicii — fie la nivelul claselor a VII-a și a VIII-a, ori al claselor a IX-a și a X-a, fie la nivelul avansat al claselor a XI-a și a XII-a. Prin mersul ei, de a simplu la complex, lucrarea se dorește a fi accesibilă oricărui om care dorește să se inițieze în informatică și în aplicațiile acestora. În același timp, cartea încearcă să fie și ghid pentru profesor, un îndrumător metodologic și pedagogic pentru toți cei care conduc cercetări de elevi în domeniul informaticii, un material de pornire pentru membrii cercurilor de matematică și informatică din școli, pentru cercurile școlare ale altor discipline teoretice sau tehnice, care folosesc calculatorul în munca lor științifică. Ea devine deci un material auxiliar învățămîntului din școli, mai ales în munca la clasă și în conducerea cercurilor din gimnaziu și din licee. Multe din capitolele acestei lucrări se pot folosi ca teme pentru cercurile de elevi. În acest scop se pot cita, spre exemplu, capitolele 21, 21 și 22, ele fiind realizate pe baza lecțiilor de inițiere în calcul numeric, logică și algebra booleană, folosite la cercurile de matematică și informatică de la Liceul „Dimitrie Cantemir“ din București, în perioada 1968—1988. Evident, materialul destul de succint prezentat în capitolele enunțate mai sus nu are pretenții de originalitate, unele din noțiunile și exemplele expuse fiind preluate din prestigioase manuale și cărți școlare sau universitare,

* Au participat 7 echipe din 6 țări: R.F.G., România, Bulgaria, Cehoslovacia, Ungaria, U.R.S.S. — aceasta fiind ordinea descrescîndă a clasamentului pe echipe. Elevul Răzvan Jigorea, din Arad, a obținut premiul special al Juriului.

conform bibliografiei citate, ele fiind în parte accesibile ultimelor clase de gimnaziu și îndeosebi primelor două clase de liceu.

Există în lucrare și unele capitole care sînt prezentate în detaliu și pentru prima dată în cărțile noastre de specialitate. Amintim în acest sens limbajul LOGO (capitolele 10, 11 și 12). Se poate menționa, de asemenea, prezența alături de carte a primelor casete magnetice cu programe care pot fi folosite pentru învățarea operării calculatorului HC-85, a limbajului BASIC, a proiectării cu circuite logice combinaționale și secvențiale etc. Casetele mai conțin interpretorul pentru limbajul LOGO, aplicații LOGO și o serie de programe pentru clasele V—VI, respectiv IX—XII, și pentru cercurile de elevi.

Desigur, există multe elemente de noutate pentru tineretul școlar în această lucrare, dar să lăsăm în seama cititorului această problemă. Am menționa totuși aici că dacă această lucrare ar fi putut să apară în anii 1987—88, cînd fusese pregătită pentru tipar, caracterul său de noutate și, mai ales, de utilitate ar fi cîștigat mai mult.

Așa cum s-a mai spus în interiorul acestei cărți, calculatoarele sînt folosite în procesul de învățămînt, în cercurile pentru pregătirea superioară a elevilor și în laboratoare pentru diverse tehnologii, fiind cuplate cu echipamente de laborator, instrumente științifice, mini-roboti industriali etc. Toate acestea permit pregătirea elevilor la nivelul cerințelor solicitate de necesitățile dezvoltării viitoare a țării noastre și ale cooperării libere și eficiente cu toate statele lumii. Pentru aceasta va trebui să facem loc unor largi inițiative în domeniul folosirii informaticii la vîrste foarte mici, prin jocuri accesibile și atractive pentru preșcolarii sau pentru școlarii din clasele primare.

Tocmai unul astfel de scop îl servește, spre exemplu, o parte din programul Cerculul experimental-pilot, numit MINICOMP, care funcționează în cadrul Institutului de Cercetări în Informatică București. Conduc de specialiști din Institut, cercul are ca scop elaborarea și validarea celor mai potrivite metode de instruire a elevilor, în vederea generalizării unor metodologii la nivelul cercurilor de acest tip din întreaga țară. Se organizează anual două tabere de pregătire (iarna și vara), acestea cuprînzînd peste 200 de copii și 50 de profesori din țară. La aceste tabere își aduc o importantă contribuție profesorii din învățămîntul general și din învățămîntul superior (catedra de calculatoare din Institutul Politehnic București), specialiști din Institut, de la Fabrica de calculatoare electronice și de la Centrele de calcul.

În prezent funcționează sute de cercuri de informatică, în care activează zeci de mii de elevi. Se creează astfel un larg acces la baza de software existentă pentru calculatoarele personale cu aceeași arhitectură.

Cite ceva despre educație și informatică în lume pînă în 1990

Deși tehnicile și strategiile pentru introducerea calculatoarelor și a tehnologiilor informaționale în învățămîntul din diverse țări* sînt diferite, există tendințe comune.

Tehnicile naționale, regionale, locale răspund — în măsuri neegale — aceluiași presiuni externe:

- cereri de restructurare a economiei;
- interese industriale (introducerea tehnologiei informaticii în școli completează și întărește dezvoltarea industriei naționale microelectronice);
- cerințe comerciale (fabricația promovează introducerea unor tehnologii în școli ca parte a propriilor strategii de marketing, de creștere a vânzărilor către diferite unități și către familii);
- dorințe ale părinților și copiilor;
- concepții culturale favorabile asocierii la tehnologii informaționale electronice;
- tendințe ale unor lideri politici de a demonstra îmbunătățiri vizibile și concrete ale sistemului educațional;
- dezvoltări tehnologice, care influențează, ele însele, dimensiunea și viteza de implementare, cu reducerea concomitentă a costurilor.

Ca răspuns la aceste presiuni, multe dintre țările dezvoltate au inițiat la începutul anilor '80 — programe ambițioase și costisitoare.

Către mijlocul anilor '80 s-au desprins două grupuri de țări:

Primul grup, care a dezvoltat strategii „restrictive” cu două obiective: 1 — introducerea învățării științei calculatoarelor în clasele secundare superioare și în școlile de specialitate

* Pierre Duguet, administrator șef al CERI din OECD „National Strategies and Their Extension to the International Level”, UNESCO, Paris, 1989.

* Fourth Conference of Ministers of Education of Member States of the Europe Region „Informatics in education”, UNESCO, sept. 1988.

pentru a dezvolta noi perspective in piata muncii și 2 — alfabetizarea in calculatoare la toate celelalte nivele educative. Austria, Belgia, Danemarca, Finlanda, R.F. Germania, Grecia, Irlanda, Italia, Japonia, Olanda, Norvegia, Suedia, Elveția și multe state ale U.S.A. aparțin acestui grup.

— Al doilea grup include țările cu strategii de mal largă înțelegere, deci nu numai de a promova instruirea în știința calculatoarelor sau alfabetizarea informatică ci, în principal, și de a utiliza tehnologiile informatice și ale comunicațiilor, pentru a îmbunătăți procesele de instruire și învățare; (sint incluse obiective complementare, ca: îmbunătățirea calității procesului de instruire, mai buna cunoaștere și influențare a proceselor de învățare, reconstruirea programelor analitice, compensarea scăderii în număr și calitate a profesorilor, îmbunătățirea accesului la educație, dezvoltarea comunicațiilor între școli, pe plan intern și internațional. Exemple sint Franța, Marea Britanie și ținutul Ontario din Canada.

Există tendința de trecere către strategii comprehensive mal largi: Italia (Programul 1985), Portugalia (Planul Minerva, 1984—1988) și Spania (Planul 1985—1989), sau chiar de planuri radical schimbate: Olanda (Planul OSTA, 1989—92). În S.U.A., un recent raport al Oficiului de mijloace tehnologice al Congresului American relatează că 24 state au planuri de lungă durată, iar 13 state au dezvoltat asemenea strategii (Power On! New Tools for Teaching and Learning, September 1988). În Japonia, Centrul de educație în calculatoare a stabilit, în 1986, ca MITI și Mombusho să investigheze sistemele educative bazate pe calculatoare. Desigur, există însemnate diferențe între intensitățile acestor procese în diferitele țări menționate, legate de resurse, de opțiunile de a modifica sau nu metodele tradiționale de organizare școlară etc. De asemenea, se evidențiază două principale strategii de implementare:

1. De la școli pilot selectate, către toate școlile (strategii influențate de resurse limitate și de evaluări ale eficienței pentru a atinge o „masă critică” de acces la hardware, la o integrare a calității software-ului cu programele analitice și la o pregătire corespunzătoare a profesorilor; școlile ce aplică aceste metode sint de ordinul zecilor în Norvegia, de ordinul sutelor în Olanda, Portugalia, Spania; Suedia a lansat un plan „Action programme for computer science education in schools, adult education and teacher training, (1988—1992)”, care include experimentarea învățării asistate de calculator într-un număr limitat de clase (250) din învățământul general. Este un contrast cu puține țări, ca Franța și Marea Britanie, unde școlile sint toate echipate, chiar dacă numai cu un mic număr de calculatoare; aceste strategii sint alese din rațiuni politice, pentru înlăturarea inegalităților între școli, pentru stimularea inițiativelor individuale ale profesorilor, elevilor, directorilor și părinților.

Strategia „ideală” este de a combina aceste două direcții către generalizarea dotării cu calculatoare (Japonia, Suedia și unele state din S.U.A. — exemplu de vîrf fiind Florida, cu raportul 1/18 calculator / elev și un program de 10,5 mil. \$ pentru „5 Model Technology Schools”).

2. De la centralizare către descentralizare, care reflectă rolul important al asistenței la nivelul național (ca în Franța, cu învățământ centralizat) sau local în Norvegia (și în statele federale cu învățământ descentralizat). Statele au rol extins în 5 domenii: hardware — cu compatibilități soft facilitate de aceleași sisteme de operare; software — organizarea unei piețe viabile și informarea profesorilor cu realizările învățământului asistat; pregătirea profesorilor, în timpul sau în afara serviciului; cercetarea — coordonată și susținută financiar; evaluarea strategiilor centrale.

Este posibil că descentralizarea asigură o dezvoltare puternică a creativității, cum rezultă din multe experimentări, care arată că abordarea strictă top-down este o tendință perturbatoare.

Țările aplică și tehnologii specifice:

— În hardware — multe țări au decis să dea prioritate sau să impună utilizarea calculatoarelor de fabricație națională: Ontario — Canada (Icon); Franța (Thomson, Bull, Goupil); Italia (Olivetti); Olanda (Philips, Computada); Norvegia (Scandis, Tiki), Portugalia (Enner 1000), Spania (Computer XP), Suedia (Compis, Microber, ABC), Marea Britanie (BBC, RM și Sinclair), Statele Unite (Apple, TRS, Commodore); Iugoslavia (Iskra, Orel). Această politică promovează industria microelectronică proprie, nu numai în piata școlară ci și în piata familială, în țară și în afara ei, iar, pe de altă parte, favorizează reducerea incompatibilității hardware, problema acută în multe școli (în Japonia în '87 s-au găsit 100 tipuri de calculatoare — iar în 1988, 11 companii electronice au realizat un prototip de calculator educațional bazat pe sistemul de operare THOM; în Norvegia, în 1982 — 75 tipuri). Un număr de țări au autorizat achiziționarea de calculatoare străine, în general compatibile cu sistemul de operare MS-DOS sau cu mediul Unix.

O altă tendință este creșterea numărului de calculatoare (1 micro la 40 elevi în învățămîntul secundar din țări industrializate) și creșterea puterii mașinilor, corelată cu noile pachete puternice de software educațional sau profesional (procesor de texte, tabelării electronice, baze de date). Aceste aspecte țin de experiența țărilor celor mai avansate, depinzînd strict de poziția națională.

— În software, cerința principală este integrarea cu programele analitice a unui software de înaltă calitate, care — însă — costă mult. Este un fel de cerc vicios: software-ul de calitate rămîne scump dacă piața este îngustă, iar piața rămîne îngustă dacă avem prețuri mari. Statul poate stimula piața, încurajînd producătorii de software și reducînd, astfel, costurile. Se estimează că cea mai mare parte a software-ului educațional este de tip instruire programată simplă, doar 10—20% fiind proiectat pentru a apela la procese de gîndire de ordin mai înalt. Oricum, în S.U.A. există disponibile peste 10 000 de programe de instruire, iar în Australia, Canada, Franța, Italia sau în Marea Britanie — între 1 000—4 000. Aceasta este situația în țările cu piețe mari. În ce privește cooperarea cu țările mici, dificultățile țin de diferențele între obiectivele educaționale și, în consecință, de programele analitice; cooperarea ar putea avea loc la diferite nivele: la un anumit nivel se poate opta pentru un transfer global al pachetelor de programe între țări similare; la alt nivel, țările pot coopera în proiectarea unui software apesific, pe baza unei gîndiri pedagogice comune (de ex. S.U.A. și unele provincii canadiene, țările scandinave, unele țări din piața comună, sau alte țări care ar putea găsi sprijin pentru realizarea unor produse de instruire interactive, evitîndu-se „reinventarea roții”).

Țările cele mai avansate se confruntă și cu problema informării profesorilor asupra calității produselor software disponibile (corespondența cu cerințele programelor analitice, conținutul și metodologia de lucru, adresații, prețul, ce hardware și software este necesar pentru funcționare, cum a fost evaluat și de cine). Pentru a asigura aceste informații și pentru a evita respingerea unor produse neadecvate, achiziționate pe bază de reclame comerciale (și — ca o consecință — rejecția calculatorului ca asistent), în multe țări au apărut agenții guvernamentale însărcinate să revizuiască și să evalueze software-ul existent și să acorde „etichete de calitate” pentru cele mai multe produse. Este cazul Australiei, Canadei, Franței, al Japoniei, Marii Britanii, S.U.A. (la nivelul statelor de ex. California sau în Florida). Se constituie baze de date, cu acces internațional.

Pregătirea profesorilor este o problemă crucială; situația nu este deosebit de bună, existînd, în afară de deficitul de hardware și software și o lipsă de profesori, cu pregătire mai ales după serviciu și, mai puțin prin sistemul de pregătire anticipată. Multe state fac eforturi în această direcție, la două nivele: pe de o parte prin cursuri introductive (una sau două săptămîni de familiarizare cu tehnologiile informatice și cu operarea echipamentelor în clasele lor; în Franța cca 25% din profesori, iar în S.U.A. circa 30%, au fost „alfabetizați” (2—5%, în țări în curs de dezvoltare); pentru unele categorii de profesori, la nivele școlare specifice, situația este mai bună, astfel, cca 95% din profesorii danezi din școlile secundare superioare și își completează „alfabetizarea” cu cca 20 ore suplimentare în școlile lor; de asemenea, utilizarea benzilor video pentru completarea cunoștințelor este răspîndită în unele țări, ca de ex. în Marea Britanie.

Dar, această pregătire introductivă poate fi insuficientă și se pune problema unui nivel avansat: astfel, în Franța, există programe cu durata de pregătire de plină la un an, prin care se „antrenează” echipe limitate de profesori „care apoi, în „cascadă” pregătesc alte grupe de colegi. Sistemul este costisitor și nu este atât de eficient, datorită și diferențelor între echipamentele pe care profesorii le vor folosi în școlile lor; o tendință nouă (de ex. în Portugalia) este pregătirea profesorilor la serviciile lor și acordarea lor a unui statut de „cercetători” (uneori în cooperare cu universități și cu alți colegi) asupra pachetelor de programe ce le utilizează.

În 1984, Organizația pentru Cooperare Economică și Dezvoltare (OECD) a organizat o plenară a 122 universități și institute de cercetare din 9 țări membre, dedicată cercetării și evaluării domeniului informaticii educaționale. Cercetarea este îndreptată spre rațiunile pedagogice, spre cele politice, pentru calificarea eficienței reale a tehnologiilor informaționale în procesul de învățare. Evaluări ale informatizării au avut loc și în Anglia (1987), Australia (1986), Scoția (1987), Franța (1986), Norvegia (1987), S.U.A. (1988), unde s-au publicat rapoarte importante, care reflectă experiența școlilor pilot și care evidențiază „masa critică” a accesului la calculatoare, calitatea software-ului integrat în programele analitice și pregătirea profesorilor pentru utilizarea pedagogică a calculatoarelor. În Marea Britanie, o unitate specială recentă a Departamentului de Educație și Știință supervizează cercetările principale orientate asupra eficacității utilizării tehnologiilor informaționale în procesele de instruire și învățare.

Este important, de asemenea, pentru organismele guvernamentale și locale ca, în perspectiva spre anul 2000, în condițiile actuale, fiind tehnologiile nu mai constituie probleme, nici o carte din domeniu să nu mai apară fără un set de dischete, cu strategii educaționale complementare conținutului cărții, cu o mare cantitate de exemple, cu metode de instruire programată, cu concepte interactive de explorare și testare a ipotezelor, cu metode de simulare, cu baze de date relevante etc.: pentru pregătirea unor sisteme educaționale cât mai consistente, în cât mai scurt timp, posibil, pentru a face față schimbărilor inevitabile pe care le așteptăm în învățămîntul informatizat.

Se prevede ca numărul microcalculatoarelor în școlile din Europa de vest să fie în 1990 de trei milioane, crescînd în ultimii trei ani cu 2 milioane. (În 1986 erau 1 milion în școli, în afara celor 12 milioane calculatoare personale aflate în funcțiune în Europa de vest — din care 27,6% în Franța, 25,3% în Marea Britanie, 13,8% în R.F.G., 11,3% în Italia / Spania, 9,2% în Scandinavia, 4,7% în Olanda, 8,3% în alte țări. În 1990 se prevede 20% în Franța, 18,2% în R.F.G., 17,9% în Marea Britanie, 19,4% în Italia / Spania, 8,7% în Scandinavia, 5,9% în Olanda, 9,9% în alte țări — din totalul de 3 milioane unități).

Sectorul cel mai avansat al învățămîntului este cel al învățămîntului superior, cele mai multe universități vestice oferind tuturor studenților puterea calculatoarelor, iar 70% din studenții lucrînd cu computere și acasă: și universitățile din est anunță că au generalizat introducerea științei calculatoarelor. Școlile secundare sînt, de asemenea, sub „vizorul” multor țări, cu o atenție aparte asupra ciclului 2, liceal, al dezvoltării software-ului și al pregătirii profesorilor: calculatoarele sînt materie de învățămînt, uneori opțională, în învățămîntul general, și în cel specializat; în R.F.G. sau Italia firmele de calculatoare sînt asociate la instruire, deosebi pentru școlile specializate. Calculatoarele sînt utilizate și drept instrumente de instruire, în forma unor pachete tutoriale sau de simulare, sau în forma unor programe utilizabile produse de industrie (prelucrarea textelor, tabelări electronice, baze de date, grafică interactivă), sau a unor programe specializate pe subiecte școlare. În primul ciclu al școlilor secundare programele sînt mult mai modeste, în simularea limbajelor, în dezvoltarea conceptelor, mai ales în matematică. În ce privește învățămîntul primar, numai puține țări (Marea Britanie, Franța) au introdus computerele, în mod special pe baza limbajului LOGO.

Alte țări (Belgia, Italia, Danemarca, Olanda, Iugoslavia) s-au limitat la experimente, iar în alte părți (ca în unele landuri ale R.F.G.) implementarea calculatoarelor a fost limitată din motive medicale: ele se utilizează doar pentru jocuri logice și pentru exerciții în forme scrise; puține țări sînt interesate la nivelul învățămîntului preșcolar și numai pentru inițierea în organizarea spațiului și a conceptelor geometice.

Multe țări din Vest, ca și din Est, au încurajat înființarea unor cluburi în afara orelor de școală, deschise unor comunități mai largi, pe bază voluntară, principalele activități fiind scrierea și schimbarea programelor. Pentru educația handicepaților sînt preocupări, în deosebi pentru programe Braille și pentru copiii care învață mai greu.

Educația la distanță progresează odată cu dezvoltarea rețelelor de calculatoare, care permit accesul la bănci de informații, la stații de instruire la domiciliu, la videotext și la poșta electronică, la transmisiile digitale, la diversificarea unor instrumente interactive pentru instruirea personalizată și în locuințe. Spre exemplu, Austria a organizat un sistem de lecții pe calculatoare personale pentru studenții care dispun de terminale la locuințe.

În ultimii 20 de ani consumul de putere de calcul a crescut după o curbă exponențială cu o dublare la fiecare an; astăzi, cea mai mare parte din aplicații se realizează pe echipamente de „birou” (desktop computer). De la Apple II (1978) și de la IBM PC (1981), cu display-uri de mică rezoluție și memorie limitată, la modelele noi supraprotective, cu diferite sisteme de operare, cu memorii mari și grafică superioară; apare o întrebare legitimă, cât timp calculatoarele personale Macintosh (1982) și IBM PS/2 (1986) vor mai rezista, chiar cu marea lor varietate de standarde grafice, în fața așa-numitelor stații de lucru?; acestea sînt caracterizate prin mai multe milioane de instrucțiuni pe secundă (MIPS) ale microprocesorului — inima unității centrale (CPU), printr-o memorie principală de 2—8 milioane de caractere (Mbytes) și un ecran depășind un milion de pixeli color. Discursurile au sute de Mbytes, ca memorii externe și mai mulți Gigabytes în rețele (memorii optice pot fi incluse). Sistemele de operare sînt de tip multiproces, alocînd mai multe ferestre, care operează simultan pe ecran: software-ul avansat implică grafică interactivă și animație: conectarea la rețele locale (LAN) și spațiale (WAN) permite dezvoltarea poștei electronice și a altor sisteme similare. Toate acestea nu puteau să nu influențeze informatizarea învățămîntului, ca și progresele în inteligența artificială în traducerea automate, recunoașterea vorbirii și formelor, sistemelor expert ș.a. Producătorii de hardware ca și editorii nu au investit mult în software-ul educațional spre deosebire de companiile de software.

Cartea de față poate constitui, prin întregul său conținut, un suport pentru o serie ierarhică de manuale de tehnică de calcul și informatică. Cuprinsul său poate orienta în elaborarea unor noi programe analitice românești, începând din 1990. În același scop, am prezentat în capitolul 16 programe analitice de informatică pentru școli primare, școli secundare, „alfabetizare” informatică, pregătirea profesorilor, din diverse țări (S.U.A., Franța U.R.S.S., Japonia). Aici, am găsit necesar să redăm și să comentăm — pentru a avea un tablou mai complet — și o programă analitică pentru integrarea informaticii în studiile tehnico-ingenerești din școli speciale și superioare din R.F.G.* Este prevăzută pentru cursuri de 3 trimestre, în primii doi ani de studiu. Informatica este considerată o metodologie științifică pentru tehnologia informației, ce interesează pe ingineri și tehnicieni ca instrument de proiectare, modelare, simulare, gestiune, fabricare, calcul.

Programa poate fi divizată în trei nivele: 1) utilizarea calculatoarelor cu software de aplicații standard; 2) aplicarea optimă a informaticii în procesele ingineresti; 3) — dezvoltarea programelor și configurațiilor hardware pentru sarcini ingineresti.

Obiectivele globale ale programei sînt: 1° — cunoașterea echipamentelor (componente, operare, instalare, diagnoza erorilor); 2° — cunoașterea sistemelor de operare și a utilitatelor (funcțiuni, comenzi DOS, rutine pentru conducere date, fișiere ș.a.); 3° — introducerea în programe aplicative standard, prelucrare texte, baze de date, tabelărie electronică, grafică, software de comunicații); 4° — conceptele științei calculatoarelor (informația ca obiect manipulabil, reprezentarea informației prin date memorate, redarea datelor în forme textuale sau vizuale, bazele înțelegerii viitoarelor domenii ale informaticii, mijloace de învățare a instrucțiunilor; termenii de CAD, CAM, sisteme specializate, procesare paralelă, transputere); 5° — dezvoltări de software (structuri de programe, structuri de control; structuri de date, scule de programare — editor, interpretor, compiler, linkeditor, generator, limbaje de programare — Turbo-Pascal, organizarea colecțiilor mari de date; metode de dezvoltarea software-ului — incluzînd testarea și documentarea — proiectarea interfețelor); 7° — impact social și economic al tehnologiei calculatoarelor.

C u r s u l p r e l i m i n a r . **Calculatoarele personale drept mijloc de scriere și desenare** (semestrul I, 30 ore curs + 20 ore practică, un C.P. la 2 studenți; 1 CP cu disc rigid pentru profesor, conectat cu celelalte pentru încărcarea programelor, tablă, proiector cu tabletă CP; sesiunile teoretică și practică, ca blocuri separate; 10 studenți la un profesor). **Structură:** 1° — Explicarea unităților vizibile ale CP. (Unitate centrală, monitor, tastatură, memorie, discuri flexibile și rigide, imprimantă; încărcarea unui program standard, copii pe disc, întreținerea fișierelor, tipărirea fișierelor). 2° — Încărcarea și pornirea unui program de prelucrare de texte. 3° — Funcțiuni de bază ale procesării de texte — intrări, tipăriri, memorări — (editare, mișcare blocuri, construcție blocuri standard, funcții de căutare, redare texte și tabele). 4° — Întreținerea fișierelor, manipularea lor. 5° — Cunoașterea mai adîncită a funcțiilor procesoarelor de texte. 6° — Cunoașterea mai adîncită a funcțiilor sistemelor de operare. 7° — Explicarea unităților funcționale ale CP. 8° — Practica instalării unităților și programelor, reconfigurarea lor. **Obiective:** operarea cu dispozitivele de intrare (tastaturi, șoricel...). Deprinderea executării cu PC a procesării de texte și de grafică tip comercial. Instalarea, după manuale, a hardware-ului și software-ului.

S e m e s t r u l II. Fundamentele informaticii (I) pentru ingineri și tehnicieni (60 ore, în clase cu aceleași condiții ca în primul semestru, plus programe CAD, bază de date relaționale, tabelărie electronică pe discuri rigide). **Structură:** 1° — Funcții de bază ale programelor CAD (desenarea unui obiect simplu tridimensional, adăugarea de texte sau măsurători). 2° — Analiza imaginilor, în termenii conceptelor informatice (limbajul vorbit, scris, simboluri, imagini, reprezentarea vizuală și percepția imaginilor, modele conceptuale de reprezentare a imaginilor) 3° — Opțiuni tehnice de stocare a datelor (caracter, înregistrare, fișier). 4° — Cunoaștere mai amănunțită a programelor CAD. 5° — Sisteme de baze de date (funcții de interogare, termeni de căutare, comenzi, analiza limitărilor). 6° — Abstracții ale funcțiilor principale, pe baza teoriei mulțimilor și a algebrei Boole. 7° — Organizarea datelor ca obiecte în baze de date relaționale. 8° — Mijloace de acces la bazele de date, limbaje de regăsire. 9° — Sisteme specializate, de ex. tabelărie electronică.

Obiective: lucrul cu programe CAD, de desenare, de organizare a datelor ș.a.

S e m e s t r u l III. Fundamentele informaticii (II) pentru ingineri și tehnicieni (60 ore + 40 ore practică, în blocuri, cu aceleași caracteristici, limbajul PASCAL).

* Peter Gorny, *Curriculum Proposal for the Integration of Informatics into Engineering Programs offered at Colleges and Universities in Developing Countries*, Congres UNESCO, Paris, aprilie, 1989.

Structură. 1° — Ce face un program în calculator (calculatorul — o mașină incompletă, analiza unui program simplu PASCAL de la program la codificarea în limbaj mașină, funcțiunile compilerului PASCAL. 2° — Bazele limbajului PASCAL. 3° — De la problemă, prin algoritm, la program (descripție semiformală prin pseudocod sau structograme, convertirea în programe, recunoaștere erori, specificare, proiectare, implementare, testare programe. 4° — Tipuri și structuri de date (predefinite și autodefinitive, masive, înregistrări, fișiere). 5° — Subprograme modulare (aplicarea procedurilor și funcțiunilor, declarații, accese la fișiere în programe PASCAL), interferențe între programe Pascal și alte programe. 6° — Metode simple de dezvoltarea software-ului (module, mașini abstracte, nume, interfețe, parametri, abstractizarea datelor, modele shell, comunicații și dialoguri între mașini, metode de documentare, practica programării sarcinilor ingineresti. **Conținut și metodologie:** calculatorul drept mașină cu stări finite, dezvoltarea sistematică a programelor, deprinderea lucrului în echipă pentru dezvoltarea soluțiilor din aplicațiile ingineresti.

Se mai poate vorbi de un semestru IV (opțional) referitor, de exemplu, la hardware (arhitectură, proiectarea interfețelor, rețelelor), la software (programare logică, grafică de computer).

Tot în perspectiva programului de informatizare a învățământului ce se va derula cu siguranță acum — după Revoluția din decembrie 1989 — trebuie cunoscute răspunsurile date de 43 state membre ale UNESCO la chestionarele ce au precedat Congresul internațional UNESCO „Educația și Informatica”, din aprilie 1989, Paris.

Astfel, ierarhizarea obiectivelor statale prioritare pentru introducerea informaticii în educație este: pregătirea tinerei generații pentru o nouă lume — 76%; îmbunătățirea procesului de învățământ — 54%; ridicarea nivelului dezvoltării economico-sociale — 38%; pregătirea specialiștilor — 33%; pregătirea utilizatorilor — 30%; descreșterea decalajului tehnologic între națiuni — 25%, sprijin față de întârzierile școlare — 14%; reducerea disparităților în cunoaștere dintre grupuri sociale — 11%; facilitarea autoinstruirii — 8%; facilitarea integrării sociale a celor fără lucru — 5%.

Numărul calculatoarelor educaționale varia, în diferitele țări, de la zeci, sute, mii, sute de mii; în țările pieței comune erau 1 milion de calculatoare personale în 1988 și mai mult de 1 milion în S.U.A.; se ajungea la 1 calculator pentru cel mult 50 copii.

Dificultățile introducerii informației sînt de trei categorii: **Administrative:** (lipsuri strategice — 22%, lipsa de suport ierarhic — 19%; limitări financiare — 60%). **Tehnice:** echipament insuficient, incompatibil — 40%; software insuficient sau de proastă calitate — 55%; probleme de limbaj — 10%; întreținere — 20%, electricitate — 14%; lipsa personalului specializat — 35%. **Pedagogice:** (lipsa unor obiective clare — 20%; cercetare și evaluare insuficientă — 46%; instruirea personalului educațional — 40%; rezistența la schimbări — 30%).

Utilizarea informației în educație pe diferite stagii și tipuri de instruire este redată sintetic în matricea:

Tip/Stagiu	Alfabetizare %	Instruire informatică %	Mijloace învățare %	Instruire instructori, prof. %	Management, administrație %
Terțiar	50/22*	57/22	27/30	30/30	38/34
Secundar	38/27	22/33	0/50	24/22	19/30
Secundar tehn./specializ.	50/19	43/19	5/46	24/22	22/33
Primar	3/8	8/5	11/24	22/30	3/27
Preșcolar	3/24	—	5/46	8/5	3/8
Special	50/19	3/—	3/16	11/14	3/16
Adulți (ed. cont.)	16/27	14/22	3/19	11/19	16/19

* Generalizat/Pilot.

Țările membre sînt: Anglia, Argentina, Belgia, Bolivia, Brazilia, Bielorusia, Camerun, Canada, Republica Centrafricană, Ciad, Chile, Congo, Costa Rica, Cuba, Cipru, Cehoslovacia, Finlanda, Franța, Gabon, R.D.G., Ungaria, India, Irlanda, Israel, Japonia, Iordania, Coreea de Sud, Kuweit, Luxemburg, Mauritius, Mozambic, Polonia, Samoa, Senegal, Spania, Srilanka, Elveția, Surinam, Siria, U.R.S.S., Iugoslavia.

Cooperarea între statele membre, mijlocită și de Intergovernmental Informatics Programme (IIP), include schimburi de informații și experiențe; schimb de software, instruirea specialiștilor și profesorilor, cercetare; asistență tehnică și financiară.

Ancheta USEIT (Use in Systems of Education of Information Technologies) a UNESCO este utilă în aprecierea stadiului național și la noi*. Anchetele complementare privind editorii de software au reliefat tipurile de pachete de software, evaluarea și proporția lor (software proiectat pentru exerciții; programe tutoriale — învățare interactivă, — simulare; prelucrare texte, tabelărie, baze de date, CAD, sisteme expert; jocuri, altele. Fabricanții de hardware au dat indicații procentuale, menționate anterior.

În „Comparative Study on Criteria and Procedures for the Evaluation of Educational Software, editat de Richard N. Tucker, prin Netherlandes Institut Voor Audio-visuele Media, prin contract între Internațional Council for Educational Media (ICEM) și UNESCO sept. 1988, la care au participat Canada, Japonia, Ungaria, Italia, Olanda, Scoția, Anglia, S.U.A., s-au analizat criteriile de evaluare a software-ului educațional, definind 3 tipuri de pachete: 1) — Programe profesionale (inițial produse pentru comunități de afaceri sau profesionale). 2) — Programe deschise sau de aplicații (proiectate pentru educație — conținut liber). 3) — Programe didactice (conținut specific educațional).

Evaluarea apare în trei faze: a) Testare în producția produselor. b) Selecția prin traducere și conversie (în țările mici, la achiziționarea și traducerea programelor). c) Selecția de către autorități și școli, de către utilizatori. Deci se poate vorbi de o evaluare formativă (în producție) și sumativă (la utilizatori).

Statele au metode instructive diferite de a evalua software-ul educațional, toate bazate pe tehnici ale mulțimilor fuzzy (vagi), adică pe chestionare cu acordarea de note pe diverse secțiuni: (A — întrebări considerate înaintea utilizării pachetelor de programe; B — întrebări considerate în timpul utilizării programelor; C — întrebări după utilizarea programelor).

Pachetul educațional trebuie să includă documentația educațională și tehnică, iar răspunsurile trebuie să se dea la următoarele:

A. Întrebări înaintea utilizării pachetelor

Obiective și scopuri educaționale

1. Sînt clare enunțurile obiectivelor educaționale?
2. Pentru ce subiecte și pentru ce studenți se utilizează?
3. Sînt obiectivele pachetului relevante în cursuri și instituții pentru studenții utilizatori; există enunțuri asupra abilității cerute utilizatorilor?
4. Sînt enunțate clar enunțurile stilului de instruire (exerciții — repetare, simulare, interactiv-tutorial ș.a.) utilizat în program?
5. Este stilul de instruire adecvat obiectivelor cursurilor, instituției, profesorului și elevilor?
6. Pachetul este pentru cazul clasei, grupurilor, studenților individuali sau profesorului?
7. Există indicații privind modul în care poate fi apreciată atingerea obiectivelor de către pachet?
8. Dacă programele pot fi utilizate fără asistență, au fost asigurate notițe pentru studenți, pentru o instruire acceptabilă?
9. Este clar ce activitate pre-, post- și în lucrul cu calculatorul este necesară și există material suficient pentru profesor sau student pentru această activitate?

Cerințe tehnice

1. Sînt clare enunțurile privind ce hardware este necesar pentru ca programul să ruleze (calculatorul, memoria, periferiile necesare)?
2. Sistemul de operare și alte cerințe software sînt indicate?

B Întrebări în timpul utilizării programului

Proiectare educațională

1. Este dată o introducere cu numele pachetului, urmată de o secțiune opțională cu instrucțiuni de utilizare?

* A se vedea în continuare.

2. Dacă pachetul poate fi utilizat cu o largă categorie de abilități la diferite nivele de dificultate incluse?

3. Este utilizatorul implicat într-o cale activă sau relevantă în interacțiunea cu programul?

4. Este adoptat un stil „conversațional” (fraze și mesaje mai curând scurte decât lungi)?

5. Sînt mesajele variate și prietenoase?

6. Dacă programul poate să răspundă precis și fiabil la diferite întrebări corecte privind o problemă particulară?

7. Sînt permise formate fără restricție sau răspunsuri lungi?

8. Este utilizatorul pus să introducă mari volume de informații?

9. Dacă se utilizează formatul cu selecții multiple pentru chestiunea pusă și răspunsul solicitat?

10. La sfîrșitul pachetului există un sumar din care utilizatorul să deducă ce a învățat și ce activitate mai trebuie depusă?

Prezentare și amplasare

1. Este prea mult text prezent pe ecran?

2. Sînt marginile și spațierile bine utilizate pentru înțelegere ușoară?

3. Sînt utilizate adecvat sublinierile, cartușele, afișările intermitente și culorile?

4. Sînt folosite efectiv sunetele și graficele și care sînt scopurile educaționale clare în utilizarea lor?

Controlul și ghidarea instruirii

1. Trebuie ca elevul să verifice succesiunea în program sau programul singur „merge” la o nouă pagină sau secțiune?

2. Trebuie ca utilizatorul să revină la pagini anterioare repetat sau cheamă utilitarul HELP? Există o cale de ieșire din program?

3. Pot elevii să corecteze ușor erorile?

4. Este întotdeauna evident pentru utilizator, în orice punct din program, care sînt opțiunile disponibile și ce trebuie să facă?

5. Sînt opțiunile disponibile într-o manieră consistentă în tot cursul programului?

6. Există o reacție semnificativă cînd se dau răspunsuri greșite?

7. Cînd se dau răspunsuri corecte se asigură o reacție pozitivă?

Robustețe tehnică

1. Se poate distruge programul prin manipularea greșită a unei taste?

2. Sînt dezactivate toate tastele cu excepția aceleia ce trebuie activată?

3. Sînt protejați utilizatorii de mesajele sistemului?

C Întrebări după utilizarea programelor

1. Care sînt obiectivele atinse (A) prin utilizarea programului? Să se revadă răspunsurile din prima secțiune din A în lumina experienței programului — au fost validate pretențiile ridicate față de program?

2. Au fost atinse alte obiective valoroase (nepropuse) prin folosirea programului?

3. Pentru atingerea obiectivelor a fost eficient utilizat calculatorul? A exploatat programul capabilitatea calculatorului (memoria, manipularea unei mari cantități de date, generarea unor grafice dinamice, interacțiunea cu calculatorul etc.) Ar fi putut fi atinse obiectivele pachetului, mai ieftin și mai convenabil, fără utilizarea calculatorului?

4. Este programul proiectat modular, pentru a permite modificări?

5. A fost pachetul testat în domeniu cu auditoriul avut în vedere? Dacă da, care au fost rezultatele testării?

6. Au fost publicate în revistele educaționale menționări asupra pachetului? Dacă da, menționați.

7. Cel care a dezvoltat sau publicat programul trebuie să fie rugat a da numele utilizatorilor curenți ai pachetului?

8. Este clar dacă sînt necesare activități viitoare? Dacă da, este furnizat suficient material studentului sau profesorului, pentru a desfășura această activitate?

● Calculatorul HC-85 extins, cu interfață de disc flexibil, interfață serială (conectare imprimantă etc.) și interfață de rețea ● Calculatorul HC-88 cu dublă compatibilitate: HC-85 extins și CP/M ● Operarea cu discul flexibil; operarea cu periferice; operarea în rețele locale; joc de rețea ● BASIC 85 extins ● Manipularea fișierelor ● Apelurile funcțiilor interfețelor din limbaj de asamblare.

Calculatorul personal HC-85 extins.

Microcalculatorul HC-85 în configurație extinsă este echipat cu o placă suplimentară, cuplată prin conectorul de extensie cu placa de bază. Aceasta placă înglobează trei interfețe: pentru unitatea singulară sau duală de disc flexibil, pentru o linie serială standard RS-232-C/CCITT V24 și pentru cuplarea microcalculatoarelor HC într-o rețea, folosind o pereche de fire torsadate.

Cuplarea unor unități duale de discuri flexibile de 5 1/4" asigură o capacitate externă de memorare de 1,28 Mo, pentru maximum 128 fișiere distincte

Interfața serială permite cuplarea la HC a unei imprimante sau interconectarea cu un alt calculator.

Interfața pentru cuplarea în rețea oferă posibilitatea legării pînă la 64 de sisteme, oferind astfel o soluție pentru aplicațiile în domeniul învățămîntului, în laboratoarele școlare de informatică.

Toate aceste noi facilități hardware de care dispune HC-85 în configurație extinsă posedă un suport software, prin extensia instrucțiunilor limbajului BASIC - HC.

Noile instrucțiuni, cît și modurile de operare cu noile facilități sînt prezentate într-o anexă.

Calculatorul personal HC-88.

Microcalculatorul HC-88, cu dubla compatibilitate, reprezintă o

soluție care înglobează atât caracteristicile sistemului HC-85 cât și pe cele ale microsistemului CUB-Z, ca mașină CP/M. HC-88 s-a conturat ca o soluție de înlocuire în producția Intreprinderii de Calculatoare Electronice București a sistemelor CUB-Z.

In configurația maximă, HC-88 constă din următoarele echipamente:

- HC de baza format din:
 - echipamentul nucleu (MP Z 80A; 80 K RAM (64+16), 2K ROM)
 - . placa de bază,
 - . sursa de alimentare,
 - . monitorul monocrom,
 - tastatura,
 - doua unități de disc flexibil de 5 1/4",
 - imprimantă matricială grafică,
 - monitor color,
 - codor RGB/PAL.

Monitorul monocrom și monitorul color pot fi cuplate simultan la calculator.

Optional, echipamentul nucleu HC-88 poate conține:

- placheta IO/EPROM,
- placheta IO/SIO,
- modul extensii EUROCARD,
- placheta programator EPROM
- placheta extensie memorie RAM 256 Ko,
- placheta extensie memorie RAM 1 Mo.

Conectorul serial permite legarea unui cablu de adaptare pentru casetofon.

CONTINUARE IN VOLUMUL 2
LA PROLOG—DIALOG—EPILOG

Cite ceva despre carte:

O prezentare sistematică există în studiul **introdactiv**, o descriere succintă — pe coperti, structura detaliată — în **tablele de materii** din volumele 1 și 2. Remarci privind **caracteristicile** ce o fac **utilă informatizării învățămîntului** nostru se dau în paginile XV—XVI din vol. 1, unde se menționează și **întîrzierea apariției** sale. Evidențiem trei factori care au concurat la începerea elaborării sale în 1986: un ciclu al Editurii Tehnice, cu apariții anterioare, dedicat de redacție calculatoarelor personale; propunerea globală primită de la cercurile de informatică extrașcolare pentru elevii din clasele I—VIII; dorința de a scrie a unor autori anteriori — creatori și utilizatori de calculatoare personale. În 1987/88, a fost gata pentru tipografie o variantă a lucrării, cu un titlu născocit de redacție (abc de calcul electronic), pentru a o feri de hotărîrea aberantă de la începutul anului 1988 a „academicienei-analfabete-dictatoare”, care elimina de la apariție toate cărțile despre calculatoare, cu o interdicție expresă privind **calculatoarele personale**, ce îi apăreau drept cele mai „nocive” cuvinte pentru popor.

Cite ceva despre autori:

ADRIAN-CRISTIAN PETRESCU, profesor la Catedra de calculatoare din Institutul Politehnic București; absolvent al facultății de Electronică și Te, iar din 1964 doctor inginer în domeniul calculatoarelor. Din 1971, Fellow of British Computer Society. A condus și a participat efectiv la proiectarea și realizarea mai multor sisteme de calcul: MAC I și MAC II — (Premiul MEI — 1965) FELIX MC 8 (Premiul Traian Vuia al Academiei R. S. România — 1975). FELIX M18, FELIX M118, FELIX M-216, aMIC, HC-85, FELIX-PC (Premiul I la Concursul de Creație Științifică și Tehnică din anul 1987 pentru Felix PC și premiul II pentru HC-85). Este autor al lucrărilor: **Calculatoare automate și programare** (Ed. I — 1970 Ed. a II-a — 1974) — Editura Didactică și Pedagogică, **Microprogramare — Principii și aplicații**, Editura Tehnică, 1975; coautor și coordonator al lucrărilor, **Microcalculatoarele FELIX M18, M18B, M118** — Editura Tehnică 1984, **Totul despre aMIC** — Editura Tehnică 1985 și coautor — editor la lucrarea „**Calculatoarele electronice din generația a cincea**”, în Editura Academiei R. S. România, 1985.

NICOLAE ȚĂPUȘ, șeful catedrei de calculatoare, Facultatea Automatică din Institutul Politehnic București. Absolvent al Facultății de Automatică (1972), iar din 1982 doctor inginer în domeniul calculatoarelor. A participat la proiectarea și implementarea sistemelor de calcul românești, FELIX MC8 (premiul Traian Vuia al Academiei R.S.R., 1975), FELIX M18, FELIX M118, FELIX M216, FELIX PC (premiul I la concursul de Creație Științifică și Tehnică din anul 1987). Este autor și coautor al unor lucrări de specialitate (manuale, articole) destinate studenților și utilizatorilor de sisteme de calcul. Este coautor al lucrării **Microcalculatoarele FELIX M18, M18B, M118**, Editura Tehnică, 1984.

TRANDAFIR MOISA, șef de lucrări la catedra de calculatoare, Facultatea Automatică din Institutul Politehnic București. Absolvent al Facultății de Automatică (1973), iar din 1982 doctor inginer în domeniul calculatoarelor. A participat la proiectarea și implementarea sistemelor românești de calcul: FELIX MC8, (premiul Traian Vuia al Academiei R.S.R. — 1975, FELIX M18, FELIX M118, FELIX M216, FELIX PC (premiul la concursul de Creație Științifică și Tehnică din anul 1987). Este autor și coautor al unor lucrări de specialitate (articole, manuale) destinate specialiștilor din domeniul tehnicii de calcul. Este coautor al lucrării **Microcalculatoarele FELIX M18, M18B, M118**, Editura Tehnică, 1984.

GHEORGHE N. RIZESCU n. 1928 Călina, Vlcea, absolvent al Facultății de Matematică și Fizică din București în 1952, profesor emerit. În 1988 iese la pensie ca director al Liceului „Dimitrie Cantemir” din București. Ca profesor și ca redactor la *Gazeta Matematică* este autor de probleme, articole și comunicări cu caracter științific sau metodic-științific; autor al lucrării „**Laboratorul de matematică — teme și fișe experimentale**, O. I. D. — I. C. P. E., București, 1978. De asemenea, este coautor la o serie de lucrări publicate în revista „**Studii și Cercetări Matematice**”, Editura Academiei R.S.R., la lucrarea „**Laboratorul de Matematică**, E. D. P., 1973 și la lucrările „**Teme pentru cercurile de matematică din licee**”, vol. I (1977, E. D. P.) și vol. II (1980, E. D. P.). Coautor la manualul de matematică „**Algebra pentru clasa a IX-a**”, E. D. P. — 1978, ... și la volumele I și II „**Totul despre calculatorul personal aMIC**”, Editura Tehnică, 1985. Autor a numeroase articole și comunicări privind informatica și învățămîntul. S-a preocupat îndelung, cu rezultate notabile, de modernizarea tehnologiei învățămîntului.

VIORICA ELENA HĂRĂBOR, analist la Institutul de cercetări pentru Informatică (ICI), București. Absolventă a Facultății de Matematică a Universității din București (1970). Se ocupă de utilizarea calculatoarelor personale și de inițierea copiilor în informatică. A publi-

Datorită complexității prevăzute pentru ansamblul cărți-casete magnetice, a unui colectiv ce a atins 8 persoane din 4 unități diferite de învățământ—cercetare—proiectare—producție, a unei structurări ambițioase, **procesul de editare a fost lung**, cu numeroase iterații de îmbunătățire și corelare a elaborărilor parțiale, de selectare a sutelor de module și pachete de programe cum și de includere a unor lucrări valoroase de programare ale unor elevi și studenți. Mari întârzieri au fost provocate de lipsa de hîrtie și de bandă magnetică pentru casete; după multă vreme consumată și multe stăruințe depuse de redacție și de unii dintre autori, s-au obținut materialele și în dec. 1989, am ajuns la tipărirea unei mari părți a lucrării. După **revoluția din decembrie 1989** (în condiții la care ne referim în pag. VIII—IX din vol. 2) cu un colectiv lărgit — de 11 persoane, folosind documentație străină, nepermisă și necunoscută în 1989, noile structuri ale învățămîntului, cercetării, noile realizări ale industriei de c.p. — interzise și ele anterior — am eliminat și am reelaborat, retipărit și tipărit peste 200 pagini, care perfecționează substanțial lucrarea și pregătește primul tiraj pentru întîlnirea cu publicul în mai 1990, sub titlul său real.

cat articole (inclusiv în seria AMC a Editurii Tehnice) și a prezentat comunicări științifice pe această temă, în țară și în străinătate.

MIHAI MĂRȘANU, șef al colectivului de cercetare pentru sisteme cu microcalculatoare din Institutul de Tehnică de Calcul, București. Absolvent al Facultății de Electronică și Telecomunicații din Institutul Politehnic București în 1967. Doctorand. Este autor și coautor al multor lucrări și comunicări științifice destinate specialiștilor din domeniul calculatoarelor, inclusiv „Calculatoarele electronice din generația a cincea”, Editura Academiei, 1985. A participat în 1986 și 1987 la o serie de acțiuni de inițiere a elevilor în domeniul calculatoarelor personale. Preocupări: sistemele multimicro, mașinile de baze de date și optimizarea accesului la date stocate în memorii rotaționale.

IOAN PAUL ZAMFIRESCU, absolvent al Facultății de Electrotehnică din Institutul Politehnic București, doctorand în calculatoare—automatizări. Activitate îndelungată ca editor de cărți și reviste la Editura Tehnică, în domeniile automatică—informatică—electronică—management. Preocupări în analiza și proiectarea asistată de calculator a sistemelor de editare—tipărire—difuzare. Specializare în informatică în Franța și proiecte pentru Hachette — Paris și I.B.M. — France. Autor de articole și de cărți.

EUGEN DOBROVIE, absolvent al Facultății de Automatică din Institutul Politehnic București în 1977. Lucrează la Întreprinderea de Calculatoare Electronice; a făcut parte din colectivele de proiectare ale calculatoarelor M118, HC-85, HC-88; a fost instructor specialist în taberele de informatică pentru elevi.

VICTOR CRISTIAN COSOSCHI, absolvent al Facultății de Automatică, Secția Calculatoare, din Institutul Politehnic București, în 1977. Lucrează de atunci la Întreprinderea de Calculatoare Electronice, contribuind în colectivele de proiectare hardware și software la realizarea sistemelor Felix M18, CUB-01, VDT 40C, HC-85, HC-88.

NICOLAE BADEA, cercetător științific la ICI București. Absolvent (1971) al Facultății de Cibernetică Economică din București; doctorand în domeniul informaticii. Autor de comunicări la manifestări științifice interne și internaționale. A contribuit la proiectarea și implementarea de sisteme și programe informatice pentru fabricație asistată de calculator, proiectare asistată, utilizarea calculatoarelor în conducere, educație. Este coautor la lucrările: *Noile tehnologii de vîrf și societatea* (1983). *Informatizarea societății — un fenomen inevitabil?* Editura Științifică și Enciclopedică 1985, *Inginerie de sistem, automatică și informatică în transporturi* vol. 1 și vol. 2, Editura Tehnică, 1988—1989.

TRAIAN MIHU, director tehnic al Întreprinderii de Calculatoare Electronice, București; absolvent al Institutului Politehnic București, Facultatea Automatică, secția calculatoare (1970). În cadrul Centralei Industriale pentru Electronică și Tehnică de Calcul a coordonat programele de asimilare pentru majoritatea sistemelor de calcul, terminalelor și echipamentelor periferice care sînt în producția actuală.

Prezent cu comunicări științifice, privind echipamente și tehnici de înregistrare magnetică, în cadrul unor sesiuni interne și internaționale.

CONSTANȚIN HĂRĂBOR, absolvent al Facultății de Matematică a Universității din București, în 1970, în prezent profesor la liceul Dimitrie Bolintineanu, București. Preocupări în utilizarea calculatoarelor personale în cercuri, tabere și în concursuri de matematică și informatică. Coautor la culegeri de probleme. Comunicări științifice în țară și în străinătate.

CUPRINS

Volumul 1

— <i>Studiu introductiv</i> (Academician Prof. Nicolae Teodorescu)	V
— Cuprins general (vol. 1 și 2)	XI
— PROLOG—DIALOG—EPILOG (continuă în vol. 2)	XIII
— Cuprins detaliat vol. 1	XXVIII

Partea I: Calculatoare, microcalculatoare și calculatoare personale în țara noastră și pe plan mondial

1

Capitolul 1. Tehnica de calcul și informatica în România. Începuturi, evoluție, aplicații

1

1.1. Contribuții și priorități românești în matematică, cibernetică, electronică, tehnică de calcul și informatică	2	1.3. Industria românească de calculatoare electronice. Informatica ..	12
1.2. Evoluția generațiilor de calculatoare în țara noastră și în lume	5	1.4. Aplicații ale calculatoarelor în societatea modernă	17

Capitolul 2. Caracterizarea și evoluția calculatoarelor personale și microcalculatoarelor

21

2.1. Ce este un calculator personal? ..	21	2.4. Industria românească de calculatoare personale și personal-profesionale	35
2.2. Microelectronica pe plan mondial	24		
2.3. Calculatoare personale, personal-profesionale și microcalculatoare în lume	28		

Partea a II-a: Calculatoare numerice. Realizare fizică — bazele aritmetice și logice

37

<i>Capitolul 3. Baze aritmetice</i>	37		
3.1. Sistemul de numerație binar ..	37	3.7.2. Reprezentarea în complementul față de unu ..	43
3.2. Conversia binar-zecimală	39	3.7.3. Reprezentarea în complementul față de doi ..	44
3.3. Conversia zecimal-binară	39	3.8. Reprezentarea numerelor reale ..	45
3.4. Reprezentările octală și hexazecimală	40	3.8.1. Reprezentarea în virgulă fixă	45
3.5. Codul binar-zecimal (2—10) ..	41	3.8.2. Reprezentarea în virgulă mobilă	46
3.6. Adunarea și înmulțirea numerelor binare	42	3.9. Reprezentarea caracterelor alfanumerice	47
3.7. Reprezentarea numerelor negative	43		
3.7.1. Reprezentarea în modul și semn	43		

Capitolul 4. Logica matematică și circuitele logice	49		
4.1. Generalități	49	4.4.1. Circuite logice combinaționale	54
4.2. Formele canonice ale funcțiilor logice	51	4.4.2. Circuite logice secvențiale	60
4.3. Identități și simplificări	54		
4.4. Realizarea fizică a funcțiilor logice	54		
Partea a III-a: Calculatorul personal HC-85. Structură, componente, operare, programare	66		
Capitolul 5. Structura și modul de operare ale calculatoarelor numerice.....	66		
5.1. Conceptul de calculator	66	5.3. Instrucțiunile calculatoarelor numerice	74
5.2. Structura și operarea calculatoarelor numerice	68	5.4. Software, programare, algoritmi	77
Capitolul 6. Structura și componentele microcalculatorului HC-85	80		
6.1. Configurație	80	6.3. Subsistemul de intrări-ieșiri: interfețe cu televizorul, tastatura, casetofonul, difuzorul: conecturul de extensii	87
6.2. Unitatea centrală: microprocesorul Z 80, memoriile RAM, ROM	81		
Capitolul 7. Elemente de programare în limbaj algoritmic	93		
7.1. Algoritmi.....	93	7.2. Limbajul algoritmic (pseudocod)	94
Partea a IV-a: Programarea în limbajul BASIC pe calculatorul HC-85 ...	103		
Capitolul 8. Caracteristicile și elementele limbajului BASIC	103		
8.1. Noțiuni introductive	103	8.4.2. Constante	114
8.2. Tastatura	105	1°Constante numerice ..	114
8.3. Moduri de lucru	106	2°Constante alfanumerice	116
8.3.1. Modul de lucru K	106	8.4.3. Variabile	116
8.3.2. Modul de lucru L	106	1°Variabile numerice ..	117
8.3.3. Modul de lucru C	108	2°Variabile alfanumerice	118
8.3.4. Modul de lucru E	108	8.4.4. Expresii	119
8.3.5. Modul de lucru G	109	1°Expresii numerice	119
8.4. Alfabetul limbajului BASIC ..	111	2°Expresii alfanumerice	120
8.4.1. Setul de caractere	112	8.4.5. Funcții standard	120
a Litere	112	1°Funcții standard matematice	120
b Cifre	112	2°Funcții standard pe șiruri de caractere	121
c Operatori aritmetici ..	112		
d Operatori de relație ..	112		
e Operatori logici	112		
f Semne de punctuație ..	112		
g Semne speciale	113		
Capitolul 9. Instrucțiunile limbajului BASIC (tratare detaliată cu exemplificări)....	122		
9.1. PRINT	123	9.6. NEW	132
9.2. RUN	129	9.7. LIST	132
9.3. DELETE	130	9.8. PRINT AT	133
9.4. GOTO	131	9.9. PRINT TAB	134
9.5. CLS	131	9.10. REM	135

9.11. LET	135	9.29. CONTINUE	161
9.12. BEEP	137	9.30. DIM	162
9.13. PAUSE	139	9.31. DATA	164
9.14. LOAD	140	9.32. READ	164
9.15. SAVE	141	9.33. RESTORE	165
9.16. VERIFY	142	9.34. DEF FN	166
9.17. MERGE	143	9.35. GOSUB-RETURN	167
9.18. INPUT	143	9.36. PEEK	169
9.19. INPUT LINE	145	9.37. POKE	169
9.20. PLOT	146	9.38. FLASH	170
9.21. DRAW	147	9.39. PAPER	170
9.22. FOR-NEXT	152	9.40. INK	171
9.23. RND	154	9.41. INVERSE	172
9.24. RANDOMIZE	154	9.42. BORDER	172
9.25. CIRCLE	155	9.43. POINT	173
9.26. OVER	156	9.44. ATTR	173
9.27. IF	158	9.45. LPRINT	174
9.28. STOP	160	9.46. LLIST	174
		9.47. COPY	174

Partea a V-a: Programarea în limbajul LOGO pe calculatorul HC-85 175

Capitolul 10. Caracteristicile și elementele limbajului LOGO 175

10.1. Evoluția limbajelor de programare	176	10.7. Reguli privind lucrul cu proceduri	194
10.2. Ce este LOGO? Caracteristici ..	178	Proceduri cu intrări	195
10.3. LOGO în școli. De ce și cum? ..	182	Tipuri de proceduri	196
10.4. LOGO pe HC-85	184	Introducerea și editarea procedurilor	197
10.5. Reguli gramaticale ale limbajului LOGO	186	10.8. Expresii condiționale	198
10.6. Obiectele limbajului LOGO ..	187	Instrucțiunea IF și predicate ..	198
Numere	187	Predicate predefinite	199
Cuvinte	188	Predicate definite de utilizator	199
Liste	189	10.9. Linii complexe și interpretarea lor	200
Delimitatori	190	10.10. Glosar de termeni LOGO	202
Variabile și atribuirea numerelor în LOGO	191		
Variabile locale și globale	193		

Capitolul 11. Primitivile limbajului LOGO, cu exemplificări 207

11.1. Caracteristici generale ale primitivelor	207	11.2.2. Primitive care specifică starea penelului	214
11.2. Primitive LOGO pentru controlul penelului și ecranului în regim grafic	208	HEADING	214
11.2.1. Primitive pentru schimbarea stării penelului		POSITION	215
BACK	209	SHOWNP	215
CLEARSCREEN	210	TOWARDS	215
FORWARD	210	XCOR	216
HIDETURTLE	211	YCOR	216
HOME	211	11.2.3. Primitive pentru utilizarea peniței electronice și a ecranului	216
LEFT	211	CLEAN	216
RIGHT	212	DOT	217
SETHEADING	212	FENCE	217
SETPOS	213	PENDOWN	218
SETX	213	PENERASE	218
SETY	213	PENREVERSE	218
SHOWTURTLE	214	PENUP	219
		SETBG	219

SETBORDER	219	FIRST	239
SETPC	220	LAST	239
SETSCHRUNCH	220	ITEM	239
WINDOW	221	11.5.2. Primitive care concate-	
WRAP	222	nează cuvinte și liste ..	240
11.2.4. Primitive care specifică		FPUT	240
starea peniței și a ecran-		LIST	240
ului	222	LPUT	241
BACKGROUND	222	SENTENCE	241
PENCOLOUR	223	WORD	242
SCRUNCH	223	11.5.3. Primitive care exami-	
11.3. Primitive LOGO pentru contro-		nează cuvinte și liste ..	243
lul ecranului în regim alfa-		ASCII	243
numeric	223	CHAR	244
BRIGHT	224	COUNT	244
CLEAR TEXT	224	EMPTY	244
COPYSCREEN	225	EQUALP	244
CURSOR	225	LISTP	245
FLASH	225	MEMBERP	245
INVERSE	225	NUMBERP	246
NORMAL	226	WORDP	246
OVER	226	11.6. Primitive LOGO care lucrează	
SETCURSOR	227	asupra variabilelor	246
SETTC	227	MAKE	247
TEXTSCREEN	227	NAMEP	248
11.4. Primitive LOGO care specifică		THING	248
operații matematice și logice ..	228	11.7. Primitive LOGO pentru co-	
11.4.1. Operatori aritmetici și		municația cu exteriorul (echipa-	
logici + - * / < >		mentele de intrare / ieșire) ...	248
=	228	11.7.1. Primitive care citesc in-	
11.4.2. Primitive pentru ope-		formații din exterior ..	249
rații matematice	231	KEYP	249
ARCCOS	231	READCHAR	249
ARCCOT	231	READLIST	249
ARCSIN	232	11.7.2. Primitive care afișează	
ARCTAN	232	pe ecranul calculatoru-	
COSINE	232	lui	250
SINE	232	PRINT	250
COTANGENT	232	SHOW	250
DIV	232	TYPE	251
INT	233	11.7.3. Primitive pentru gene-	
PRODUCT	233	rare de sunete	251
RANDOM	233	SOUND	251
REMAINDER	234	11.8. Primitive LOGO care asigură	
ROUND	234	ramificația în program	252
SQRT	234	11.8.1. Primitive pentru rami-	
SUM	235	ficație condiționată ..	252
TANGENT	235	IF	253
11.4.3. Primitive pentru ope-		11.8.2. Primitive pentru întrer-	
rații logice	235	uperea procedurilor ..	253
AND	236	BYE	253
NOT	236	OUTPUT	254
OR	236	STOP	255
11.5. Primitive LOGO pentru lucrul		WAIT	256
cu cuvinte și liste	236	TOPLEVEL	256
11.5.1. Primitive care specifică		11.8.3. Primitive pentru exe-	
preluarea unor elemen-		cuția și repetarea unei	
te din cuvinte sau liste	238	liste de instrucțiuni ..	257
BUTFIRST	238	REPEAT	258
BUTLAST	238	RUN	258

11.9. Primitive pentru lucrul cu fișiere	259	ERN	266
11.9.1. Primitive pentru încărcarea de pe casetă	260	ERNS	267
LOAD	260	ERPS	267
LOADD	260		
LOADSCR	260	11.11. Primitive pentru definirea și modificarea procedurilor	267
11.9.2. Primitive pentru salvarea pe casetă	260	11.11.1. Primitive pentru definirea și editarea procedurilor	267
SAVE	261	EDIT	267
SAVEALL	261	EDNS	269
SAVED	261	TO	269
SAVESCR	261	END	270
11.9.3. Primitive pentru controlul imprimantei și tipărirea informației de pe ecran	261	11.11.2. Primitive pentru modificarea procedurilor sub controlul programului	270
PRINTON	262	COPYDEF	270
PRINTOFF	262	DEFINE	270
COPYSCREEN	262	DEFINEP	271
11.9.4. Primitive pentru lucrul cu microdriverul	262	PRIMITIVEP	271
SETDRIVE	263	TEXT	271
CATALOG	263	11.12. Primitive pentru funcții diverse	272
ERASEFILE	263	11.12.1. Primitive pentru accesul la memoria calculatorului	272
11.10. Primitive pentru gestiunea spațiului de lucru	263	● BLOAD	272
11.10.1. Primitive pentru examinarea spațiului de lucru	264	● BSAVE	272
NODES	264	● CALL	273
RECYCLE	264	● DEPOSIT	273
11.10.2. Primitive pentru afișarea conținutului spațiului de lucru ..	264	● EXAMINE	274
PO	265	● RESERVE	274
POALL	265	● RESERVED	275
PONS	265	11.12.2. Primitive pentru lucrul cu interfața serială	275
POPS	265	● SETSERIAL	275
POTS	265	● SERIALIN	276
11.10.3. Primitive pentru ștergerea informației din spațiul de lucru	266	● SERIALOUT	276
ERALL	266	11.12.3. Primitive pentru specificarea obiectelor și primitivelor LOGO	276
ERASE	266	● CONTENTS	276
		● PRIMITIVES	276
Capitolul 12. Tehnici de programare în LOGO și aplicații	277		
12.1. Proiectarea programelor	277	naturale din baza 10	
12.2. Recursivitatea în LOGO	284	într-o bază mai mică	
12.3. Mici aplicații în LOGO	298	decit 16	303
12.3.1. Exemple grafice	298	12.3.5. Conversia dintr-o bază	
12.3.2. Suma a două numere aleatoare	301	oarecare în altă bază ..	304
12.3.3. Ordonare alfabetică ..	302	12.3.6. Desenează interactiv ..	307
12.3.4. Conversia numerelor		12.3.7. Turnurile din Hanoi	309
		12.4. Mesaje de eroare în LOGO	312

27.09.1984
S. Stancu

CALCULATOARE, MICROCALCULATOARE ȘI CALCULATOARE PERSONALE ÎN ȚARA NOASTRĂ ȘI PE PLAN MONDIAL*

Capitolul 1 | Tehnica de calcul și informatica în România. Începuturi, evoluție, aplicații.

Încă din anii '60, dar și la începutul anilor '70, ilustrul profesor și renumitul matematician Grigore C. Moisil a publicat o serie de articole prin care leagă matematica de știința informaticii, dar și de cultura generală, pledând pentru informatică și învățarea ei. Dintr-unul din aceste articole din 1971, care constituie prefața volumului 13—14 al seriei continue AMC (Automatică, Metrologie, Calculatoare) a Editurii Tehnice, am selectat câteva idei sugestive pentru cei tineri și nu numai pentru ei:

Problema calculatoarelor nu este numai problema construcției lor; ea este în primul rând problema utilizării lor.

Un calculator costă atât de mult, încât orice pierdere de timp din cauză că programarea este neluând în seamă trebuie să fie evitată. Cei ce lucrează la calculator trebuie să fie deosebit de bine instruiți. O cultură matematică insuficientă costă mult. Un calculator utilizat de un om nu îndestul de priceput nu se strică, e mai rău: lucrează cu randament scăzut. E mai rău zic, fiindcă o mașină care stă, se vede că stă; toată lumea o vede că stă. O uzină trebuie să dea o anumite producție; dacă nu o dă, se vede că uzina e prost utilizată. Un calculator care din cauza proastei programări dă numai a zecea parte din ce poate da, merge; nimeni nu-l vede că merge încet: fiindcă el merge la fel de repede, dar treaba pe care o face, e pus s-o facă încet... Pe bună dreptate se spune că construcția unui calculator pune în evidență toată capacitatea tehnică a celor ce îl construiesc, nu numai capacitatea tehnică în construcția calculatoarelor.

* Atragem atenția cititorilor că pentru a ilustra familiile de calculatoare personale și personal-profesionale, ca și aplicațiile lor în învățământ și în alte domenii, cartea cuprinde un grupaj de pagini cu felurite anexe, explicate, care au legătură cu textul din partea I (ca și cu textele din alte părți ale cărții). A se vedea și secțiunea PROLOG-DIALOG-EPILOG-

Dar tot atît de mult utilizarea unui calculator pune în evidență toată capacitatea științifică a celor ce îl întrebuințează. Cunoștințele de logică formală sînt tot atît de esențiale pentru un informatician ca și cele de analiză numerică...

Din planul de învățămînt al specializării în „Știința Calculatoarelor” dat în acest număr se vede că cultura de bază a celui ce trebuie să se ocupe de „software” e aceea a unui matematician instruit în matematica modernă: abstractă, algebrică, formală.

Cei ce organizăm învățămîntul în acest domeniu, cei care predăm cursuri asupra acestor discipline vom medita mult asupra paginilor ce urmează.

...Nu putem crede că această muncă se poate duce fără a ști ce fac ceilalți. Editura tehnică a înțeles acest lucru și numărul cărților traduse și publicate de această editură e mare. După cum noi nu trebuie să credem că un om sau un popor poate progresa lucrînd izolat, fără să știe ce fac ceilalți, tot astfel nu trebuie să ne închipuim că putem cu forțele noastre numai, să acoperim toată producția de cărți pe care e nevoie să le citim. Un număr mare de traduceri de cărți bune, din toate limbile, contribuie la ridicarea nivelului științific. În acest caz și numai în acest fel vom putea și noi să scriem cărți bune, care să fie și ele traduse în alte limbi. În unele domenii această treaptă a fost atinsă. În domeniul Științei Calculului ea trebuie să fie atinsă, dar sîntem foarte la început.

Dar progresul științific și tehnic nu se poate face numai cu cărțile ce le putem edita. Nici un popor nu poate edita toate cărțile de care are nevoie, pe care le citește. O carte tradusă e întotdeauna în urmă cu mulți ani. Dacă cititorul nu-și dă seama de aceasta, atunci el dovedește că e cu mai mulți ani în urma cărții...

Această confruntare a muncii științifice naționale pe plan internațional e absolut necesară.

... Știința calculului nu se dezvoltă însă numai spre beneficiul ingineriei, statisticii și economiei. Lingvistica, medicina, biologia, psihologia, sociologia, arheologia, știința literaturii, toate aceste discipline umanistice cer acces la calculator. Aceasta e marea mirare a vremii noastre. Trebuie deci să pregătim învățămîntul despre calculatoare și pentru studenții facultăților științelor vieții, ale omului și ale societății...

Am fost deosebit de bucuros scriînd și citînd o parte a acestui număr al revistei. Cititorul, mai imparțial decît mine, va compara ce s-a făcut cu ceea ce va trebui să se facă. Va constata că ne stă în față o muncă imensă, ceea ce e înviorător.

Să ne oprim puțin asupra citorva din contribuțiile aduse de profesori și specialiști români de prestigiu în domenii ca: matematica, cibernetică, neurocibernetică, electronica, microelectronica și desigur în cele ale științei calculatoarelor și informaticii.

1.1. Contribuții și priorități românești în matematică, cibernetică, electronică, tehnică de calcul și informatică

MATEMATICA. Intemeietorii școlii matematice moderne românești sînt considerați savantul multilateral *Spiru Haret* (1851—1912), primul român care obține în 1878 titlul de doctor în matematică și *David Emmanuel* (1854—1941), profesor strălucit și al doilea român doctor în matematică.

Dintre iluștrii reprezentanți ai matematicii în România la începutul acestui secol, amintim pe *Tratan Lalescu* (1882—1929), *Dimitrie Pompetu* (1873—1954) și *Gheorghe Țișea* (1873—1939), doctori în matematică, pedagogi și creatori de renume mondial.

Contribuții importante în dezvoltarea matematicii sînt datorate savanților de renume internațional *Miron Nicolescu* (1903—1975), *Simion Stoilow* (1887—1961), *Alexandru Ghica* (1902—1964) și *Alexandru Myller* (1879—1965).

Trebuie menționată contribuția originală a academicienilor *Octav Onicescu* (1892—1983) și *Gheorghe Mihoc* (1906—1981) la dezvoltarea teoriei probabilităților și statisticii matematice, începînd cu conceptul de lanț probabilistic cu legături complete, pe care l-au introdus în anul 1935. Academicianului *Octav Onicescu* îi datorăm, printre altele, primul curs din România de calculul probabilităților (1925), introducerea conceptului de energie informațională (1966), mecanica invariantivă (1954—1964) și contribuțiile la teoria jocurilor (1960).

Meritul de a forma și orienta școala românească de logică matematică cu o largă recunoaștere internațională îi revine academicianului profesor doctor docent *Grigore C. Moisil*

(1906—1973). Prima sa lucrare de logică matematică a fost publicată în țara noastră încă din anul 1936, dar lucrarea sa fundamentală rămâne „*Teoria algebrică a mecanismelor automate*” publicată în 1959 și reeditată în mai multe țări. Din colectivul de cercetători în matematică pe care l-a format, s-a desprins în 1955 grupul de cercetare în domeniul logicii matematice aplicate în tehnică (teoria algebrică a automatelor) și, ceva mai târziu, un grup de cercetare în domeniul logicii matematice cu aplicații în rezolvarea unor probleme economice (programarea pseudo-booleeană). În anul 1957 inițiază primele cursuri de calculatoare electronice, la Facultatea de Matematică din București.

Matematicienii români contemporani au dezvoltat cercetări noi în domeniile variate ale matematicii, în informatică, mai ales în teoria algoritmilor și limbajelor și există preocupări intense în domeniile inteligenței artificiale, lingvisticii matematice, teoriei jocurilor, modelării învățării, modelării matematico-semiotice, simulării, statisticii și teoriei probabilităților.

CIBERNETICA. Doctorul român *Ștefan Odobleja* (1902—1978) este un adevărat precursor al ciberneticii, primul care a evidențiat principiile ciberneticii generalizate în lucrarea sa în două volume „*Psihologia consonantistă*”, publicată în limba franceză în 1938—1939 și difuzată în rîndul specialiștilor, inclusiv în străinătate.

El generalizează fenomenul circularității și stabilește printre altele, legea reversibilității în știință, psihologie, economie și societate (cercul vicios de acțiuni și reacțiuni reciproce, concept similar cu ceea ce numim azi bucla de reacție cibernetică). Odobleja preconiza gândirea artificială și mașinile cu gândire.

Alți doi români pot fi considerați precursori ai ciberneticii. Academicianul medic *Daniel Danielopolu* (1884—1955) a folosit în 1928 conceptul de mecanism circular, explicînd fenomenele de reglare în organismul uman și a evidențiat procesele biologice informaționale, devenind un ilustru precursor român al biociberneticii. În rînd independent, prof. dr. ing. Paul Postelnicu a generalizat ulterior procesele electronice cu reacție (conexiunea inversă sau complexul vicios, după denumirea dată inițial) privind nașterea și evoluția viețuitoarelor. Lucrările profesorului *Paul Postelnicu* au fost elaborate în anul 1944, dar trimise pentru publicare, nu au fost acceptate, astfel încît ele au apărut abia în 1968 și 1979.

Dar, pe plan mondial, *Norbert Wiener*, rămîne acreditat cu introducerea termenului de cibernetică și cu fundamentarea principiilor de bază, deși a făcut aceasta mult mai târziu, în anul 1948, cînd a publicat în S.U.A. cartea „*Cybernetics or Control and Communication in the Animal and the Machine*”.

Astfel, România apare în mod evident, ca una din țările care a contribuit la fundamentarea ciberneticii.

O serie de profesori români contemporani aduc contribuții importante la dezvoltarea ciberneticii tehnice, socio-economice, la aprofundarea relațiilor dintre informatică, cibernetică, revoluția tehnico-științifică și informatizarea societății.

Cercetări în domeniul neurociberneticii au efectuat academician *Constantin I. Parhon* (1874—1969), profesor doctor inginer *Gheorghe Cartianu* (1907—1982) și alți distinși profesori, ingineri, medici și matematicieni contemporani.

ELECTRONICA. În perioada 1900—1960 o serie de academicieni, profesori și inventatori din țara noastră au pus bazele școlii științifice românești în domeniul electronicii. Astfel, s-au remarcat, prin contribuții originale în lucrările lor științifice, tehnice și didactice:

— *Dragomir Hurmuzescu* (1865—1954) în electricitate, ca inventator al electroscopului în 1894 și în radiocomunicații cu primul radioreceptor și primele radioemitoare din țara noastră, construite în 1925, respectiv 1927 și 1928 la Institutul Electrotehnic de pe lângă Universitatea București.

— *Mihai Konteschweller* (1897—1947) în radiofonie și mai ales în telemecanică, domeniu în care a realizat un prim experiment în 1934, publicînd apoi primul tratat de telemecanică în 1937.

— *Ion Constantinescu* (1884—1963) în telecomunicații, cu primul curs, publicat ca profesor la Institutul Politehnic din București încă din 1925 și în rețele telefonice, primul studiu pentru țara noastră fiind elaborat în 1945.

— *Augustin Maior* (1882—1964), inventatorul și experimentatorul telefoniei multiple care reușește să transmită, în 1906, cinci convorbiri pe un singur circuit telefonic, pe o linie de 15 km, rezultat publicat în 1907.

— **Tudor Tăndăscu** (1901—1961), profesor la Institutul Politehnic București, unde a organizat laboratoarele de radioelectronică și apoi a contribuit la înființarea Facultății de Electronică, poate fi considerat întemeietorul școlii românești moderne de electronică. A publicat primele cursuri de radiocomunicații (1930), de tuburi și circuite electronice (1955) și apoi de tranzistoare (1961). A dezvoltat cercetări în domeniul circuitelor electronice (amplificatoare de putere clasa C, stabilitatea circuitelor și oscilatoarelor), al electronicii industriale și nucleare, al automaticii. Profesorul **Tudor Tăndăscu** a fost primul doctor în electronică din România (1940) și a scris primul studiu din țara noastră despre calculatoarele electronice.

— **Gheorghe Cartianu** (1907—1982), remarcabil profesor la Institutul Politehnic din București, membru corespondent al Academiei din 1964, este considerat creatorul școlii românești moderne de radiotehnică și radiocomunicații. A fost asistent al profesorului **Tudor Tăndăscu** din 1934 și ulterior, timp de 25 de ani începând din 1952, șeful catedrei de radiocomunicații din cadrul Facultății de Electronică și Telecomunicații din București. Profesorul **Gheorghe Cartianu** a efectuat cercetări teoretice și experimentale, privind mai ales modulația de frecvență, comunicațiile, circuitele electronice și cibernetica.

În ultimii 25—30 de ani, profesorii și inginerii de mare prestigiu au condus în continuare activitățile de cercetare și didactice din domeniul electronicii, privind dispozitivele electronice, mai ales dispozitivele semiconductoare și circuitele integrate, electronica funcțională, modelarea și simularea dispozitivelor semiconductoare, circuitele cu impulsuri, automatale secvențiale, sinteza rețelilor electrice, electronica profesională (aparate), radioelectronica, antenele, electroacustica, televiziunea, teoria și aplicațiile transmisiunii informațiilor.

Dezvoltarea electronicii în țara noastră trebuie privită și în corelare cu rezultatele teoretice și experimentale remarcabile, obținute în domeniile înrudite ale fizicii, electrotehnicii și automaticii.

Primele institute de cercetare pentru electronică din România au fost înființate în 1966 și 1968, primul având ca profil electronica profesională de radiocomunicații, aparatura electronică de măsură și electronică medicală, iar cel de al doilea vizând componentele electronice.

TEHNICA DE CALCUL ȘI INFORMATICA. Evoluția tehnicii de calcul și informaticii în România se înscrie în contextul existenței și dezvoltării unor serii importante de contribuții originale românești în domeniile matematicii, ciberneticii, electronicii și automaticii.

Primele calculatoare electronice din România au fost construite la București, Cluj-Napoca și Timișoara.

În anul 1953 începe elaborarea *primului calculator românesc CIFA-1*, de către un colectiv de specialiști* de la Institutul de Fizică Atomică din București. Calculatorul CIFA-1 a fost pus în funcțiune în anul 1957, după care au urmat CIFA-2 (în 1959), CIFA-3 (în 1961) și CIFA-4 (în 1962). CIFA-1 reprezintă totodată primul calculator elaborat în țările socialiste după realizările din U.R.S.S., deci o replică rapidă dată de România la numai câțiva ani de la apariția primelor calculatoare din generația I-a pe plan mondial. De altfel, în colaborare cu inginerii români, inginerii din Bulgaria au construit un calculator similar pe baza documentației calculatorului CIFA-1 și a specializărilor pentru acest produs, calculator denumit inițial SEMBÂN și expus la Moscova în 1963, la Expoziția Națională a R. P. Bulgaria.

Alt colectiv** de la IFA elaborează în anii 1959—1964, calculatoarele *CIFA-101 (1962)* și *CIFA-102 (1964)*, tot din generația I, calculatoare ce au beneficiat de numeroase inovații tehnologice și de arhitectură, CIFA-102 fiind implementat în mai multe centre de calcul.

Apoi, colectivul care se afirmase cu modelele CIFA 1—4, realizează în cadrul IFA și modelele de calculatoare din generația a doua, *CET-500 (1964)* și *CET-501 (1966)*.

Tot la IFA, s-a dezvoltat în anii '60 un colectiv puternic pentru programe de sistem și programe aplicative.

În București s-au mai realizat și alte calculatoare electronice numerice din generațiile 1 și 2, cât și calculatoare electronice analogice***.

La Institutul de calcul din Cluj-Napoca, înființat în 1957 de academician Tiberiu Popoviciu, se construiește în 1959 calculatorul experimental cu relee MARICCA, iar apoi un colectiv de cercetare realizează calculatoarele numerice din generația a 2-a, *DACCIC-1 (1961)* și *DACCIC-2 (1969)*.

DACCIC-1 a fost primul calculator românesc tranzistorizat, cu memorie internă cu ferite, memorie externă cu tambur și programabil în limbajul FORTRAN, aparținând generației a 2-a.

DACCIC-200, construit în numai doi ani (1967—1969), a fost cel mai rapid calculator românesc (200000 operații/s) până în anii '70 și primul care a dispus de un sistem de ope-

* Cordonator dr. ing. V. Toma; ** Cordonatori ing. A. Segal și ing. E. Clobanu; *** MECAN la Institutul de Energetică al Academiei (coordonator prof. V. M. Popov), MAC-1 la Institutul Politehnic București (colectiv dr. ing. A. Petrescu, ing. P. Dimo și ing. I. Șipoș) și unele modele la Academia Militară din București.

rare de concepție proprie. DACCIC-200 aparținea tehnologic generației a 2-a, dar încorporea numeroase concepte ce se regăseau la generația a 3-a, deci din nou o reușită românească la numai cinci ani după lansarea generației a 3-a, pe plan mondial, în 1964.

La Institutul Politehnic Timișoara, sub coordonarea mai multor cadre didactice, se construiesc *calculatoarele** *MECIPT-1 (1962) din generația 1 și MECIPT-2 (1966) din generația a 2-a.*

MECIPT-1, cu tuburi electronice, era similar calculatoarelor CIFA, iar MECIPT-2 a fost primul calculator românesc la care s-a aplicat principiul microprogramării. La elaborarea calculatoarelor MECIPT, la perfecționarea acestora de-a lungul mai multor ani și programarea lor, au adus contribuții o serie largă de cadre didactice, cercetători și studenți.

La Cluj-Napoca și Timișoara s-au înființat, în 1970, primele *Centre teritoriale de calcul electronic din țara noastră.*

1.2. Evoluția generațiilor de calculatoare pe plan mondial și în țara noastră

Ce evenimente au marcat însă, evoluția calculatoarelor în lume?

Pe plan mondial, *primul calculator electronic digital, denumit ENIAC (Electronic Numerical Integrator And Calculator)*, a fost construit la Universitatea din Pennsylvania, în intervalul 1942—1945.

ENIAC avea în structura sa aproape 20000 de tuburi electronice, ocupa o suprafață de 160 m², avea o greutate de cca. 30 de tone și o viteză de prelucrare a datelor de maximum 5000 de adunări și scăderi pe secundă.

Cam în aceeași perioadă (1939—1944), la Universitatea Harvard se realizase în colaborare cu inginerii de la firma IBM mașina de calcul automată Mark I, dar care utiliza relee și benzi de hirtie perforată.

Construirea calculatorului electronic ENIAC a fost facilitată de elaborarea *mașinii de calcul cu relee MARK I*, de cercetările teoretice ale lui Turing și de descoperirea circuitelor electronice de numărare, bazate pe tuburi. Despre ENIAC, contemporanii spuneau că „mașina este așa de mare că de fiecare dată când i se dă drumul se ard două tuburi“. Introducerea datelor și a instrucțiunilor se făcea de la 40 de panouri cu fișe, introducerea programului dura câteva zile, iar verificarea alte câteva zile. Oricum, programele se rula de câte, două ori, introducând din când în când date de testare pentru găsirea tuburilor arse.

În 1946/1947, *John von Neumann* publica în S.U.A. proiectul primului calculator cu program memorat, cu prelucrare secvențială a instrucțiunilor și datelor, memorate împreună, cu aceeași formă și accesibile în același mod, denumit *EDVAC (Electronic Discrete Variable Computer)*. „Principiul von Neumann“ caracterizează aproape toate calculatoarele produse până astăzi, acestea fiind numite calculatoare de tip von Neumann**.

Dar prima aplicare a noului principiu din 1946 este datorată profesorului *M. Wilkes* de la Universitatea Cambridge din Anglia, care a construit, în 1949, primul calculator din lume cu program stocat într-o memorie cu întârziere — *EDSAC (Electronic Delay Storage Automatic Calculator)*.

Primul calculator comercial a fost livrat de firma UNIVAC la începutul anilor '50.

Specialiștii consideră că în anul 1953 a fost lansată comercial generația I de calculatoare electronice, în același an începând și cercetările originale românești în acest domeniu, finalizate în 1957, într-o primă etapă, așa cum s-a arătat mai sus, prin modelul CIFA-1, deci după numai patru-cinci ani!

Prezentăm mai jos câteva caracteristici care se pot asocia celor cinci generații de calculatoare ce s-au succedat pe plan mondial până în prezent, cu mențiunea că generația a cincea este în curs de elaborare și finalizare la orizontul anilor '90.

* Cu participarea inițială a specialiștilor D. Kaufmann și M. Löwenfeld

** Cele mai recente modele, care nu mai respectă acest principiu sînt grupate în *clasa non von Neumann*.

● *Generația I* a cuprins intervalul 1953—1958, (sau după unele criterii extinse perioada 1946—1956), calculatoarele având următoarele caracteristici:

- tehnologie cu tuburi electronice;
- arhitectură cu un singur dispozitiv de memorie de mică capacitate și timp de acces mare, de regulă de tip tambur magnetic, utilizat ca memorie internă
- viteza de calcul de câteva mii de operații/sec. (max. 10 mii operații/sec.)
- programe în cod mașină și în limbaje de asamblare.

● *Generația a 2-a*, 1957/1959-1963/1964, se diferențiază prin:

- tehnologie cu tranzistoare și diode semiconductoare;
- memorie internă, de capacitate sporită (max. 32 koct.) și cu timp de acces mult mai mic, bazată pe inele (toruri mici) de ferită;
- apariția memoriei magnetice externe de tip tambur, disc sau bandă magnetică, în general de mică capacitate;
- viteza de calcul sporită, până la cca. 100—200 mii operații/sec.
- îmbunătățirea regimului de lucru al sistemului, astfel încât perifericele (echipamente cu cartele perforate, imprimante rapide) pot lucra simultan cu unitatea centrală;
- apariția primelor sisteme de operare;
- apariția limbajelor de programare de nivel înalt cum sînt: FORTRAN (Formula TRANslator) lansat de IBM în 1955, cu varianta FORTRAN II în 1958, limbaj destinat calculului tehnico-științific, apoi COBOL (Common Business Oriented Language) — 1960, pentru aplicații economice cu prelucrări nenumerice și ALGOL (ALGOrithmic Language) — 1960, limbaj algoritmic care facilitează prelucrările numerice, ca și FORTRAN-ul.

● *Generația a 3-a*, după 1964, (care după sursa* citată cuprinde intervalul 1964—1981), se caracterizează prin:

- tehnologie cu circuite integrate;
- memorii interne semiconductoare cu performanțe sporite, cu capacitate până la 2 Megaocteți;
- la cele două mari părți distincte ale calculatoarelor din generațiile 1 și 2, hardware (circuite electronice, echipamente) și software (programe de sistem, programe aplicative), se adaugă și a 3-a parte denumită firmware (care conține microprogramele înscrise în memorii fixe);
- discuri magnetice de medie și mare capacitate;
- apariția sistemelor de operare evaluate;
- viteza de calcul apreciabilă, în plaja 0,5—5 MIPS (milioane instrucțiuni pe secundă);
- multiprogramare;
- dezvoltarea limbajelor de programare de nivel foarte înalt: PL/1, PASCAL, LISP, FORTRAN 77 (derivat din FORTRAN IV), BASIC, C și limbajele grafice, completează gama limbajelor anterioare FORTRAN IV — 1962, FORTRAN standard — 1966 și ALGOL-68. Limbajul PASCAL-1971, cu o serie de versiuni ulterioare, facilitează aplicarea principiilor programării structurate. Limbajul LISP-1960, (LIST Processing Language) cu toate dezvoltările care au urmat, este orientat pe prelucrarea datelor reprezentate prin liste, fiind primul limbaj al domeniului inteligenței artificiale. Limbajul BASIC-1975 (Beginner's All-purpose Symbolic Instruction Code) este cel mai răspândit limbaj interactiv.

● *Generația 3,5—4*, în perioada 1982—1989, implică o serie de progrese tehnologice și arhitecturale:

- circuite integrate pe scară largă și foarte largă (ajungînd la 1 milion de tranzistoare per circuit integrat);
- memorii interne cu circuite integrate, avînd o mare capacitate de memorare (8 Moct.) și un timp redus de acces;
- memorii externe perfecționate, care permit reducerea timpului de acces la informațiile stocate pe suporturi magnetice, unitățile de discuri magnetice devenînd dominante în majoritatea configurațiilor (totodată apar discurile optice de mare capacitate);
- sisteme de operare perfecționate, orientate pe lucrul cu discurile magnetice;
- creșterea ponderilor valorice pentru software și firmware față de hardware;
- creșterea fiabilității sistemelor;
- viteza de prelucrare crește sensibil, ajungînd pînă la 30 MIPS;

* Mihai Drăgănescu — „Informatica și societatea”, București, 1987.

- generalizarea teleprelucrării, modulul de lucru în regim interactiv de la terminal, dezvoltarea facilităților de (inter)conectare în rețele de calculatoare;
- creșterea ponderii implementărilor de sisteme de gestiune a bazelor de date;
- dezvoltarea unor noi limbaje de programare numite concurente, cum sînt: PASCAL CONCURENT (1975), MODULA-2 (1979), ADA (1980) și EDISON (1980), derivînd din PASCAL. Principiul prelucrării concurente presupune existența simultană a mai multor programe active (sau părți, module, „sarcini”) din program, care solicitînd

aceleași resurse de calcul, pot fi executate în paralel, ținînd seama de interacțiunile — colaborarea prin comunicare și sincronizare reciprocă — dintre ele. Limbajul ADA este un limbaj algoritmic modern, care include facilitățile clasice de tip PASCAL, oferînd posibilități noi de definire a tipurilor de date corelate cu aplicația, de simplificare a structurilor de control, de modularizare a programelor și de tratare a aplicațiilor în timp real.

Un nou limbaj, influențat de LISP și diferit de limbajele convenționale îl reprezintă FORTH (versiunile 1979 și 1983).

● *Generația a 5-a* de calculatoare, a cărei elaborare a început după 1980, fiind prevăzute pe plan mondial obiective eșalonate pînă la începutul anilor 1990, se distinge substanțial de celelalte generații, urmărind nu atît creșterea vitezei de calcul numeric (pînă la miliarde de operații/sec.), ci mai ales dezvoltarea capacității calculatoarelor de a efectua raționamente, într-un mod apropiat de modul în care raționează experții umani, manipulînd baze de cunoștințe.

În acest sens, dezvoltarea deosebită a domeniului inteligenței artificiale, a programării logice (limbajul PROLOG), corelat cu arhitecturile de tip paralel (prin care se renunță la conceptele von Neumann) și progresele tehnologice impresionante care se întrevăd în vecinătatea anului 1990 (circuitul VLSI — cu integrare pe scară foarte largă, cu peste 10 milioane de elemente active pe pastila de siliciu, inclusiv circuite tridimensionale, superdiscuri magnetice și optice cu capacități de memorare de ordinul Gigaocteților și timpi reduși de acces, la costuri tot mai mici), vor permite construirea calculatoarelor mari, mini sau de tip personal, din generația a 5-a, calculatoare suport ale sistemelor expert pentru cele mai variate domenii social-economice. Vor fi implementate limbaje naturale și vor exista facilități pentru dialog prin voce și recunoaștere de forme.

În prezent, au loc pe plan mondial experimente vizînd realizarea calculatoarelor începînd cu generația a 6-a, pe baza progreselor determinante obținute în opto-electronică și mai ales în bioelectronică*.

Ținînd seama de criteriile de mai sus, se prezintă comparativ, în tabelul 1.1, pentru prima dată în acest mod, principalele caracteristici ale citorva modele reprezentative de calculatoare în România în perioada 1957—1969 și ale modelelor anilor 1970—1989, evidențînd astfel evoluția generațiilor de calculatoare în țara noastră. Iată o distribuție aproape completă a calculatoarelor românești pe generații:

- generația 1: CIFA 1 + 4, CIFA 101/102, MARICCA, MECIPT-1
- generația a 2-a: CET 500/501, MECIPT-2, DACCIC-1, 2 și 200
- generația a 3-a: FELIX C-256, FELIX C-32, FELIX C-512/1024
- generația 3.5: seriile INDEPENDENT și CORAL, FELIX C-5000.

Microcalculatoarele evaluate românești bazate pe structuri cu microprocesor de 16 biți aparțin, conform criteriilor de mai sus, generației 3.5 — 4 și vor fi prezentate distinct, în capitolul 25.

Din tabelul 1.1, se constată evoluția deosebit de dinamică a performanțelor calculatoarelor românești în cei 30 de ani, cu o rată similară celei medii pe plan mondial și anume de la calculatoarele cu tuburi electronice (cu o viteză de prelucrare de 50—2000 operații/s., memorie internă mică de tip tambur cu 2—16 kiloocteți, lector lent de bandă perforată, 16/32 instrucțiuni, programare în cod mașină și o putere consumată de 5 kW), la minicalculatoarele actuale cu circuite integrate de memorare și microprocesoare (cu o viteză de prelucrare de 0,5—2,5 milioane de operații/sec., memorie internă de 1—4 Megaocteți, periferice și terminale, inclusiv unități de discuri de mare capacitate, 200 de instrucțiuni, sistem de operare evoluat, compilatoare pentru limbaje de nivel înalt, limbaj de asamblare, pachete extinse de programe aplicative, sisteme de gestiune a bazelor de date), minicalculatorul avînd la aceste performanțe, numai jumătate din puterea consumată la modelele din prima generație.

* Mihai Drăgănescu — „Actualitate și perspectivă în bioelectronică și electronică moleculară”, în Buletinul Român de Informatică și Tehnică de Calcul, vol. VIII, nr. 4 din 1987, buletin editat de I.T.C.I. București.

EVOLUTIA GENERAȚIILOR DE CALCULATOARE ȘI TELECOMUNICAȚIILOR PE PLAN MONDIAL

Generația Perioada	1 1946—56	2 1957—63	3 1964—81	3,5÷4 1982—89	5 1990—
	1	2	3	4	5
	2	3	4	5	6
Hardware calculatoare	Relee Tuburi electronice Tambur magnetic Tub catodic	Tranzistoare Memorii cu ferite	Circuite integrate Memorii semiconductoare Microprocesoare Discuri magnetice Minicalculatoare	Sisteme distribuite VLSI Discuri optice Microcalculatoare de 16/32 biți Minisupercalculatoare Supercalculatoare	Tehnici evolute de împachetare și interconectare Integrare pe scara ultra largă (ULSI) Protectarea circuitelor integrate 3-D Tehnologii Ga-As și Josephson Componente optice Arhitecturi paralele pentru prelucrarea inferențelor
Software calculatoare	Programe cablate Cod mașină Autocod	Limbaje de nivel înalt (ALGOL, FORTRAN, etc)	Limbaje de nivel foarte înalt Sisteme de operare orientate pe limbajul Pascal Programare structurată LISP Timp partajat Grafică pe calculator Baze de date	ADA Pachete de programe de largă utilizare Sisteme expert Limbaje orientate pe obiecte Baze de date relaționale	Limbaje concurente Programare funcțională Prelucrare simbolică (limbaje naturale, recunoașterea formelor: vedere/vorbire, planificare) PROLOG Baze de cunoștințe Sisteme expert evaluate

Tabel 1.2. (continuare)

1	2	3	4	5	6
Exemple de calculatoare	Eniac, Edvac, Univac, IBM 650 CIFA 1 + 4 CIFA 101/102 MARICCA MECIPT-1	NCR 501 IBM 7094 CDC 6600 DACICC-1/2 CET 500/501 MECIPT-2 DACICC-200	IBM360-370 PDP-11 Spectra 70 Honeywell 200 Cray-1 Illiac-IV Cyber-205 RIAD-1* RIAD-2* Felix C-256 Felix C-32 Felix C-512/1024 Independent 100 Independent 102F Coral 4001 Coral 4030	IBM 43XX VAX-11/7XX IBM-308X Amdhal 80 RIAD-3* Coral 4021 Independent 106 Felix C-5000 Coral 8730	Proiect al Japoniei și alte proiecte noi elaborate în unele țări sau grupuri de țări
Tehnologia telecomunicațiilor	Telefon Teletype	Transmisiuni numerice Modulație în coduri de impulsuri	Comunicații prin satelit Microunde Rețele Fibre optice Comutare de pachete	Rețele integrate de comunicații numerice (digitale)	Dezvoltarea extensivă a sistemelor distribuite, modularitate Fuzionarea tehnologiilor comunicațiilor și calculatoarelor
Performanțele calculatoarelor	Memorii de 2 koct Viteza de operare: 10000 instr./s	Memorii de 32 koct Viteza de operare: 200000 instr./s	Memorii de 2 Moct Viteza de operare: 5 mil. instr./s	Memorii de 8 Moct Viteza de operare: 30 mil. instr./s	Viteza de operare: 1 Giga instr./s + 1000 Giga instr./s (1 G = 1000 Mil.)

* RIAD 1, 2 și 3 reprezintă familii de calculatoare compatibile, elaborate și produse în unele țări membre ale CAER

În tabelul 1.2. se prezintă evoluțiile generațiilor de calculatoare și telecomunicațiilor pe plan mondial, ilustrând caracteristicile tehnice pentru cele 5 generații definite pînă în prezent, referitor la hardware (tehnologia calculatoarelor, configurații), software (limbaje, sisteme de operare, aplicații), telecomunicații și performanțele calculatoarelor. Alături de modelele de calculatoare românești sînt incluse și cîteva exemple de calculatoare și familii de calculatoare cunoscute pe plan mondial.

1.3. Industria românească de calculatoare electronice. Informatica

Gînditori și economiști români au militat pentru crearea unei industrii naționale, apoi pentru industrializarea țării, încă de la finele secolului trecut și de la începutul acestui secol, creîndu-se o tradiție progresistă a gîndirii românești.

Electronica, domeniu industrial ce marchează profund societatea omnească în acest secol, se poate considera că s-a dezvoltat în România cu precădere după cel de-al doilea război mondial, în două etape.

În *prima etapă*, pînă în cea de-a doua jumătate a deceniului șapte, asistăm la edificarea unei industrii orientate mai ales pe producția de bunuri de larg consum: receptoare radio, TV etc.

În *etapa următoare* au fost realizate de către stat o serie de investiții care, în principiu, au avut ca scop realizarea unei baze de componente electronice pasive și active (rezistori, condensatori, tranzistori, circuite integrate ș.a.), cît și implementarea unui transfer de tehnologie ce a vizat producția de aparatură electronică de automatizare și tehnică de calcul.

La început, pe bază de licențe, au fost asimilate familii de echipamente de automatizare, cît și echipamente de tehnică de calcul de capacitate mică și medie. S-a urmărit, astfel, atît echiparea industriei cu aparatură de automatizare, cît și introducerea unei evidențe economice, în condițiile încercării de realizare a unui sistem informațional economic, a folosirii calculatoarelor în proiectare, transporturi etc.

Deși, în mare, deciziile privind susținerea unor asemenea domenii, la prima vedere, par a fi corecte sub aspect economic, ele au fost într-o măsură hotărîtoare influențate de factorul politic. Aceasta a condus la soluții neeconomice, fără a se ține seamă de tendințele pe plan mondial și de studiile elaborate de specialiști, România plătind, în condițiile unor perspective limitate, un preț deosebit de mare pentru tot ceea ce a realizat.

În deceniul opt alocațiile valutare au fost din ce în ce mai reduse, fapt care a influențat negativ calitatea și competitivitatea produselor românești. Hotărîrile au fost luate de factorii politici, în atmosfera conducerii întregii economii pe bază de indicații, orientări și soluții aberante.

În continuare se prezintă unele date privind evoluția domeniului tehnicii de calcul în țara noastră.

Trebuie arătat că în *învățămîntul superior*, chiar din 1972, s-a revenit asupra nomenclatorului de specialități, specialitatea calculatoare dispărînd. Această decizie aberantă a influențat în mod negativ pregătirea specialiștilor în domeniul tehnicii de calcul, ca de altfel și în alte specialități de vîrf. Cadrele didactice au fost obligate să recurgă la numeroase artificii, pentru asigurarea predării cunoștințelor de specialitate, în condițiile unui trunchi comun de discipline cu caracter general și lateral, care ocupa mai bine de 65—70 la sută din întregul buget de timp al planului de învățămînt.

La 1 noiembrie 1967 a fost înființată *Comisia Guvernamentală pentru dotarea cu echipamente de calcul și automatizarea prelucrării datelor* (desființată după câțiva ani), iar în trimestrul I al anului 1968, s-a înființat *Institutul de Cercetare și Proiectare pentru Utilaj Electronic de Calcul (ICPUEC)* prin reunirea principalelor grupuri de cercetare din București, Cluj-Napoca și Timișoara.

În anii 1968—1970, s-au înființat Institutul Central de Informatică (ICI), Întreprinderea pentru Întreținerea și Repararea Utilajelor de Calcul (IIRUC), Întreprinderea de Calculatoare Electronice (ICE), Întreprinderea de comerț exterior de specialitate s-a extins profilul IPRS. — Băneasa în domeniul producției de circuite integrate T.T.L.*

Apoi, în perioada 1973—1975 s-au constituit *Societatea Mixtă ROMCONTROL DATA (RCD) pentru unele tipuri de echipamente periferice, Fabrica de Memorie Electronice și Componente pentru Tehnica de Calcul (FMECTC — Timișoara) și Întreprinderea de Echipamente Periferice (IEPER)*, fiind astfel conturată, până în 1975, structura de cercetare, producție și service a domeniului tehnicii de calcul și informaticii și consolidată într-o mică măsură baza inițială pentru industria de componente.

Anii 1967—1975 au constituit perioada de organizare a informaticii prin concentrarea acestei activități în centrele de calcul teritoriale, departamentale și uzinale, înființarea unor centre de calcul noi și dotarea acestora având loc pe măsura dezvoltării producției proprii.

După 1980 Microelectronica București a început să producă circuite integrate tip MOS**

Treptat au fost înființate și dezvoltate o serie de unități de pregătire și specializare a cadrelor, ori au primit astfel de sarcini unele unități existente, menționând astfel: *CEPECA — Centrul de perfecționare a cadrelor, Centrul de Calcul economic și cibernetică economică de la ASE București, Centrul de calcul al Universității București, secțiile de specialitate în diferite facultăți ale institutelor de învățământ superior din București, Timișoara, Cluj-Napoca și Iași, respectiv liceele cu profil de informatică din București, Cluj-Napoca, Iași, Timișoara, Brașov și Petroșani.*

De menționat că în anul 1985 a fost înființat *Institutul de Cercetare Științifică și Inginerie Tehnologică pentru Tehnica de Calcul și Informatică (ITCI — București)* prin unirea formală a resurselor umane și materiale existente în domeniul cercetărilor de profil, ceea ce a condus în ianuarie 1990 la adoptarea deciziei juste de descentralizare prin organizarea a trei institute, cu domenii de activitate clar specificate.

* T.T.L. = logică tranzistor — tranzistor

** MOS = metal — oxid — semiconductor

Toate aceste întreprinderi și institute de profil își desfășoară activitatea în cadrul *Ministerului Industriei Electrotehnicii, Electronicii și Informaticii* (nou înființat după Revoluția din Decembrie 1989), în afară de Institutul de Cercetări pentru Informatică, centrele teritoriale de calcul și alte unități ce depind de noua Comisie Națională pentru Informatică, dependentă direct de Consiliul de Miniștri.

Dintre evenimentele și tendințele apărute în acești 20 de ani în tehnica de calcul românească, menționăm câteva mai importante (suplimentar față de cele date în tabelul 1.3, în care se ilustrează evoluția produselor noi din domeniile electronicii și tehnicii de calcul) și anume:

- creșterea lentă a producției fizice de calculatoare, înnoirea bazată numai pe cercetări proprii, a structurii producției destinate atât dotării interne, cât și exportului;
- organizarea și extinderea Bibliotecii Naționale de Programe la Institutul Central de Informatică, dezvoltarea rețelei de centre de calcul teritoriale și uzinale;
- transformarea țării noastre în țară exportatoare de tehnică de calcul începând cu anul 1978, de când s-au exportat peste 200 de minicalculatoare INDEPENDENT și CORAL, în țări socialiste și capitaliste, inclusiv ca sisteme de aplicații la cheie;
- dezvoltarea, în special în ultimii 5 ani, a unei familii de modele de calculatoare personale, inclusiv personal-profesionale, fabricate însă în serii total insuficiente față de cerințe.

La realizările industriei de tehnică de calcul și informaticii românești, (pe care nu le vom prezenta în detaliu în această carte), în contextul dat, *un aport însemnat l-a avut și îl are în continuare învățământul superior și*, în principal, Institutul Politehnic București prin Facultatea de Automatică (Catedra de Calculatoare) și Facultatea de Electronică.

Profesori și șefi de lucrări de la Catedra de Calculatoare din IPB au contribuit pe baza unor inițiative proprii, la dezvoltarea unor domenii, la proiectarea și asimilarea unor produse noi, cum sînt:

- calculatoare personale și personal-profesionale (de 8 și 16 biți) compatibile cu alte tipuri larg răspândite pe plan mondial (asimilate în colaborare cu Întreprinderea de Calculatoare Electronice);
- terminale inteligente, display-uri grafice (în curs de asimilare la Întreprinderea de Calculatoare Electronice);
- inteligență artificială și sisteme expert în diferite domenii (proiectare de mașini, medicină etc.);
- baze de date relaționale;
- compilatoare noi și instrumente software;
- proiectarea circuitelor VLSI* și a noulor arhitecturi de calculatoare;

* VLSI = circuite integrate pe scară foarte largă (very large scale integration)

**EVOLUȚIA ELECTRONICII ȘI TEHNICII DE CALCUL ÎN ROMÂNIA ILUSTRATĂ
PRIN CÎTEVA PRODUSE REPREZENTATIVE**

Anul orientativ al lansării produsului	Tipul
Bunuri de larg consum	
1949	Radioreceptor cu tuburi electronice
1960/1961	Radioreceptoare tranzistorizate
1961	Televizoare alb/negru cu tuburi electronice
1972/1974	Televizoare alb/negru cu tranzistoare și circuite integrate
1984	Televizoare color cu circuite integrate
Componente electronice active	
1961	Tranzistoare cu germaniu
1969/1970	Circuite integrate TTL
1980/1981	Circuite integrate MOS și CMOS
1985/1986	Microprocesoare de 8 biți și memorii integrate în tehnologie MOS
Calculatoare electronice	
1957	CIFA-1, primul calculator electronic paralel cu tuburi
1962	CIFA-101 primul calculator electronic serial cu tuburi
1964	CET 500, primul calculator electronic tranzistorizat
1969	DACCIC 200, primul calculator electronic tranzistorizat de mare viteză
1969	Calculatoare de birou respectiv mașini de facturat și contabilizat tranzistorizate pe baza licenței FRIDEN-SINGER
1970	FELIX C-256, primul calculator electronic cu circuite integrate produs în serie pe baza licenței firmei CII — Franța (modelul IRIS-50)
1973	Blocuri și module de memorie internă cu ferite
1974	FELIX C-32, primul calculator electronic de concepție proprie, dezvoltat pe baza tehnologiei modelului C 256
1975/1976	Calculatoare de facturat și contabilizat FC-16, 32, 64 și 128, cu circuite integrate, dezvoltate prin concepție proprie pe baza modelelor anterioare
1976/1977	FELIX C-512/1024, al doilea calculator electronic de concepție proprie dezvoltat pe baza tehnologiei modelului C-256, dar avînd performanțe net superioare, viteza de prelucrare aproape dublă și capacitatea memoriei interne mărită de 2+4 ori
1976/1977/1980	Microcalculatoare românești de 8 biți (bazate pe microprocesoarele 8008 și 8080) MC-8, MC-80, M-18, M-118
1977/1978 cu modele moder- nizate în 1982/1985	INDEPENDENT și CORAL, primele minicalculatoare de 16 biți de concepție proprie, compatibile la nivel de utilizator cu seria PDP-11 larg răspîndită pe plan mondial (I-100, 102F, 106 și CORAL 4011, 4030, 4021, 4015)
1983/1986	Microcalculatoare de 8 biți bazate pe microprocesorul Z-80 cu memorie externă tip disc flexibil (M-118 B, Junior, CUB-Z) și cele cu minicasetă (AMIC, PRAE, HC85, TIM-S, COBRA); FELIX-PC, microcalculator profesional personal de 16 biți, compatibil cu IBM-PC și bazat pe microprocesorul INTEL 8086/8088
1987	FELIX-5000 calculator de medie capacitate de 32 biți, microprogramat, cu o capacitate a memoriei interne crescută de 4 ori față de modelul anterior FELIX C-1024, avînd o tehnologie pe scară largă, (unitatea centrală cu numai 7 plăci imprimare) și dimensiuni de minicalculator, păstrînd compatibilitatea cu seria FELIX
1987/1988	Junior XT, microcalculator profesional personal de 16 biți, compatibil cu IBM PC/XT
1988/1989	În cadrul familiilor CORAL (modelul 8730) și INDEPENDENT sînt elaborate tipurile de 32 de biți, primul fiind deja asimilat în producție

Notă: Evoluția echipamentelor periferice, terminalelor, sistemelor de operare și pachetelor de programe aplicative nu se prezintă explicit în acest tabel.

De asemenea, cadrele didactice ale Facultății de Electronică din IPB au adus contribuții însemnate la dezvoltarea unor domenii și la proiectarea-asmilarea unor produse noi cum sînt:

- minicalculatoare tip CORAL (asimilate în colaborare cu Întreprinderea de Calculatoare Electronice);
- display-uri grafice (produse la IEPER);
- mașini LISP (în curs de asimilare la IEPER);
- electronica funcțională;
- prelucrarea imaginilor;
- modelarea și simularea dispozitivelor semiconductoare și circuitelor VLSI.

Contribuții noi aduc în prezent, în colaborare cu institutele și întreprinderile de profil, cadre didactice de la Institutul Politehnic Traian Vuia din Timișoara (microcalculatoare, limbaje), Universitatea Brașov (microcalculatoare), Universitatea București (simulare), Universitatea Cluj-Napoca (limbaje), Institutul Politehnic din Iași (prelucrarea imaginilor), ș.a.m.d.

Astfel, în ultimii ani, în industria românească de tehnică de calcul și informatică, pe baza colaborării dintre colectivele de specialiști din cercetare, producție și învățămîntul superior, s-au intensificat cercetările, obținîndu-se rezultate noi în domenii de mare interes, cum sînt:

- calculatoare personale și personal-profesionale, domeniu la care se vor face referiri de ansamblu și de detaliu în această carte; de pildă, modelele aMIC și IC-85, respectiv modelele anterioare AC-8, MC-80, M-18, M-118, Junior și cele noi, CUB-Z, FELIX-PC și Junior XT la asimilarea cărora și-au adus contribuția cadrele didactice de la Catedra de Calculatoare din IPB și o serie de specialiști din industrie, apoi modelele PRAE elaborate de colectivul de la ITCI Cluj-Napoca, modelul COBRA proiectat la ITCI Brașov și TIM-S asimilat la FMECTC — Timișoara în colaborare cu IPT. Modelul aMIC este deja cunoscut cititorilor prin cartea în două volume ce l-a fost dedicată, iar modelul HC-85 se prezintă pe larg în această carte;
- minicalculatoare stabile, cu productivitate sporită, modelele INDEPENDENT și CORAL de 16 și 32 b'i;
- calculatoare de capacitate medie FELIX C-5000 compatibile cu seria FELIX anterioară.

Menționăm de asemenea și alte direcții importante ale cercetărilor din țara noastră în acest domeniu, cum sînt:

- conceperea sistemului informatic național, elaborarea de metode, tehnici, modele și elemente tipizate destinate realizării componentelor acestuia, precum și a rețelelor largi și locale de calculatoare, mini și microcalculatoare;
- elaborarea sistemelor de operare noi pentru microcalculatoare (sistem U), minicalculatoare (MIX-PLUS) și calculatoare medii (HELIOS);
- realizarea unor noi compilatoare, programe de sistem și aplicative;
- realizarea sistemelor de gestiune a bazelor de date (produsele SOCRATE-MINI — de către Centrul teritorial de calcul Constanța, respectiv ARGUS, LEDA, RECOL, STAR, FOCUS și TRANS elaborate la ITCI);
- sisteme de programe pentru proiectarea asistată de calculator și sisteme grafice la cheie pentru aplicații în electronică (proiectarea circuitelor larg integrate și a plăcilor imprimate), mecanică, arhitectură, construcții, topografie, cartografie;
- sisteme informatice pentru conducerea producției, proceselor tehnologice și gestiune în unitățile economice, precum și a sistemelor flexibile de fabricație;
- inteligență artificială, sisteme expert și baze de cunoștințe;
- ingineria programării;
- echipamente periferice de tip bandă magnetică cu transfer continuu, ori de tip cititor de cartelă magnetică cu cap magnetic integrat;

În noile condiții revoluționare se acordă o atenție deosebită organizării și consolidării în țara noastră a industriei de calculatoare și programe.

I. CALCULATOARE ÎN ȚARĂ ȘI ÎN LUME

1.4. Aplicații ale calculatoarelor în societatea modernă

Cu toate că s-au scurs puțini ani, chiar relativ la istoria civilizației tehnice moderne a omenirii, calculatoarele electronice au pătruns în activitatea și chiar în viața personală pe o scară atât de largă încît ele reprezintă astăzi o resursă indispensabilă a societății. Evoluția lor, atât constructiv cît și ca utilizare, se află în strînsă interdependență, atât cu evoluția tehnicilor și tehnologiilor din toate domeniile de activitate socială, cît și cu evoluția gîndirii umane privind modul de abordare a dezvoltării diverselor ramuri economice. Calculatoarele au oferit gîndirii umane mijloace noi de investigare, posibilități noi de sistematizare a datelor care erau de multe ori ignorate, de analiză a interdependenței lor, au determinat chiar o organizare superioară, prin simplitate și limpezime, a modului de abordare a problemelor.

În prima etapă a evoluției calculatoarelor (pînă în 1960) cererile de astfel de produse, cu excepția unor cereri izolate, unde se impuneau din necesități obiective, au apărut din motive de modă tehnică, de teama rămîinerii în urmă, tradiția fiind spre sisteme tot mai performante ca putere de calcul. Reacția celor care au investit sume mari în echipamente și specialiști de înaltă calificare, capabili să le utilizeze, a fost promptă: ceva trebuia schimbat atât în privința mașinilor, la care se dovedea necesar să fie dimensionate nu numai pentru superprobleme, ci și pentru aplicații mai limitate, de zi cu zi, cît și în privința modului de interacționare cu operatorul, care era de dorit să fie chiar utilizatorul procesului de calcul. Efectul acestei reacții îl regăsim în jurul nostru *astăzi cînd îți înțelnești calculatoare în grădinițe, școli, la domiciliu, în cercetare-proiectare, în conducerea de utilaje, instalații și procese industriale, în investigații medicale, în biblioteci, în magazine, în bănci, în aeroporturi, la întreceri sportive, în prognozarea meteorologică, în teledetecție la prognozarea recoltelor de cereale, în comunicații, la controlul zborului navetelor spațiale*. Ele sînt utilizate direct de oameni a căror profesie aparține domeniului de aplicație și nu de specialiști în domeniul calculatoarelor.

Unul dintre cele mai populare exemple în această direcție îl reprezintă jocurile, cum este jocul de ping-pong „jucat“ acasă pe televizorul existent. Videojocurile reprezintă prima utilizare de masă a ceea ce se definește sub denumirea de „computer graphics“, adică crearea și manipularea de imagini cu ajutorul calculatorului. Un proverb datînd din antichitatea chineză sintetizează importanța acestui mod de abordare a interacțiunii om-calculator afirmînd că „o imagine valorează cît o mie de cuvinte“.

Domeniile de utilizare a calculatoarelor, așa cum rezultă de mai sus, sînt extrem de numeroase și în continuă expansiune. În continuare se vor prezenta cîteva direcții de aplicare.

În cercetarea științifică modernă, modelarea matematică a fenomenelor și proceselor fizice reprezintă un instrument extrem de puternic. Oamenii de știință elaborează modelul matematic al fenomenului în cauză. Pentru ecuațiile matematice, care stau la baza modelului, se stabilesc metode numerice de rezolvare. Metodele de rezolvare sînt descrise în limbaje de programare, pentru a obține programele care, în final, se execută pe un calculator dat.

Programele permit obținerea soluțiilor într-o gamă largă de variație a datelor de intrare, a parametrilor proceselor fizice studiate. Se pot analiza situații

limită: aducerea sistemelor fizice în starea de avarie, găsirea limitelor de stabilitate a proceselor, studierea unor mărimi care nu pot fi măsurate direct, etc. În acest sens se poate aminti folosirea calculatorului în modelarea fenomenelor întâlnite în instalațiile termonucleare, în studiul dinamicii aparatelor de zbor supersonice, în studiul sistemelor complexe de transport al energiei electrice, în astrofizică. Calculatorul permite astfel înlocuirea experimentelor în instalațiile fizice costisitoare, asigurând reproducerea, prin modelare, a numeroase variante, fără cheltuieli suplimentare.

Un domeniu care a beneficiat din plin de utilizarea calculatorului este cel al *prognozei vremii*, pe baza datelor culese cu ajutorul stațiilor meteorologice, al sateliților, etc. Datele sînt prelucrate cu calculatoare de mare capacitate, programate să rezolve ecuații cu derivate parțiale, cu numeroase și variate condiții la limită și inițiale. Rezultatele sînt prezentate sub formă de hărți sinoptice, cuprinzînd harta teritoriului interesat, curbele izobare, izoterme, etc.

Calculatoarele se folosesc pentru *colectarea și prelucrarea datelor* în cadrul diverselor experimente. Astfel, prevăzute cu traductoare (echipamente care transformă în semnale electrice — tensiuni sau curenți — diferite mărimi fizice neelectrice — lungimi, viteze, accelerații, debite, presiuni, temperaturi etc.), calculatoarele pot stoca și prelucra, în timp real, valorile diferiților parametri ce caracterizează un proces sau sistem fizic. Rezultatele sînt prezentate în forme adecvate, pe ecranul unui dispozitiv de afișare (display) sau cu ajutorul unui înregistrator (plotter), ceea ce facilitează interpretarea lor de către experimenterii. Un exemplu îl constituie analizoarele de vibrații pentru motoarele cu ardere internă sau pentru motoarele reactive.

Pe ecran pot fi afișate sub formă grafică amplitudinile semnalelor la frecvențele de rezonanță cu valorile numerice corespunzătoare.

Utilizarea calculatoarelor în medicină a căpătat o largă răspîndire. Ele sînt folosite pentru monitorizarea pacienților, în sălile de operație și de terapie intensivă, în instalațiile pentru efectuarea automată a analizelor, în investigațiile termografice, ecografice și tomografice (prelucrarea de imagini). Imaginile obținute cu ajutorul termografului și al tomografului sînt prelucrate în vederea accentuării contururilor, eliminării unor semnale parazite, calculului suprafețelor cu o anumită nuanță de gri și colorării artificiale (atribuirea unei culori pentru o nuanță dată de gri), etc.

Un domeniu care utilizează din ce în ce mai mult calculatorul este cel al proiectării — *proiectare asistată de calculator* (PAC). Sistemele de Proiectare Automată (SPA) au căpătat o largă răspîndire în construcția de mașini, arhitectură, industria lemnului (mobilier), industria ușoară (modele de țesături, tipare, tricotate), etc.

Proiectarea automată a matricelor necesită programe complexe care operează în trei dimensiuni (3 D), permițînd afișarea pe ecran a piesei dorite, în diferite proiectii. Calculatorul furnizează automat banda perforată pentru conducerea mașinii de așchiere sau electroeroziune, cu comandă numerică. Astfel, crește productivitatea muncii și se reduc termenele de execuție.

Proiectarea asistată de calculator se folosește în mod frecvent în *studiul comportării statice și dinamice a diverselor sisteme, echipamente, mașini, etc.* Astfel, pot fi puse în evidență componentele mai solicitate și se pot lua măsuri pentru creșterea fiabilității acestora.

Proiectarea asistată de calculator joacă un rol important în *realizarea de noi substanțe chimice, medicamente, etc.* Pe baza modelelor teoretice, elaborate pentru interacțiunile la nivel molecular, ale diversilor compuși chimici, se pot proiecta noi compuși, vizualizându-se pe ecran structurile moleculare ale acestora, fără a mai efectua experimente de laborator costisitoare și laborioase.

Sistemele de rezervare a locurilor pentru pasageri, în transporturile aeriene sau pe cale ferată, reprezintă unele din importantele aplicații ale calculatoarelor în sfera serviciilor. Agențiile de voiaj sînt prevăzute cu terminale, conectate la distanță, la un calculator central. Acesta din urmă dispune de capacități importante de stocare și regăsire a informațiilor referitoare la cursele de avioane, orarul trenurilor, listele cu pasagerii care au rezervat locurile, numărul locurilor disponibile, etc. Terminalul aflat la agenția de voiaj poate elibera, de asemenea, automat, biletul solicitat la avion sau la tren.

Sistemele de stocare și regăsire a informațiilor și-au găsit importante utilizări în *marile biblioteci și instituții de documentare*. Cărțile, revistele, articolele de specialitate sînt introduse în memoria calculatorului sub forma unor înregistrări care conțin o serie de informații esențiale: numele autorilor, titlul, editura, anul apariției, numărul de figuri, numărul referințelor bibliografice, cuvinte cheie, un scurt rezumat, etc. Pe baza unor cuvinte cheie date, calculatorul selectează, în mod automat, din baza de date, lucrările corespunzătoare, ușurînd astfel, în mod substanțial, munca de cercetare bibliografică.

În ultimii ani o largă răspîndire a găsit *prelucrarea textelor cu calculatorul*. Pe baza unui program special, numit procesor de texte, utilizatorul operează cu calculatorul în maniera în care ar lucra cu o mașină de scris inteligentă. Textul introdus este afișat pe ecran, existînd comenzi speciale pentru inserarea/inlăturarea de cuvinte, fraze sau paragrafe. La sfîrșitul rîndului, automat, se face despărțirea în silabe, respectiv alinierea cuvintelor. Documentele astfel editate sînt stocate pe suport magnetic (disc flexibil, casetă magnetică). Ele pot fi ulterior modificate după cerințe sau extrase la imprimantă, ori de cîte ori este nevoie.

Preocupările constructorilor de calculatoare continuă în direcția simplificării sau naturalizării interacțiunii operator-calculator. În acest sens realizările obținute pe linia sintezei și recunoașterii vorbirii (pînă la 20.000 și respectiv 1.000 de cuvinte) sînt încurajatoare, chiar dacă unii sînt încă sceptici datorită restricțiilor privind ritmul vorbirii. Există anunțuri recente ale unor firme americane privind comercializarea de „secretare computerizate“ care pot redacta corespondența după dictare.

În ultimii ani s-au înregistrat importante progrese pe linia realizării unor sisteme expert, în diverse domenii de activitate: medicină, biologie, geologie, proiectare, întreținere de instalații complexe, etc. Bazate pe calculatoare evolute înzestrate cu programe adecvate înglobînd principiile inteligenței artificiale, aceste sisteme sînt capabile să manipuleze cunoștințe, folosind reguli de inferență, care permit modelarea raționamentului unor experți umani, în diferite domenii, cu consecințe economice și sociale foarte importante. În cazul *roboților*, problemele ce vizează aspectele de inteligență artificială se referă la faptul că, pentru aceste echipamente nu este suficient să fie capabile să manipuleze obiecte; robotul trebuie să fie capabil să primească și să interpreteze informații legate de mediul înconjurător și să-și adapteze acțiunile sale ținînd cont de aceste informații.

Cu sistemele realizate în sectoarele de proiectare, cu elementele de programare și conducere a fabricației, cu realizările din domeniul roboților, cu mașini prelucrătoare dotate cu calculatoare sau minimum cu comenzi numerice, a apărut ca posibilă o automatizare flexibilă, adaptabilă în timp scurt varietății pieselor de fabricat, în scopul obținerii unei productivități cit mai ridicată. Este cunoscut faptul că flexibilitatea și productivitatea sînt în general două deziderate contradictorii și de aceea a apărut ca necesară găsirea unei metode de îmbinare optimă a acestora, fără a accentua prea mult pe una în detrimentul celeilalte.

Calculatoarele pot fi interconectate în cadrul unor *rețele locale sau globale* (la nivel de țară sau continent).

În *rețelele locale*, calculatoarele permit, de pildă, implementarea conducerii operative a unor întreprinderi, instituții dispuse pe o arie geografică restrînsă.

Rețelele globale conectează, de exemplu, centrele de calcul teritoriale sau calculatoarele plasate la distanță, în cadrul întreprinderilor diverselor ministere economice.

Reducerea continuă a costurilor componentelor electronice, apariția microprocesorului, perfecționarea în general a tehnologiilor electronice, au facilitat *apariția calculatoarelor personale*. Acestea, în funcție de tipul microprocesorului, de capacitatea memoriei, de echipamentele periferice folosite pot fi plasate în clasa calculatoarelor personale de uz general, sau în clasa calculatoarelor personale — profesionale (microcalculatoare evoluat tehnologic).

Această dezvoltare tehnologică a calculatoarelor personale, creșterea performanțelor lor pe baza progreselor deosebite ale microelectronicii, le-au transformat într-o resursă importantă pentru rezolvarea unor probleme și aplicații, care pînă în acești ultimi 3—5 ani erau adresate minicalculatoarelor și chiar calculatoarelor medii — mari.

Calculatoarele personale sînt folosite de largi categorii de utilizatori: ingineri, medici, economiști, profesori, tehnicieni, studenți, arhitecți, agronomi, muzicieni, elevi, etc.

Calculatoarele informatizează activitățile de rutină, care necesită un mare volum de muncă, permițînd utilizatorului concentrarea asupra aspectelor creatoare ale muncii sale. Pentru a obține beneficiile maxime din utilizarea calculatoarelor este necesar ca programarea operării lor să fie efectuată pe baza unor algoritmi corecți de calcul, fără ambiguități, inexactități, etc. În acest sens se impune din partea utilizatorului o profundă înțelegere a problemei, care este programată spre a fi rezolvată cu calculatorul.

Am început acest subcapitol afirmînd că, în societatea modernă calculatoarele reprezintă o resursă cheie a dezvoltării societății; cu fiecare lună, trimestru sau an, ele devin o forță de producție esențială. Toate meseriile sînt practic influențate de impactul calculatorului și vor fi în continuare, poate într-un ritm tot mai ridicat. Pregătirea tuturor membrilor activi ai societății pentru utilizarea calculatorului reprezintă o condiție strategică a etapei actuale, această carte încercînd să aducă o contribuție la punerea în practică a acestei strategii.

2.1. Ce este un calculator personal?

Cartea aceasta, prima carte scrisă la noi pentru învățarea utilizării unui calculator personal compatibil cu o familie larg răspândită pe plan mondial, se referă la modelul românesc HC-85, asimilat în 1985 în fabricația de serie la Întreprinderea de calculatoare electronice din București.

Calculatorul HC-85 se înscrie în seria echipamentelor de calcul românești competitive, cu performanțe apreciabile, îndeosebi pentru instruirea asistată de calculator. HC-85 este fiabil, economic și compatibil cu un alt model, larg răspândit pe plan mondial.

Caracteristicile de mai sus pot fi în general exprimate sub forma unui criteriu sintetic de apreciere, dat de raportul dintre performanță și preț. Raportul este îmbunătățit continuu pe plan mondial, prin creșterea performanțelor (viteza de prelucrare, capacitatea de memorare, facilități pentru un dialog „prietenos“ om-mașină, programe eficiente, fiabilitate) și reducerea prețului.

Calculatorul personal este realizat în jurul unui circuit integrat complex, numit microprocesor și care este ajutat în „activitatea“ sa de alte circuite, unele mai simple, altele complexe, asigurând funcțiuni specializate. În ultimii ani au apărut microprocesoare noi, care prelucrează de două ori și chiar de patru ori mai multe unități de informație, în unități de timp de câteva ori mai mici. Astfel, o dată cu utilizarea microprocesoarelor de 16 și de 32 de biți, a crescut viteza și puterea de prelucrare a datelor.

Totuși, *calculatorul personal HC-85 selectat pentru instruirea elevilor, are un microprocesor clasic, tip Z80A, de 8 biți, la fel ca alte calculatoare din grupa sa. În țara noastră s-au proiectat și se fabrică și calculatoare cu microprocesoare de 16 biți, evident la un preț de cost mult mai ridicat.*

Nu puțini dintre tinerii elevi participanți la cercurile de informatică au o curiozitate științifică demnă de viitori specialiști și își pun o serie de întrebări.

Unii se interesează mai mult de utilizarea calculatorului personal, privit ca un instrument nou, așa cum în alte timpuri au fost abacul sau rigla de calcul. Dorința lor de a deveni utilizatori evoluți este în concordanță cu evoluția folosirii tot mai extinse a calculatoarelor.

Alții se interesează mai ales de „arhitectura“ și tehnologia microelectronică pe care se bazează calculatoarele personale și doresc să afle cum funcționează calculatorul și chiar microprocesorul, „creierul și inima“ calculatorului personal. Microprocesorul este un circuit integrat cu o zonă activă de câțiva mm² în care este „imprimată“ o schemă echivalentă cu 10 mil sau uneori chiar 300 de mil de tranzistoare, schemă în care cele mai fine trasee au o lățime de 3 microni, 2 microni și mai recent sub 1 micron. În viitor vor coborî sub o jumătate de micron. Și apoi? Iată, notăm și alte întrebări.

Care sînt tendințele pe plan mondial în acest domeniu? Dar la noi în țară? Cum se clasifică calculatoarele personale? Unde se încadrează HC-85? Care este durata de viață pentru utilizarea calculatoarelor cu microprocesoare de 8 biți? Care tehnologii ale microelectronicii sînt mai răspîndite? Cum evoluează microprocesoarele? Care limbaj de programare trebuie învățat mai întîii? Dar specialiștii de azi și de mîine, ce limbaje au sau vor avea la dispoziție? Cum se dezvoltă instruirea asistată de calculator pe plan mondial?

Calculatorul cel mai des întîlnit în următorii cîțiva ani, pe masa de lucru a elevilor pionieri, dar și a elevilor UTC-Iști, și într-o anumită măsură a studenților, a inginerilor, a profesorilor, va fi calculatorul HC-85. Vor exista și alte modele total compatibile cu HC-85, destinate aceluiași aplicații.

Și totuși... peste 10—15 ani, elevii aflați azi pe băncile școlii vor deveni muncitori cu înaltă calificare, medici, ingineri, chimiști, fizicieni, matematicieni, profesori. Unii dintre ei vor fi proiectanți de calculatoare, informaticieni de înaltă competență.

Îată de ce, întîind gîndurile multor tineri cititori, includem cîteva pagini despre prezentul și viitorul calculatoarelor. Vor fi introduse unele noțiuni noi care se vor explica gradat. Se vor da și trimiteri la o bibliografie selectată și comentată, care cuprinde atît lucrări de popularizare, cît și unele lucrări de specialitate existente în bibliotecă.

Ce este deci, un *calculator personal*? Cum să definim aceste *microsisteme de echipamente și programe*?

Vom spune simplu, fără a formula o definiție extinsă, că un calculator personal este interactiv, portabil, are o structură cu unul sau mai multe microprocesoare, o tastatură și un afișaj de tip TV ori monitor, monocrom sau color, una sau mai multe memorii externe, unul sau mai multe limbaje de programare de nivel înalt, poate cîteva opțiuni (dispozitive și programe), și toate acestea la un preț din ce în ce mai redus.

Dintre caracteristicile de bază, următoarele prezintă o importanță deosebită și anume:

- utilizarea individuală interactivă, toate resursele microsistemului fiind la dispoziția operatorului — programator (exceptînd memoria permanentă, protejată la scriere);
 - interfața om-mașină „prietenosă”, facilitînd dezvoltarea dialogului și ghidarea continuă a operatorului;
 - facilitățile modelelor profesionale de interconectare în rețele locale sau de cuplare la calculatoare mai mari, permițînd accesul la baze de date și cooperarea calculatoarelor la rezolvarea unor probleme de mai mare complexitate.
- În continuare, vom considera că un calculator personal (sau, chiar un microcalculator) este alcătuit în principal din mai multe elemente, dispozitive și programe, selectate dintr-o gamă largă, care cuprinde:
- modulul de bază cu structura logică, cu unul sau mai multe microprocesoare și alte circuite menționate mai jos, inclusiv pentru controlul imaginii;
 - memoria în care se scrie și se citește orice zonă, pentru programele utilizatorului, respectiv pentru ecran;
 - memoria permanentă care se poate doar citi, pentru monitorul de comenzi, interpretor și programe utilitare;
 - circuitele de interfașă cu echipamentele periferice;
 - sursa de alimentare încorporată în calculator sau distinctă;
 - dispozitive și periferice de intrare/ieșire, încorporate în modulul de bază sau distincte, cum sînt: tastatura, dispozitivul de afișaj monocrom/color tip TV sau monitor, dispozitivul de poziționare cu bilă („mouse”) tip penel sau cu manetă („joystick”), ambele cu butoane pentru poziționarea cursorului pe ecran, minilmpri-manta și trasatorul de curbe (plotter de masă);
 - unități de memorare externă, încorporate în modulul de bază sau distincte, pentru calculatoare personale uzuale (casetofon) și pentru tipurile profesionale (unități de discuri magnetice flexibile și/sau unități de disc fix sau rigid, tip Winchester, cu capacități de memorare de 0,25—1 Moet., respectiv de 10—40 Moet.);

- monitor — dezasamblor, editor, asamblor, interpretor pentru BASIC și câteva compilatoare pentru limbaje evaluate (în cazul calculatoarelor personale portabile, de familie, etc.), sau
- interpretor de BASIC extins, eventual

sistem de operare, translație pentru programe în limbaj de asamblare și în BASIC, compilatoare pentru limbaje evaluate, respectiv medii de dezvoltare pentru programe noi (în cazul calculatoarelor personale-profesionale);

După descrierea gamei extinse din care se aleg configurațiile tipice ale calculatoarelor personale și personal-profesionale, să trecem în revistă câteva noțiuni utile programatorilor începători, dar mai ales celor care au depășit fazele inițiale, noțiuni care se referă la interpretoarele BASIC, compilatoare, monitor, sistem de operare, modul interactiv de lucru și instruirea asistată de calculator.

Interpretorul BASIC analizează și execută direct instrucțiune cu instrucțiune programul sursă (scris în BASIC de programator). Utilizatorul obține rezultatele execuției programului și nu un program obiect intermediar în cod mașină, adică în instrucțiunile microprocesorului. Interpretorul asigură un caracter interactiv, dar este în general mai puțin eficient decât compilatorul.

Compilatorul transformă programul sursă (scris într-un limbaj de nivel înalt, de pildă FORTRAN) într-un program obiect echivalent (scris în cod mașină). Aceasta înseamnă transformarea unui șir de caractere (programul sursă) într-un șir de biți (programul în cod mașină) și implică o serie de operații:

- analiza lexicală pentru a evidenția constante, variabile, operatori, cuvinte-cheie.

- analiza sintactică (corecțitudine gramaticală) și semantică (sensul cuvintelor înlănțuite), pentru a verifica regulile stabilite la definirea sintactică a limbajului;

- generarea intermediară a codului în limbaj de asamblare;

- optimizarea codului pentru reducerea spațiului de memorare și a timpului de execuție;

- generarea codului obiect prin operația de asamblare (se traduce limbajul intermediar în cod mașină);

- în toate fazele compilării, enumerate mai sus, au loc două operații privind alcătuirea și gestiunea tabelor (tabele de simboluri, de constante, de cicluri), respectiv analiza erorilor. Pentru unele compilatoare la care se generează direct codul mașină, operația de asamblare nu mai este necesară.

Monitorul reprezintă în cazul calculatoarelor personale, o colecție de comenzi și rutine care se pot apela în două moduri: de la tastatură și prin programele utilizatorului. Monitoarele din această categorie ocupă 1—8 Koct. în memoria permanentă. Monitoarele calculatoarelor personale românești introduse în fabricație, includ 10—15 comenzi, dintre care menționăm, de pildă:

- afișarea pe ecran a conținutului unei zone de memorie;

- afișarea registrelor interne;

- lansarea în execuție a unui program obiect aflat în memorie.

Un asamblu de componente software Monitor-Asamblor-Editor de texte rezident în memoria permanentă constituie un nucleu de sistem de operare.

Sistemul de operare este, în general, un asamblu de programe care realizează gestiunea resurselor unui sistem de calcul, rezolvă conflictele care apar în alocarea resurselor sau între utilizatori (la micro sistemele cu acces multiplu, de la mai multe terminale) și asigură o utilizare eficientă a echipamentelor și programelor.

Calculatoarele personal-profesionale produse în țara noastră, pot fi împărțite în două grupe (cărora le corespund sisteme de operare specifice):

- compatibile cu sistemele de operare CP/M pentru microcalculatoare de 8 biți (mult mai răspândite decât sistemul de operare anterior SFDX).
- compatibile cu sistemele de operare MS-DOS și PC-DOS pentru microcalculatoare de 16 biți (asemănătoare principial cu CP/M).

În aceste două cazuri, încărcarea sistemului de operare în memoria calculatorului se realizează prin transfer de pe suportul magnetic extern.

Modul interactiv, conversațional, de lucru al (micro) calculatorului personal permite utilizatorului să fie în dialog, în contact permanent cu problema dată, astfel încât programele pot fi dezvoltate și corectate mult mai rapid. Un domeniu deosebit de atrăgător pentru instruirea asistată de calculator, îl constituie animația și grafica color interactivă (însoțite de efecte sonore) realizate cu ajutorul unor dispozitive de intrare și a programelor adecvate pentru poziționarea și selecția unui „punct” pe ecran (tip „mouse” sau penel, „joy-stick”, creion optic, tastatură funcțională, tabletă grafică, ecran sensibil la atingere, etc.), respectiv cu dispozitive grafice de ieșire (display, miniimprimantă, trasa-tor de curbe).

Limbajele principale utilizate în domeniul calculatoarelor personale sînt: BASIC, LOGO, FORTH.

Pentru cele profesionale, există și alte limbaje evaluate: C, TURBO C, FORTRAN 77, TURBO-PASCAL, MODULA, ADA, LISP și PROLOG.

2.2. Microelectronica (microprocesoare și alte circuite integrate) pe plan mondial

Încă de la mijlocul deceniului trecut, experții atrăgeau atenția asupra faptului că importanța microelectronicii pentru o țară industrializată egalează pe aceea a ingineriei nucleare!

Într-adevăr, întreaga dezvoltare a tehnicii de calcul începînd cu anii '70 se bazează pe *evoluția rapidă a microelectronicii*, pe creșterea spectaculoasă a mărimii seriilor de fabricație, a complexității, vitezei și fiabilității circuitelor integrate, în condițiile reducerii dramatice a consumului de putere și a prețurilor. Tehnica de calcul, microelectronica, informatica, automatizarea, reprezintă pilgurile cheie pentru creșterea productivității muncii și oferă soluții noi pentru dezvoltarea economică intensivă, în cadrul unor resurse materiale și financiare limitate la anumite niveluri date.

Se poate afirma că, deși valoarea microprocesoarelor și a celorlalte circuite integrate din componența calculatorului este mică, în comparație cu valoarea perifericelor și a programelor de bază și aplicative, totuși rolul acestor circuite complexe și miniaturizate este vital. *Ponderea valorică a circuitelor integrate din totalul echipamentelor de tehnică de calcul reprezintă numai 14%, dar ponderea valorică a producției de circuite integrate din producția de componente semiconductoare este pe plan mondial de circa 85%. De altfel, toate țările dezvoltate industrial investesc an de an sute de milioane de dolari în dezvoltarea microelectronicii, asigurînd un ritm alert de înnoire a tehnologiilor și producții de mare serie, cu costuri în reducere.*

Consumul de circuite integrate are în 1989, conform unor estimări recente, următoarea structură calculată în ponderi valorice (considerînd consumul egal cu producția valorică plus import minus export), pentru 6 țări capitaliste dezvoltate industrial (SUA, Japonia, RFG, Marea Britanie, Franța și Italia);

I. CALCULATOARE NUMERICE

Circuite integrate, total 100%
din care:

— Memorii	31%	— Circuite fabricate la comandă	16%
— Microprocesoare, microcalculatoare într-o singură capsulă și circuite specializate	23%	— Circuite analogice	19%
		— Familii de circuite logice standard	11%

Principalele familii de circuite integrate logice sînt următoarele, date în ordinea descrescătoare a ponderilor valorice, din consumul total de circuite din această grupă, care totalizează doar 11%:

- **TTL Schottky (logica tranzistor-tranzistor)**, cu complexitate medie și viteză mare de lucru (timp redus de propagare a semnalului logic prin circuit, numit și timp de propagare sau comutare), incluzînd și tipurile de circuite TTL Schottky Low Power, deci versiunile cu consum redus de putere;
- **CMOS (logica complementară în tehnologia metal-oxid-semiconductor)**, de mare complexitate și densitate per unitate de arie, cu viteza medie spre mare și consum de putere extrem de redus;
- **ECL (logica cu cuplaj al tranzistoarelor prin emitor)**, de complexitate medie, dar de foarte mare viteză;
- **TTL de mare viteză**, cu complexitate mică spre medie.

Dintre acestea, primele două sînt cel mai mult folosite în domeniul calculatoarelor personale și microcalculatoarelor.

Grupa circuitelor integrate de memorare (31%) include:

- **memoriile cu acces aleator (RAM)**, fiind posibilă scrierea sau citirea datelor în orice zonă a memoriei, cu tipurile dinamice (datele sînt recirculate continuu) și statice. Tipurile de memorii dinamice și cele mai multe din cele statice sînt realizate în diferite versiuni ale tehnologiei MOS, iar memoriile statice au și variante bipolare;
- **memoriile de tip numai-citește (ROM)**, cu mască, unele tipuri fiind programabile (PROM), ori cu posibilitatea de a fi șterse electric (EEPROM), sau cu ra-
- ze ultraviolete (EPROM). Se mai numesc memorii fixe sau permanente, avînd un conținut fixat de producător, și peste care utilizatorul nu poate înscrie alte date sau programe. Sînt realizate în tehnologiile MOS, CMOS și bipolară;
- **memorii neconvenționale realizate cu tehnologii noi**, cum sînt dispozitivele cu cuplaj prin sarcină (CCD) și cele cu bule magnetice (magnetic bubble), ambele avînd însă pînă în prezent o pondere foarte mică.

Memoriile RAM dinamice (DRAM) și cele fixe de tip EPROM sînt larg utilizate în calculatoarele personale și micro, ca memorii interne pentru date și programe. Dintre memoriile RAM dinamice cu capacități de 64, 256, 1024 și 4096 de kilobiți/circuit integrat, cele mai utilizate sînt cele de 256 kbiți și 1Mbit, care au împreună o pondere valorică de 94% în 1989, din totalul memoriilor RAM destinate utilizării în industria de tehnică de calcul din SUA (cu mențiunea că ritmul de creștere al producțiilor de circuite de 1Mbit este mult mai mare, acest tip tinzînd să depășească 75% în 1989). Prețurile pentru memoriile de 64 și 256 kbiți, chiar pentru cantități mici, au scăzut enorm și au ajuns la un nivel extrem de redus, sub un dolar și respectiv cîțiva dolari, comparativ cu circuitul integrat de 1Mbit care costă cu aproape două ordine de mărime mai mult.

Cu toate acestea circuitul de 1 Mbit cu timp de acces redus la 100 ns este preferat atît din punctul de vedere al performanței (mai apropiate de aceea a noilor microprocesoare) cit și din punct de vedere economic (deoarece elimină memoriile RAM statice rapide, folosite pentru a compensa decalajul dintre vitezele de lucru tot mai mari ale microprocesoarelor și viteza relativ redusă a memoriilor de 64/256 kbiți).

Cea de a doua grupă de circuite integrate (conform tabelului 2.1) pe care o prezentăm în continuare, este cea mai importantă pentru calculatoarele personale și microcalculatoare. Această grupă include microprocesoare, microcalculatoare pe o pastilă de siliciu (numite one-chip), circuite integrate specializate pentru comanda memoriei interne, a echipamentelor periferice, pentru prelucrarea semnalelor, sinteza vocii, etc.

Dacă vom considera o subgrupă distinctă compusă din microprocesoare (mP de 8, 16 și 32 de biți) și microcalculatoare one-chip (mC de 4, 8 și 16 biți) luate împreună, atunci ponderile în consumul SUA, în 1989, sînt estimate astfel:

Tabelul 2.2

**Ponderea valorică
în 1989, în S.U.A.**

**Microprocesoare și
microcalculatoare one-chip, total 100%**

din care:

— mP și mC, de 8 biți	45%	— mP de 32 biți	15%
— mP și mC, de 16 biți	33%	— mC de 4 biți	7%

Din acest tabel rezultă că producția valorică și deci consumul valoric de aceste componente profesionale este maxim în plaja performanțelor medii (8 biți și 16 biți), ceea ce este și mai evident pentru producția fizică și consumul fizic exprimate prin numărul de circuite.

S-a exemplificat prin structura consumului SUA de microprocesoare și microcalculatoare one-chip, deoarece acest consum valoric reprezintă aproape 60% din suma consumurilor similare din alte 5 țări dezvoltate industrial (Japonia, RFG, Anglia, Franța și Italia), iar pe de altă parte, statisticile publicate în literatura de specialitate sînt mai detaliate pentru SUA.

Dacă se analizează livrările fizice (exprimate în numărul microprocesoarelor), atunci se constată că peste 85% din numărul microprocesoarelor livrate au fost tipurile de 8 biți.

În ce constă de fapt secretul acestei mari durate de viață pentru microprocesoarele de 8 biți, apărute în urmă cu 15—16 ani și despre care se presupune că nu au atins încă virful livrărilor proprii și vor depăși 25 de ani de producție? Cel puțin doi factori au un rol determinant:

- creșterea continuă a performanțelor microprocesorului de 8 biți de bază, prin încorporarea (integrarea) unor funcțiuni noi de control al memoriei și al perifericelor, și prin mărirea vitezei interne de lucru;
- cuplarea arhitecturilor de 8 biți cu sistemul de operare CP/M, larg răspîndit și mai simplu decît sistemele de operare Unix sau MS-DOS și dezvoltarea unui mare număr de programe de aplicații compatibile CP/M.

În figurile 2.1 și 2.2 sînt date evoluția complexității microprocesoarelor, mărirea estimativă a seriilor de producție pentru circuitele clasice și de virf din familiile microprocesoarelor.



Fig. 2.1. Evoluția complexității microprocesoarelor



Fig. 2.2. Producția mondială de microprocesoare de 8 și 32 biți (cantități însumate în perioada 1974—1988)

Iată de ce, colectivul de autori consideră că HC-85, calculatorul personal românesc, bazat pe cunoscutul microprocesor de 8 biți Z80A (avînd în vedere fiabilitatea sa, extensiile de care dispune, inclusiv pentru lucrul cu unitatea de discuri flexibile, cît și prețul de cost redus), ca și celelalte modele românești compatibile cu HC-85, reprezintă o bună alegere pentru utilizare largă în școli, la cursurile și la cercurile de informatică pentru elevi, pionieri și UTC-iști.

2.3. Calculatoare personale, personal-profesionale și microcalculatoare în lume

Industria relativ recentă a calculatoarelor personale pe plan mondial are deja o istorie de peste 10 ani, cu o etapă nouă începută în anul 1981, prin pătrunderea spectaculoasă a celei mai mari firme de calculatoare din lume, IBM-SUA, în domeniul calculatoarelor personale, IBM reușind în numai un an să treacă de pe locul doi pe primul loc, schimbând poziția cu firma APPLE-SUA.

Fără a prezenta o imagine completă, menționăm câteva dintre firmele producătoare de calculatoare personale larg răspândite pe plan mondial și bine cunoscute de asemenea și la noi în țară, precizând câteva modele în ordinea relativă a introducerii în fabricație:

- Sinclair — Marea Britanie: ZX80, ZX81, ZX Spectrum, ZX Spectrum Plus și QL;
- Commodore — SUA: PET Commodore, VIC20, Commodore 64, 16, 116, Plus/4, 128 și PC;
- Apple — SUA: Apple I, II și III, Lisa, Macintosh, Mac Plus, Mac SE și Mac II.
- IBM — SUA: IBM PC, PC/XT, PC Jr., PC/AT, PC/RT și seria PS/2;
- COMPAQ — SUA: DESKPRO 386.

Pentru modelele mai cunoscute dăm în tabelul 2.3 câteva caracteristici tehnice. Pentru aceste modele, prețurile aflate în continuă reducere acoperă o plajă largă, desigur și în funcție de configurație, de la sub 100 \$ pentru Spectrum, la 1000—5000 \$ pentru IBM PC/AT și 10000 \$ pentru ultimul model PS/2.

În clasificarea adoptată de revista Electronics — SUA, calculatoarele personale sînt plasate în plaja de prețuri pînă la 5000 \$, iar microcalculatoarele (calculatoare personal-profesionale) între 5 și 20 mii \$ (incluzînd microsistemele multiutilizator și stațiile de lucru), dar uneori în diverse lucrări se folosește fie denumirea de calculatoare personale, fie de microcalculatoare, în extensie pentru ambele grupe.

O scurtă prezentare privind începutul istoriei microcalculatoarelor și calculatoarelor personale, inclusiv utilizarea lor, este dată în suplimentul revistei „Știință și tehnică” din septembrie 1986, intitulat „Calea-

lătorul, nămie mai simplu”, iar o serie de articole tehnice, detaliate și atractive, pot fi citite în revista menționată, mai ales în numerele apărute în ultimii 3 ani.

În țările socialiste s-a acordat o importanță deosebită acestui domeniu, fiind elaborate în ultimii ani prin concepție proprie câteva zeci de modele de calculatoare personale, cu o fabricație anuală în serii care cresc rapid, de la tipurile familial și pentru școală, pînă la cele profesionale avansate, ca de pildă:

- HT10802 și PRIMO (RPU) ROBOTRON 1715 (RDG), MERITUM (RPP);
- AGAT (URSS) și PRAVETZ (RPB) compatibile APPLE;
- ISKRA 250 (URSS), SMEP (RSC), ROBOTRON (RDG), PROPER (RPU) și INTELEXT (RPB), primele două compatibile cu IBM PC, iar celelalte cu IBM PC XT.

Tabelul 2.3

MODEL DE CALCULATOR	ANUL LANSĂRII	MICRO-PROCESOR	MEMORIA INTERNĂ	PERIFERICE (OPȚIUNI)	SISTEM DE OPERARE (OPȚIUNI)	LIMBAJE/APLICAȚII
1	2	3	4	5	6	7
ZX SPECTRUM	1982	Z80A, 8 biți	16—48 k oct	Casetă magnetică (microdrive)	—	Interpretor de BASIC rezident și pe casetă, Logo, Forth, Pascal, Micro-Prolog
COMMODORE 64	1982	6510, 8 biți	64 k oct.	Casetă magnetică (disc flexibil)	Monitor (CP/M)	Interpretor de BASIC rezident, Pascal, Logo, Forth
COMMODORE 128	1984	8701 8 biți	128—256 k oct.	Disc flexibil (casetă magnetică)	CP/M	BASIC, Fortran, Pascal, Logo, Forth, dBASE II
APPLE II C	1983	6502 8 biți	64—128 k oct.	Casetă magnetică (disc flexibil, miniimprimantă)	Apple-DOS	BASIC, Pascal, Logo, Forth, Fortran
MACINTOSH	1984	68000 16 biți coprocesor matematic 68081	128 k oct —1 M oct	Discuri flexibile, Disc fix, Miniimprimantă	Apple-DOS	BASIC, Pascal, C, Forth, Fortran, etc.
MACINTOSH II	1986	68020 32 biți coprocesor matematic 68881/16 MHz	1—8 M oct	Discuri flexibile, Disc fix 40 M oct, Imprimantă cu laser	DOS, UNIX	Toate limbajele PC larg răspândite și aplicațiile standard
IBM PC și modelul extins IBM PC/XT	1981 1983	8086/8088 16 biți coprocesor matematic 8087	256—640 k oct.	Discuri flexibile Disc fix tip Winchester, Miniimprimantă	MS-DOS	BASIC, Fortran, Cobol, Pascal, C, Forth, Lisp.

2. CALCULATOARE PERSONALE

Tabelul 2.3. (continuare)

1	2	3	4	5	6	7
IBM* PC/AT	1984	80286 16 biți 10 MHz coprocesor matematic 80287	256 koct - -2 Moct	Discuri flexibile, Disc fix, Miniimprimantă	MS-DOS. XENIX	BASIC, Turbo-Pascal, Fortran 77, Cobol, C, Forth, Lisp, dBASE III, ORACLE, LOTUS 1-2-3, AUTO CAD
DESK PRO 386 (COMPAQ)	1987	80386 32 biți 16/20 MHz coprocesor matematic 80387/16 MHz	1-16 Moct.	Discuri flexibile, Disc fix 100,300 Moct., Imprimantă cu laser	MS-OS/2	Toate limbajele PC larg răspândite și aplicațiile standard

* Noua serie PERSONAL SYSTEM II cu modele de 16 și 32 de biți, lansată recent de IBM este prezentată în detaliu în capitolul 25.

Pentru țările capitaliste dezvoltate industrial s-a ales prezentarea structurii consumului general de tehnică de calcul direct și indirect, într-un grup de 6 țări, care include, SUA, Japonia, RFG, Marea Britanie, Franța și Italia în 1989, structură dată în tabelul 2.4.

Ponderea consumului general de tehnică de calcul conform tabelului 2.4., din consumul total aferent întregii industrii de electronică din cele 6 țări este de peste 66%, ceea ce ilustrează în continuare atât un aport major al industriei

Tabelul 2.4

PREVIZIUNI PRIVIND CONSUMUL CUMULAT ÎN ANUL 1989,
DE ECHIPAMENTE PENTRU PRELUCRAREA DATELOR, PROGRAME DE BAZĂ,
APLICATIVE ȘI CITEVA GRUPE DE PRODUSE BAZATE PE TEHNICA
DE CALCUL ȘI MICROPROCESOARE ÎNTR-UN GRUP* DE 6 ȚĂRI

— ponderea valorică, % —

TOTAL	100%
din care:	
1 — echipamente pentru prelucrarea datelor	66%
2 — programe de bază, aplicative și instrumente pentru aplicații	20%
(livrate distinct)	
3 — echipamente pentru comunicații de date și terminale facsimil	5%
4 — produse de consum personal bazate pe microprocesoare și circuite integrate specializate	2%

I. CALCULATOARE ÎN ȚARĂ ȘI ÎN LUME

5 — echipamente pentru controlul proceselor	6 %
6 — sisteme CAD/CAE/CAM pentru industria electronică inclusiv microelectronică (proiectare, inginerie și fabricație asistate de calculator)	1 %
* Grupul de 6 țări cuprinde: SUA, Japonia, RFG, Marea Britanie, Franța și Italia	

NOTĂ:

- ponderile din tabelele 2.4, 2.5 și 2.6 au fost calculate prin prelucrarea selectivă și agregarea datelor publicate în revista Electronics și în alte lucrări din 1988 și 1989, considerând consumul egal cu producția livrată minus export plus import.
- În revista Electronics subgrupele 3, 4, 5 și 6 sînt incluse la alte industrii din componența industriei electronice, dar în tabelul 2.4. au fost însumate pentru a ilustra mai bine impactul tehnicii de calcul în industrie.

Tabelul 2.5

STRUCTURA CONSUMULUI ÎN 1989 ÎN SUA, DE ECHIPAMENTE PENTRU PRELUCRAREA DATELOR ȘI PROGRAME, PE PRINCIPALELE GRUPE

— ponderea valorică, % —

Echipamente de prelucrare a datelor și programe, total	100 %
din care:	
— echipamente de prelucrare a datelor, total	77 %
din care:	
— sisteme de calcul	54 %
— echipamente periferice de intrare/ieșire, inclusiv grafice	12 %
— memorii externe	7 %
— terminale	4 %
— programe*de bază, aplicative și instrumente pentru aplicații, total	23 %
din care:	
— programe de sistem	12 %
(sisteme de operare, instrumente de diagnosticare și punere la punct)	
— programe aplicative	6 %
(prelucrare de texte, calcule tehnico-științifice, economice, etc.)	
— instrumente pentru aplicații	5 %
(baze de date, programe CAD/CAE/CAM pentru mecanică, programe de editare publicitară, de inteligență artificială, de prelucrare de imagini, etc.)	

* Livrări distincte ale producătorilor de software (exclusiv atît programele și microprogramele incluse în prețul sistemelor, cît și programele proprii elaborate de utilizatori).

2. CALCULATOARE PERSONALE

Tabelul 2.6

DISTRIBUȚIA CONSUMULUI ÎN 1989 DE SISTEME DE CALCUL, PE PRINCIPALELE SUBGRUPE, COMPARATIV, ÎN SUA (I) ȘI ÎNTR-UN GRUP DE 6 ȚĂRI (II) CARE INCLUDE SUA

— ponderea valorică, % —

	I*	II*
Sisteme de calcul, total	100 %	100 %
din care:		
— calculatoare personale	31 %	20 %
— microcalculatoare multiutilizator și stații de lucru	12 %	14 %
— minicalculatoare și superminicalculatoare	25 %	21 %
— calculatoare medii și medii-mari (tip mainframes)	30 %	44 %
— minisupercalculatoare și supercalculatoare	2 %	1 %

*I — SUA

II — SUA, Japonia, RFG, Marea Britanie, Franța și Italia

directe de tehnică de calcul, cât și o creștere a ponderii grupelor de produse asociate (raportate valoric în cadrul altor industrii, dar bazate pe calculatoare și microprocesoare).

În tabelul 2.5 se dă structura consumului estimat de tehnică de calcul (echipamente și programe) în 1989 în SUA, ilustrând următoarele grupe de produse cu ponderi mari:

— sisteme de calcul	— 54 %	— programe de sistem, aplicative și instrumente pentru aplicații (livrări distincte)	— 23 %
— memorii externe, periferice de intrare / ieșire și terminale	— 23 %		

Distribuția consumului de sisteme de calcul pe principalele subgrupe, conform tabelului 2.6. arată că subgrupele de calculatoare personale și microcalculatoare, considerate împreună ajung la o pondere de 43% în acest an, devenind cel mai mare sector de tehnică de calcul din SUA, în timp ce în grupul de 6 țări menționate, ponderea respectivă atinge nivelul de 34%. De altfel, începând cu anul 1988, chiar numai ponderea stațiilor de lucru de 16 și 32 de biți devine mai mare decât ponderea minicalculatoarelor clasice!

În aceste condiții, apar redistribuiri în favoarea subgrupelor cu performanțe superioare minicalculatoarelor (de fapt cu un raport performanță/cost mai bun) și anume mai ales spre subgrupa superminicalculatoarelor, dar și spre calculatoarele medii, medii-mari, minisupercalculatoare și supercalculatoare.

Cele două subgrupe apărute în ultimii ani în clasificarea adoptată pe plan mondial, pot fi definite pe scurt astfel: *superminicalculatoarele* reprezintă minicalculatoare cu performanțe de vîrf, în timp ce *minisupercalculatoarele* acoperă intervalul dintre cele mai performante supermini și calculatoarele medii-mari sau supercalculatoare. Minisupercalculatoarele au facilități de prelucrare vectorială integrate, dar costă mult mai puțin decât supercalculatoarele.

O caracteristică importantă a industriei de calculatoare personale este dată de seriile mari de fabricație, de pildă ZX Spectrum cu peste un milion și IBM PC cu citeva milioane. Se estimează că la începutul anului 1988 erau instalate numai în America de Nord peste 14 milioane de calculatoare personale dintre care 10—20% sînt conectate în rețele.

I. CALCULATOARE ÎN ȚARĂ ȘI ÎN LUME

Stațiile de lucru, avînd o dinamică mare a producției, reprezintă posturi ale activității ingineresti de înaltă productivitate, fiind dotate cu microprocesoare de 16 și 32 de biți, afișaj grafic color, discuri magnetice rapide, imprimantă grafică, plotter și mai ales cu instrumente de programare și proiectare cu facilități grafice interactive (grafică 3D) adecvate familiei respective de aplicații în electronică, mecanică, etc.). Cu puțin timp în urmă, stațiile de lucru ofereau o viteză de 4—5 milioane de operații/s. (MIPS), apoi au apărut mai multe modele de 10 MIPS și începînd cu mijlocul anului 1988 au fost anunțate primele modele de 20 MIPS, fiind în elaborare modele cu viteze apreciabil mai mari.

Memoriile externe incluse în tabelul 2.5 cuprind: unități cu discuri magnetice fixe (se mai numesc tip Winchester sau rigide, suportul magnetic nefiind interschimbabil între unități), unități cu discuri magnetice flexibile, unități cu pachet de discuri amovibile, unități cu benzi magnetice încasate și tip minicasetă, și noua subgrupă a discurilor optice.

Discurile rigide (cu suport magnetic fix cu diametrul de 3 1/2, 5 1/4, 8 sau 14 in) și discurile flexibile (cu suport magnetic flexibil cu diametrul de 3 1/2, 5 1/4 sau 8 in) au împreună o pondere valorică de aproape 84% din totalul memoriilor externe, unitățile cu disc fix avînd ponderea de 72%. Dintre aceste tipuri de discuri, cele mai des folosite în configurațiile microcalculatoarelor personale sînt unitățile cu disc fix și flexibile cu suporturi magnetice de 5 1/4 și 3 1/2 in.

Echipamentele periferice de intrare-ieșire includ: imprimante seriale, imprimante de linii de medie și mare viteză, imprimante cu laser, digitizoare, plottere, etc. Imprimantele cu impact au cea mai mare pondere valorică (38%) din echipamentele de intrare-ieșire și sînt larg răspîndite în configurațiile microcalculatoarelor, stațiilor specializate de proiectare și ale unor calculatoare personale, mai ales modele profesionale, dar și imprimantele cu laser dețin în 1989 o pondere importantă (peste 20%).

Menționăm cîteva echipamente și dispozitive periferice de mare performanță, utilizate în configurațiile calculatoarelor personale și micro, evolute:

- unități miniaturizate de disc Winchester de 10/20 Moct (numite Hardcard), montate pe o parte din placheta logică, deci încorporate în calculator, la fel ca orice plachetă;
- unități de disc optic cu laser, care permit numai citiri (utilizate de firmele IBM, DEC și APPLE) și pot avea o capacitate de 200 ÷ 600 Moct (echivalentul a 200000 pagini de text)
- imprimante cu laser, de pildă Laser Jet de la firma Hewlett-Packard și Laser

- Writer de la Apple cu viteza de tipărire de 8 pg./min. sau modelul Xerox 4045 cu 10 pg./min. la o rezoluție maximă pentru text și grafică, de 300 × 300 puncte/in²;
- display grafic cu o mare putere de rezoluție, de 640 × 480 puncte adresabile pe ecran, cu selectarea a 16 culori dintr-o gamă de 256.
- tastatură complet autonomă cu comanda în infraroșu (introdusă prima dată la modelul IBM PC Jr.)

Evoluția industriei de calculatoare personale și microcalculatoare ilustrează următoarele caracteristici și tendințe:

a) „maturizarea“ domeniului, reflectată prin:

- modificarea ritmului de creștere a vânzărilor de la 50—100% în 1980—1984, la 8—20% pentru perioada 1985—1990, ceea ce înseamnă că faza „explozivă“ a fost depășită;
- reducerea severă a numărului de firme producătoare, concentrarea producției la firmele care pot asigura compatibilitatea deplină cu modelele devenite „standard mondial“ și dezvoltă producții de mare

serie, rentabile, inclusiv în fabrici automatizate (cele mai „agresive“ produse noi au fost lansate de IBM, Apple, Compaq, Tandy și Zenith din SUA privind gama calculatoarelor personale și microcalculatoarelor evoluat tehnologic);

- dezvoltarea deosebită a producției de programe destinate aplicațiilor de o mare diversitate, bazate pe calculatoare personale și micro. O mare parte a ela-

boratorilor și furnizorilor de programe este organizată sub forma „caselor de software“, cu echipe mai mici, dar foarte productive. Modelele Sinclair Spectrum și IBM PC beneficiază, de pildă, de câteva mil de programe fiecare. Astfel, ponderea programelor aferente microcalculatoarelor și calculatoarelor personale, a ajuns în anul 1988, în SUA, la peste 16% din totalul livrărilor de software;

b) creșterea evidentă în anii 1984—1989 a ponderii valorice a calculatoarelor personale și micro din totalul sistemelor de calcul (de la circa 20 la peste 40%);

c) diversificarea programelor aplicative pe următoarele direcții:

- programe pentru educație și învățământ, instruire asistată de calculator;
- programe specializate (pentru calcule tehnico-științifice, economice, financiare, de proiectare automată, conducerea proceselor de producție);
- programe de divertisment (Jocuri logice, educaționale, etc.);
- software integrat pentru modele profesionale (incluzând programe de gra-

fică, baze de date, prelucrare și actualizare de tabele, prelucrare de texte, programe pentru telecomunicații);

- programe pentru sisteme expert implementate pe microcalculatoare profesionale și stații de lucru avansate. Pachetele de programe sînt bazate pe tehnici ale domeniului AI (Artificial Intelligence), includ cunoștințe despre sute de obiecte sau stări și oferă soluții respectînd seturi de cîteva sute de reguli.

d) elaborarea unor sisteme de operare evoluat. Sistemul de operare UNIX elaborat de firma AT & T tinde să devină începînd cu 1988/1989 un standard mondial. De altfel, în 1991 față de 1985/1986, ponderea valorică a sistemelor UNIX din totalul sistemelor de operare va fi de 4 ori mai mare. Privind sistemele noi de operare pentru microcalculatoare bazate pe microprocesorul Intel de 32 de biți, 80386, este de așteptat o competiție între versiunea UNIX-386 și noul sistem de operare 386-OS/2 lansat de firma Microsoft, dar și acesta din urmă este considerat numai un produs de tranziție pînă la apariția unui nou standard în 1989/1990.

e) atenuarea și chiar dispariția unei distincții nete între micro, mini și calculatoare medii, ca urmare a introducerii microprocesoarelor de 32 de biți. Această afirmație trebuie înțeleasă în sensul că microcalculatorul de azi, înlocuiește prin performanță minicalculatorul de ieri. Dar toate grupele actuale din cadrul sistemelor de calcul au performanțe noi și sînt adresate unor sfere noi de aplicații. De pildă, în SUA, valoarea totală anuală a instalărilor de supercalculatoare s-a dublat în ultimii trei ani, ritm extrem de rapid. De altfel, într-o perioadă de 13 ani pînă la începutul anului 1985, au fost instalate primele 100 de supercalculatoare Cray, iar în numai doi ani și jumătate pînă în septembrie 1987, se ajunsese la cel de-al 200-lea sistem livrat. Expertii au estimat că în următorii 10 ani vor fi instalate în lume cîteva mii de supercalculatoare dedicate aplicațiilor științifice și ingineresti, cu puteri de calcul de minim 100—200 milioane de operații în virgulă mobilă/secundă și la prețuri de 1—50 milioane \$ (printre cele mai rapide supercalculatoare, introduse recent, fiind Cray-3 și ETA GF 10 cu viteze de 10 miliarde de operații în virgulă mobilă/secundă). Supercalculatoarele au devenit un standard și nu o excepție.

fiind necesare pentru o serie de aplicații cu calcule laborioase privind: meteorologia, mecanica fluidelor, fizica plasmei, fizica atomică și nucleară, ingineria nucleară, sistemele ecologice, prelucrarea digitală a imaginilor recepționate de la sateliți, prelucrarea grafică și recunoașterea formelor, inventarierea resurselor terestre, modele globale ale sistemelor socio-economice, dar și pentru simulări în chimia computațională, în industriile de electronică, auto și aerospațială, respectiv în industria petrolieră.

Astfel, dacă în urmă cu 5 ani existau numai 10 programe aplicative pentru supermașinile Cray, astăzi există peste 459 de programe aplicative, iar la unele modele, instalate în universități, se pot conecta prin stații de lucru sau calculatoare personale până la 5000 de utilizatori per sistem.

Dezvoltarea domeniului tehnicii de calcul este atât de rapidă, încât începând cu anul 1988, dar mai ales din 1989, are loc o înlocuire treptată atât a superminicalcutoarelor cât și a calculatoarelor medii-mari clasice, cu 3 noi tipuri de calculatoare: cu arhitecturi paralele, specializate pentru prelucrarea on-line a tranzacțiilor și cu set redus de instrucțiuni (denumite în literatură RISC, de la Reduced — Instruction — Set — Computers).

După unii experți, introducerea cu succes a unor noi sisteme este condiționată în 1989 de îndeplinirea a cel puțin 4 criterii principale și anume:


- raport performanță / preț optim;
- sistem de operare UNIX standard;
- facilități de interconectare;
- interfețe prietenoase cu utilizatorul.

2.4. Industria românească de calculatoare personale și personal-profesionale

După anul 1976 și mai ales în ultimii 3+5 ani, s-a pus un accent deosebit pe elaborarea și asimilarea unor modele noi de calculatoare personale și personal-profesionale sau microcalculatoare. Câteva modele produse în serie, au fost incluse în enumerarea din tabelul 1.3.

În cele ce urmează, vom da o clasificare pentru calculatoarele personale și personal-profesionale românești (aflate în stadii diferite, de la prototip la producție de serie și prezentate de la simplu la complex, inclusiv modelele care nu se mai fabrică):

* O serie de elemente suplimentare despre supercalculatoare se află în grupajul de articole publicat în revista ȘTIINȚĂ ȘI TEHNICĂ Nr. 4 din 1987 și în câteva numere apărute în 1988 și 1989.



Se evidențiază astfel în țara noastră, *trei direcții și compatibilități* în domeniul calculatoarelor personale și personal-profesional, bine definite și anume:

1 — Sinclair Spectrum-BASIC

2 — sistem de operare CP/M pentru calculatoare de 8 biți

3 — sistem de operare MS-DOS sau tip UNIX-micro, pentru calculatoare de 16 biți tip IBM-PC și PC/XT.

Ținând seama de aceste compatibilități, cel mai mare număr de calculatoare personale produse în 1985—1989 este din clasa compatibil SPECTRUM, modelul HC-85 avînd seria de fabricație cea mai mare.

* În cadrul părții a XI-a se prezintă distinct caracteristicile generale ale calculato-
rului FELIX-PC.

**CALCULATOARE NUMERICE. REALIZARE FIZICĂ
—BAZELE ARITMETICE ȘI LOGICE.**

Capitolul 3 | **Baze
aritmetice.**

Calculatoarele electronice sînt contruite pe baza unor dispozitive fizice (tranzistoare, circuite integrate logice etc.), care funcționează ca elemente cu două stări (conducție/blocat), cărora le sînt asociate două niveluri de tensiune (coborît/ridicat). Informația elementară manipulată de calculator va fi astfel asociată cu cele două niveluri de tensiune: coborît (0V) și ridicat (+5V), care vor fi folosite pentru reprezentarea numerelor cu ajutorul unui limbaj binar, al cărui alfabet A constă din două caractere:

$$A = \{0,1\}$$

În acest limbaj cuvintele vor fi de forma: 0, 1, 10, 11, 100, 101, 110, 111 etc. Ele pot fi tratate ca numere binare sau drept coduri ale elementelor unei mulțimi:

$$0 \leftrightarrow a_0$$

$$1 \leftrightarrow a_1$$

$$10 \leftrightarrow a_2$$

3.1. Sistemul de numerație binar

Elementele limbajului binar pot fi folosite pentru reprezentarea numerelor în sistemul de numerație în baza doi. În acest scop fiecare element al cuvîntului va reprezenta coeficientul unei puteri a lui doi, elementul din extrema dreaptă

fiind coeficientul lui doi la puterea 0. Ponderile coeficienților (puterile lui doi) cresc de la dreapta, la stânga. Acesta este un sistem de numerație pozițional.

În cazul general un număr N va putea fi scris în baza q astfel

$$N = a_{n-1} q^{n-1} + a_{n-2} q^{n-2} + \dots + a_i q^i + \dots + a_1 q^1 + a_0 q^0$$

unde: $0 \leq a_i < q$.

sau prescurtat:

$$N_{(q)} = a_{n-1} a_{n-2} \dots a_i \dots a_1 a_0$$

Se consideră cazurile particulare, cu bazele: $q=2$ și $q=10$.

Pentru $q=10$ avem:

$$N = b_{n-1} 10^{n-1} + \dots + b_i 10^i + \dots + b_0 10^0$$

unde: $0 \leq b_i < 10$,

sau prescurtat:

$$N_{(10)} = b_{n-1} b_{n-2} \dots b_i \dots b_0$$

Pentru $q=2$ avem:

$$N = c_{p-1} 2^{p-1} + \dots + c_i 2^i + \dots + c_0 2^0$$

unde: $0 \leq c_i < 2$,

sau prescurtat:

$$N_{(2)} = c_{p-1} c_{p-2} \dots c_i \dots c_0$$

În sistemul de numerație binar, o cifră binară mai poartă numele de bit (*rang binar* — *binary digit* — în engleză).

Ca valoare numărul zecimal 1987 este interpretat astfel:

$$1 \cdot 10^3 + 9 \cdot 10^2 + 8 \cdot 10^1 + 7 \cdot 10^0 = 1000 + 900 + 80 + 7.$$

Numărul binar 1110 va fi interpretat ca valoare astfel:

$$1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 8 + 4 + 2 + 0 = 14$$

El va reprezenta în binar numărul întreg 14, în baza 10.

Un număr binar cu n biți permite reprezentarea unor numere întregi în gama: $0 \leq N \leq 2^n - 1$.

EXEMPLE:

— un cuvânt de 8 biți permite reprezentarea întregilor între 0 și $2^8 - 1 = 255$.

— un cuvânt de 16 biți permite reprezentarea întregilor între 0 și $2^{16} - 1 = 65535$.

Numărul de biți n , necesari într-un cuvânt binar, pentru a reprezenta un număr întreg N , dat în baza 10, se poate calcula cu ajutorul formulei:

$$n = \lceil \log_2 N \rceil \text{ biți.}$$

unde simbolul $\lceil \rceil$ reprezintă valoarea întreagă imediat superioară lui $\log_2 N$.

II. CALCULATOARE NUMERICE

Practic, n se poate calcula prin încadrarea lui N între puterile lui doi.

$$2^{n-1} < N \leq 2^n$$

Dacă $N=1987$, rezultă:

$$N = \lceil \log_2 N \rceil = 11$$

sau:

$$2^{10} = 1024 < 1987 < 2^{11} = 2048$$

3.2. Conversia binar – zecimală

Conversia binar-zecimală se poate realiza pe baza următoarei secvențe de operații (algoritm):

1. Se execută o poziționare la dreapta numărului.
2. Se inițiază: $N=0$ și $n=0$.
3. Se citește următorul caracter din stînga.
4. Dacă caracterul este un spațiu liber (blanc), atunci STOP.
5. Dacă caracterul este zero, atunci se incrementează n cu o unitate ($n+1$) și se trece la 3; dacă caracterul este unu, atunci se calculează $N+2^n$, se incrementează n cu o unitate ($n+1$) și se trece la 3.

EXEMPLU

100111 reprezintă, în zecimal, numărul: $N=1+2+4+32=39$

Numerele binare terminate în zero sînt numere pare, în timp ce numerele binare terminate în 1 sînt numere impare.

3.3. Conversia zecimal – binară.

Această conversie se realizează prin împărțiri succesive ale cîtului, de la împărțirea precedentă, la noua bază — doi, și reținerea restului curent, pînă cînd cîtul curent devine mai mic decît noua bază. Ca prim cît se ia numărul N . Resturile obținute reprezintă cifrele numărului binar, în ordinea crescătoare a ponderilor. Cifra cu ponderea cea mai mare este primul cît obținut, cu valoarea mai mică decît doi.

Fie numărul întreg N , în baza 10. El poate fi convertit în binar astfel:

$$N = 2 \cdot q_0 + r_0, \quad 0 \leq r_0 \leq 1.$$

Dacă: $q_0 \geq 2$, atunci:

$$q_0 = 2 \cdot q_1 + r_1, \quad 0 \leq r_1 \leq 1.$$

3. BAZE ARITMETICE

Dacă: $q_1 \geq 2$, atunci:

$$q_1 = 2 \cdot q_2 + r_2, \quad 0 \leq r_2 < 2.$$

Dacă: $q_1 < 2$, atunci:

$$N = q_1 \cdot 2^1 + \dots + r_2 \cdot 2^2 + r_1 \cdot 2^1 + r_0$$

Fie numărul $N=37$. Operațiile implicate de conversie vor fi următoarele:

$$\begin{array}{r}
 37 \quad | \quad 2 \\
 r_0 = 1 \quad 18 \quad | \quad 2 \\
 r_1 = 0 \quad 9 \quad | \quad 2 \\
 r_2 = 1 \quad 4 \quad | \quad 2 \\
 r_3 = 0 \quad 2 \quad | \quad 2 \\
 r_4 = 0 \quad q_0 = 1
 \end{array}$$

rezulta:

$$N_2 = 100101$$

3.4. Reprezentările octală și hexazecimală

Numerele binare sînt practic greu de manipulat, avînd prea multe ranguri. În scopul scrierii lor sub o formă condensată, se folosesc sistemele de reprezentare octal și hexazecimal. Avînd ca baze puteri ale lui doi (8 și respectiv 16), ele permit o conversie mecanică, fără calcule, între reprezentările octală/hexazecimală și respectiv binare.

Sistemul octal folosește un alfabet A cu 8 cifre:

$$A = \{0, 1, 2, 3, 4, 5, 6, 7\}$$

Conversia binar-octală este următoarea:

binar	octal	binar	octal
000	0	100	4
001	1	101	5
010	2	110	6
011	3	111	7

Numărul 110101 se scrie în octal sub forma 65 deoarece: ,

$$110 = 6_{(8)}, \text{ iar } 101 = 5_{(8)}.$$

Conversia binară a unui număr octal rezultă prin înlocuirea fiecărei cifre octale cu triada binară echivalentă.

Sistemul hexazecimal folosește un alfabet A, cu 16 cifre:

$$A = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}$$

Mai jos se dau unele echivalențe între numere zecimale, hexazecimale și binare:

ZECIMAL	HEXAZECIMAL	BINAR
0	0	0000
1	1	0001
2	2	0010
.....		
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111
16	10	0001 0000
17	11	0001 0001

Conversia unui număr binar în hexazecimal se realizează prin împărțirea numărului binar în tetrade, începând de la dreapta spre stânga, și înlocuirea lor cu cifrele hexazecimale corespunzătoare:

$$10001101 = 8D_{(16)}$$

3.5. Codul binar-zecimal (2-10)

Cifrele sistemului de numerație zecimal pot fi înlocuite cu tetrade binare, rezultând astfel posibilitatea de a reprezenta direct în calculatoare cifrele zecimale, fără a fi necesară o conversie prealabilă.

Astfel, cifrele zecimale au următoarele tetrade binare echivalente:

cifra zecimală	tetrada binară
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Tetradele 1010, 1011, 1100, 1101, 1110, 1111 sînt incorecte, neavînd cifre zecimale asociate.

EXEMPLE:

65 **-01100101**
88 **-10001000**

Conversia „binar — zecimal” — „zecimal” se realizează mecanic.

3. BAZE ARITMETICE

3.6. Adunarea și înmulțirea numerelor binare

Adunarea are loc pe baza regulilor concretizate în următorul tabel:

$$\begin{aligned} 0+0 &= 0 \\ 0+1 &= 1 \\ 1+0 &= 1 \\ 1+1 &= 10 \end{aligned}$$

În ultima linie se constată apariția transportului egal cu unu.

EXEMPLU:

$\begin{array}{ccccccc} & 1 & 1 & 1 & & 1 & \\ & \swarrow & \swarrow & \swarrow & & \swarrow & \\ & 1 & 1 & 1 & 0 & 1 & \\ + & 1 & 0 & 1 & 0 & 1 & \\ \hline 1 & 1 & 0 & 0 & 1 & 0 & \end{array}$	$\begin{array}{r} 29 \\ +21 \\ \hline 50 \end{array}$
---	---

În partea superioară au fost marcați biții de transport la adunarea fiecărui rang.

Dacă numărul de biți, din cuvintele binare, este fixat la o valoare dată n (5 în cazul de mai sus), la adunare poate apărea pericolul depășirii capacității de reprezentare, când suma obținută este mai mare decât $2^5 - 1$ (31 — în exemplul de mai sus).

Înmulțirea numerelor binare se bazează pe regulile din tabelul de mai jos:

$$\begin{aligned} 0 \times 0 &= 0 \\ 0 \times 1 &= 0 \\ 1 \times 0 &= 0 \\ 1 \times 1 &= 1 \end{aligned}$$

Algoritmul înmulțirii a două numere binare este următorul:

1. Se inițializează $N=0$
2. Se inițializează rezultatul (produsul) $P=0$.
3. Se examinează cifrele înmulțitorului începând de la dreapta spre stînga.
4. Dacă cifra curentă este absentă (blanc), atunci STOP.
5. Dacă cifra curentă este 0, se incrementează N cu o unitate ($N+1$) și se trece la 3.
6. Dacă cifra curentă este 1, de înmulțitul, deplasat cu N ranguri spre stînga, se adună la P ; se incrementează N cu o unitate și se trece la 3.

EXEMPLU:

$\begin{array}{r} 1101 \\ \times 101 \\ \hline 1101 \\ 1101 \\ \hline 100001 \end{array}$	$\begin{array}{r} 13 \times \\ 5 \\ \hline 65 \end{array}$
---	--

Înmulțirea a două numere binare de câte n biți generează un rezultat cu $2n$ biți.

3.7. Reprezentarea numerelor negative

3.7.1. Reprezentarea numerelor în modul și semn.

În scopul reprezentării semnului unui număr binar, se va alocă bitul plasat în extrema stângă a numărului dat.

Astfel, dacă pentru reprezentarea numărului se folosesc n biți, bitul de rang $(n-1)$ va fi folosit pentru codificarea semnului conform convenției de mai jos:

$$\begin{aligned} & \text{„+“} \leftarrow \text{„0“} \\ & \text{„-“} \leftarrow \text{„1“} \end{aligned}$$

Fie numerele $+6$ și -6 , reprezentate în binar, cu câte 8 ranguri, inclusiv rangul de semn:

$$\begin{array}{rcl} & 00000110 & \text{reprezintă } +6 \\ \text{semn} \leftarrow \begin{array}{c} \updownarrow \\ \updownarrow \end{array} & & \\ & 10000110 & \text{reprezintă } -6 \end{array}$$

Acest mod de reprezentare are o serie de dezavantaje:

- biții de semn ai numerelor care participă la operația aritmetică dată, trebuie tratați separat,
- apare o configurație de număr binar, 1000...00, care trebuie interpretată ca zero (zero negativ), existând astfel două reprezentări pentru zero.
- se impune definirea unei operații de scădere a valorilor absolute ale numerelor.

3.7.2. Reprezentarea în complementul față de 1.

Numărul negativ în complementul față de 1 se va obține prin înlocuirea fiecărei cifre binare a numărului dat, prin complementul ei față de 1.

Astfel, complementul față de 1, al lui 0, este 1, iar complementul față de 1, al lui 1, este 0.

Un număr pozitiv are în complementul față de 1 aceeași reprezentare ca în modul și semn.

EXEMPLU:

$$\begin{aligned} & +5_{(10)} \leftarrow \rightarrow 0101 \\ & -5_{(10)} \leftarrow \rightarrow 1010 \end{aligned}$$

Se constată că bitul cel mai semnificativ este folosit pentru reprezentarea semnului.

Adunarea bit cu bit a unui număr binar și a complementului său față de 1, generează un rezultat egal cu zero, reprezentat printr-un număr ai cărui biți sînt egali cu 1.

Astfel, la adunarea lui $+5$ cu -5 se obține: 1111, ceea ce reprezintă complementul față de 1 al lui 0000.

În această reprezentare există două forme pentru zero ($+0$ și -0), ceea ce constituie un inconvenient în realizarea circuitelor electronice pentru operațiile de adunare.

Se observă că operația de scădere se reduce la o operație de adunare la scăzut a complementului față de 1 al scăzătorului.

La adunarea numerelor reprezentate în complementul față de 1, transportul care apare la stînga rangului de semn se va aduna ciclic la ultimul bit din dreapta, al rezultatului. Rangurile de semn ale numerelor participă și ele la operația de adunare.

EXEMPLU:

$$\begin{array}{r} (+6) \\ +(-5) \\ \hline 1 \end{array} \qquad \begin{array}{r} 0110 \\ +1010 \\ \hline 10000 \\ | \quad +\uparrow \\ \hline 0001 \end{array}$$

Pentru a evita neajunsurile acestei reprezentări s-a recurs la folosirea complementului față de 2.

3.7.3. Reprezentarea numerelor în complementul față de 2.

Se consideră operația de scădere a cifrelor zecimale $8-6=2$. Dacă, în loc de -6 , la 8 se aduna complementul față de 10 al lui 6 (adică 4) se va obține: $8+4=12$. Rezultatul (2) va fi corect, dacă se va neglija transportul (1).

În mod similar, în binar, numerele negative vor fi înlocuite prin complementul față de 2. Acesta se obține (pentru numerele negative) generînd complementul față de 1 și adunînd o unitate în rangul cel mai puțin semnificativ al rezultatului.

Fie numărul binar: $-0001001.$

Complementul față de 1 va fi: 11110110

Complementul față de 2: 11110111 se va obține prin adu-

narea unei unități în rangul cel mai puțin semnificativ al complementului față de 1, al numărului dat (11110110).

Într-un cod binar cu patru cifre, reprezentările complementelor față de 2, ale numerelor $+6$ și -6 vor fi:

$$\begin{array}{r} +6 \\ -6 \end{array} \qquad \begin{array}{r} 0110 \\ 1010 \end{array}$$

Dacă se efectuează adunarea, se obține:

$$\begin{array}{r} (+6) \\ +(-6) \\ \hline 0 \end{array} \qquad \begin{array}{r} 0110 \\ +1010 \\ \hline 10000 \\ \uparrow \\ \text{se neglijează.} \end{array}$$

Se constată că zero are o singură reprezentare în complementul față de 2.

Adunarea codurilor complementare a două numere se face rang cu rang, inclusiv rangurile de semn, neglijîndu-se transportul în afara rangului de semn.

Mai jos se dau numerele binare cu patru biți, care codifică în complementul față de doi, numerele întregi cuprinse între -8 și $+7$.

1000	-8	0111	7
1001	-7	0110	6
1010	-6	0101	5
1100	-4	0100	4
1011	-5	0011	3
1101	-3	0010	2
1110	-2	0001	1
1111	-1	0000	0

Prin definiție se consideră 1000 reprezentarea binară, în complementul față de 2, a numărului 8.

Complementul față de doi este utilizat în reprezentarea numerelor, în toate calculatoarele moderne.

Cu ajutorul a n biți se pot reprezenta, în complementul față de doi, numerele N , cuprinse în gama:

$$-2^{n-1} \leq N \leq 2^{n-1} - 1$$

EXEMPLE:

pentru $n=8$, rezultă: $-2^7 \leq N \leq 2^7 - 1$ sau $-128 \leq N \leq 127$.

pentru $n=16$, rezulta: $-2^{15} \leq N \leq 2^{15} - 1$ sau $-32768 \leq N \leq 32767$.

3.8. Reprezentarea numerelor reale

În prelucrarea numerelor reale se impune definirea unor modalități de reprezentare a părților întregă și subunitară, precum și a virgulei.

În practică se întâlnesc modurile de reprezentare în virgulă fixă și în virgulă mobilă.

3.8.1. Reprezentarea în virgulă fixă.

Reprezentarea în virgulă fixă presupune, în cazul general, existența unei părți întregi și a uneia subunitare, despărțite printr-o virgulă a cărei poziție este fixă. De asemenea, mai sînt fixați numărul de biți afectați fiecărei părți.

Dacă se consideră un cuvînt de opt biți, din care primii patru sînt folosiți de partea întregă și ultimii patru, de partea subunitară, vom avea:

$$N = a_3 \cdot 2^3 + a_2 \cdot 2^2 + a_1 \cdot 2^1 + a_0 \cdot 2^0 + a_{-1} \cdot 2^{-1} + a_{-2} \cdot 2^{-2} + a_{-3} \cdot 2^{-3} + a_{-4} \cdot 2^{-4}$$

În cazul în care $N > 0$, gama de reprezentare va fi:

$$0000,0000 \leq N \leq 1111,1111 \quad \text{sau} \\ 0,0 \leq N \leq 15,9375.$$

Dacă N ia valori, atît pozitive, cît și negative, folosind bitul cel mai semnificativ, pentru codificarea semnului, gama de reprezentare va fi:

$$-8,9375 \leq N \leq 7,9375.$$

Se constată astfel, pentru un număr dat de ranguri binare, o gamă relativ restrinsă în care se pot reprezenta numerele reale. Numărul de ranguri este limitat din considerente economice și constructive.

În calculatoarele moderne poziția virgulei este plasată la dreapta numărului, operându-se numai cu întregi, în formatul cu virgula fixă. Tratarea numerelor subunitare necesită introducerea unor factori de scară în programul care se execută pe calculator.

3.8.2. Reprezentarea în virgulă mobilă.

În acest caz, un număr fracționar dat, de exemplu:

$$143,73675 \times 10^2$$

se va reprezenta sub forma unei mantise înmulțită cu 10 la un anumit exponent. Pentru exemplul dat există mai multe forme echivalente:

$$143,73675 \times 10^2$$

$$0,14373675 \times 10^5$$

$$0,014373675 \times 10^6$$

$$14373675 \times 10^{-3}$$

În funcție de valoarea exponentului, se deplasează și poziția virgulei — de aici și denumirea de virgulă mobilă.

Pentru a înlătura eventualele ambiguități se va considera întotdeauna mantisia subunitară și normalizată la dreapta, adică cifra plasată imediat la dreapta virgulei va fi diferită de zero.

În exemplul de mai sus forma corectă, normalizată va fi:

$$0.14373675 \times 10^5$$

Astfel, în reprezentarea în virgulă mobilă, numărul va fi caracterizat prin mantisa, afectată de semn, și prin exponent, de asemenea, afectat de semn.

În binar, în mod similar, numărul se va scrie sub forma unei mantise normalizată, înmulțită cu doi, la un exponent corespunzător.

De exemplu numărul:

$$10111.1011$$

$$0.101111011 \times 10 \uparrow 0101^*$$

unde mantisa este: 0.101111011 , iar exponentul are valoarea: 0101 .

Atât mantisa, cât și exponentul sînt pozitive, fapt indicat prin valoarea „0” plasată în rangurile de semn.

În concluzie, un număr în virgula mobilă este caracterizat printr-un număr binar, pozitiv sau negativ, subunitar (cu virgula plasată la dreapta rangului de semn), în valoare absolută $\geq 0,5$, numit mantisă, și printr-un număr binar întreg, pozitiv sau negativ, reprezentînd exponentul.

* Simbolul \uparrow semnifică ridicarea la putere iar, virgula zecimală va fi reprezentată prin punct.

De exemplu, într-o implementare dată (biblioteca aritmetică în virgulă mobilă pentru microcalculatorul HC-85), pentru exponent pot fi alocați 8 biți (un octet sau byte), în care este inclus și semnul, iar pentru mantisa 32 de biți (patru octeți), inclusiv semnul. Aceasta va permite manipularea unor numere cu o precizie de 9–10 cifre zecimale, într-o gamă de valori cuprinse între $10 \uparrow 38$ și $4.10 \uparrow -39$.

3.9. Reprezentarea caracterelor alfanumerice

Calculatoarele prelucrează, atât informație numerică, cât și texte. Aceasta impune folosirea unor coduri binare pentru reprezentarea cifrelor, literelor și a altor informații cu caracter de comandă, semne aritmetice, de punctuație etc.

Pentru codificarea literelor (mari și mici) și a cifrelor, se poate folosi un cod binar de 7 biți ($2^7 = 128$ semne diferite).

Din motive legate de creșterea fiabilității s-a introdus și cel de-al optulea bit, numit bit de paritate. Acesta se adaugă automat la codul de 7 biți, al caracterului, pentru a face ca numărul total de unități, din codul binar respectiv, să fie par sau impar. Odată convenția aleasă, se pot face verificările de paritate pe parcurs, pentru a constata o eventuală eroare în transmisia datelor.

În practică, pentru caractere, se folosesc două tipuri de coduri ASCII și EBCDIC.

Mai jos se prezintă codul ASCII

Caracter sau Comanda	ASCII (hexa)	Caractere speciale	ASCII (hexa)	Caractere majuscule	ASCII (hexa)	Caractere mici	ASCII (hexa)
NUL	00	(SP) spațiu	20	e	40		60
SOH	01	!	21	A	41	a	61
STX	02	"	22	B	42	b	62
ETX	03	#	23	C	43	c	63
EOT	04	\$	24	D	44	d	64
ENQ	05	%	25	E	45	e	65
ACK	06	&	26	F	46	f	66
BEL	07	'	27	G	47	g	67
BS	08	(28	H	48	h	68
HT	09)	29	I	49	i	69
LF	0A	*	2A	J	4A	j	6A
VT	0B	+	2B	K	4B	k	6B
FF	0C	,	2C	L	4C	l	6C
CR	0D	—	2D	M	4D	m	6D
SO	0E	.	2E	N	4E	n	6E
SI	0F	/	2F	O	4F	o	6F

DLE	10	0	30	P	50	p	70
DC1	11	1	31	Q	51	q	71
DC2	12	2	32	R	52	r	72
DC3	13	3	33	S	53	s	73
DC4	14	4	34	T	54	t	74
NAK	15	5	35	U	55	u	75
SYN	16	6	36	V	56	v	76
ETB	17	7	37	W	57	w	77
CAN	18	8	38	X	58	x	78
EM	19	9	39	Y	59	y	79
SUB	1A	:	3A	Z	5A	z	7A
ESC	1B	;	3B	[5B	{	7B
FS	1C	<	3C	 	5C	!	7C
GS	1D	=	3D]	5D	}	7D
RS	1E	>	3E	^	5E	~	7E
US	1F	?	3F	—	5F	DEL	7F

4.1. Generalități

În sensul cel mai larg, logica este știința care studiază formele și legile gândirii, iar logica matematică este știința care utilizează metodele matematice pentru soluționarea problemelor de logică.

Domeniul logicii matematice, intitulat „algebra logicii“, a găsit o largă aplicabilitate în analiza și sinteza schemelor care intră în componența calculatoarelor numerice.

Algebra logicii operează cu aserțiuni, reprezentând afirmații despre care se poate spune că sînt fie adevărate, fie false.

Aserțiunile de tipul: „BASIC este un limbaj de programare conversațional“, „sistemul de numerație binar folosește două simboluri pentru reprezentarea cifrelor“, „la nivelul mării apa fierbe la 90°C“ etc., sînt evaluate numai prin prisma adevărului sau falsității lor, independent de conținut. Unei aserțiuni adevărate i se atribuie valoarea 1, iar unei aserțiuni false — valoarea 0. O aserțiune nu poate fi în același timp falsă și adevărată. Două aserțiuni sînt echivalente, din punctul de vedere al algebrei logicii, dacă sînt simultan adevărate sau false.

Aserțiunile pot fi simple și compuse, cele din urmă obținîndu-se ca rezultat al reunirii lor cu ajutorul unor legături logice de tipul conjuncției, disjuncției, negației etc.

Pentru simplificarea calculului cu aserțiuni, algebra logicii folosește metoda simbolică, înlocuind disjuncția, conjuncția și negația cu simbolurile: „ \cup “, „ \cap “ și „ \neg “, iar aserțiunile simple și compuse cu simboluri literale — litere.

În cadrul algebrei logicii aserțiunile simple și cele compuse iau valori din mulțimea bivalentă $[0, 1]$.

Aserțiunile compuse mai poartă numele și de funcții logice. Ele pot fi reprezentate prin tabele de adevăr, care conțin valorile variabilelor (aserțiunile simple), ca argumente, și valorile corespunzătoare ale funcției. În general funcțiile logice se mai pot reprezenta cu ajutorul „formelor canonice“, diagramelor Karnaugh, Veitch etc.

Adevărul sau falsitatea unei aserțiuni compuse sînt determinate de valoarea aserțiunilor simple, din care se compune, precum și de natura legăturilor logice.

Legătura logică numită disjuncție se poate pune în evidență în structura unei aserțiuni compuse, notată cu y , ca funcție de două aserțiuni simple, notate cu x_0 și x_1 :

$$y \text{ este } x_0 \text{ sau } x_1,$$

ceea ce se mai poate scrie sub formă simbolică astfel:

$$y = x_0 \cup x_1$$

În mod similar, conjuncția și negația se pot pune în evidență prin aserțiuni compuse de tipul:

y este x_0 și x_1 , adică:

$$y = x_0 \cap x_1,$$

respectiv:

y este non x , sau

$$y = \bar{x}$$

În tabela 4.1 sînt date, sub forma de tabele de adevăr, funcțiile logice: disjuncția, conjuncția și negația.

Tabela 4.1

TABELELE DE ADEVĂR PENTRU: DISJUNCȚIE, CONJUNCȚIE ȘI NEGAȚIE

Disjuncția		
x_1	x_0	y
0	0	0
0	1	1
1	1	1
1	1	1

Conjuncția		
x_1	x_0	y
0	0	0
0	1	0
1	0	0
1	1	1

Negația	
x	y
0	1
1	0

Datorită asemănării între operațiile aritmetice de adunare și înmulțire, pe de-o parte, și operațiile (legăturile) logice, disjuncție și conjuncție, pe de altă parte, acestea din urmă au mai primit denumirile de: sumă logică și respectiv — produs logic.

Dacă două funcții logice, de aceleași variabile, iau valori identice, pentru aceleași seturi de variabile, ele sînt identice.

De exemplu:

$$\overline{(\bar{y})} = y$$

reprezintă identitatea între funcția logică inițială și dubla ei negație.

Suma logică și produsul logic se bucură de proprietățile de: comutativitate:

$$x_1 \cup x_0 = x_0 \cup x_1$$

$$x_1 \cap x_0 = x_0 \cap x_1$$

asociativitate:

$$(x_2 \cup x_1) \cup x_0 = x_2 \cup (x_1 \cup x_0)$$

$$(x_2 \cap x_1) \cap x_0 = x_2 \cap (x_1 \cap x_0)$$

distributivitate:

$$x_2 \cup (x_1 \cap x_0) = (x_2 \cup x_1) \cap (x_2 \cup x_0)$$

$$x_2 \cap (x_1 \cup x_0) = (x_2 \cap x_1) \cup (x_2 \cap x_0)$$

4.2. Formele canonice ale funcțiilor logice

Funcțiile logice se pot reprezenta, fie prin tabele de adevăr, fie prin așa-numitele forme canonice. O funcție logică de mai multe variabile trebuie să fie definită pentru toate combinațiile posibile ale variabilelor.

După cum se poate constata ușor, o funcție logică de n variabile logice trebuie să fie definită pentru toate cele 2^n combinații (seturi de valori) ale variabilelor. Astfel, tabela de adevăr va avea $n+1$ coloane și 2^n linii.

În tabela 4.1 disjuncția și conjuncția, ca funcții de două variabile, sînt definite pentru $2^2=4$ combinații posibile ale variabilelor, în timp ce negația, ca funcție de o singură variabilă este definită pentru $2^1=2$ valori ale acesteia.

Manipularea funcțiilor logice cu ajutorul tabelelor de adevăr devine greoaie, mai ales în scopul efectuării unor simplificări.

Formele canonice oferă, în această privință, numeroase avantaje.

Examinînd tabela de adevăr pentru conjuncție, se constată că funcția y_2 ia valoarea 1 numai cînd x_1 și x_2 sînt simultan 1, ceea ce se mai poate scrie și sub forma de produs:

$$y_2 = x_1 \cap x_2^*)$$

Plecînd de la tabela de adevăr a disjuncției, funcția corespunzătoare y_1 se mai poate scrie și sub forma unei disjuncții de produse logice:

$$y_1 = \bar{x}_1 x_0 \cup x_1 \bar{x}_0 \cup x_1 x_0$$

Astfel, o tabelă de adevăr, pentru o funcție de n variabile, poate fi interpretată ca o sumă logică a produselor logice P_i (în care intervin n variabile diferite, unele eventual negate), pentru care funcția y ia valoarea 1:

$$y = \bigcup_{i=0}^{2^n-1} a_i \cdot P_i$$

unde:

— produsul logic P_i , de forma $x_0 \bar{x}_1 \dots x_i \bar{x}_{i+1} \dots x_{n-1}$, poartă numele de **minterm** sau **constituent** al unității. Indicele i are valoarea zecimală a numărului binar cu n ranguri, obținut prin înlocuirea variabilelor negate cu 0 și a celor fără negație cu 1, în expresia lui P_i . Mintermul P_i ia valoarea 1 numai pentru produsul logic i al variabilelor;

* În continuare, pentru conjuncție nu se va mai folosi simbolul "∩". Expresia constituită din două sau mai multe variabile scrise alăturat va semnifica produsul logic al acestor variabile.

— coeficientul a_i ia valorile 0 sau 1, după cum funcția ia valorile 0 sau 1, pentru setul i al variabilelor.

Pentru disjuncție se poate scrie:

$$y_1 = 0 \cdot P_0 \cup 1 \cdot P_1 \cup 1 \cdot P_2 \cup 1 \cdot P_3 \text{ sau}$$

$$y_1 = 0 \cdot \bar{x}_1 \bar{x}_0 \cup 1 \cdot \bar{x}_1 x_0 \cup 1 \cdot x_1 \bar{x}_0 \cup 1 \cdot x_1 x_0$$

Forma de exprimare a unei funcții logice prin suma logică a mintermilor se mai numește forma disjunctiv-normal-perfectă (FDNP).*

În mod similar, o funcție logică se mai poate exprima și sub forma conjunctiv-normal-perfectă (FCNP)**, efectuînd conjuncția tuturor sumelor logice S_i (în care intervin toate variabilele, eventual unele negate) pentru care funcția logică ia valoarea 0.

Suma logică $S_i = x_0 \cup x_1 \cup \dots \cup x_i \cup \bar{x}_{i+1} \cup \dots \cup x_{n-1}$ mai poartă numele de maxterm sau constituent al zeroului. El ia valoarea zero numai pentru disjuncția de rang i a tuturor variabilelor, în care variabila x va fi fără negație, dacă valoarea ei binară, în setul dat, este 0 și va fi negată în caz contrar.

În cazul general FCNP a unei funcții logice se mai poate scrie:

$$y = \bigcup_{i=0}^{2^n-1} (a_i \cup S_i)$$

în care a_i ia valoarea 0 sau 1, după cum funcția y are valoarea 0 sau 1, pentru setul i de variabile.

În cazul conjuncției a două variabile rezultă:

$$y_2 = (0 \cup x_1 \cup x_0)(0 \cup x_1 \cup \bar{x}_0)(0 \cup \bar{x}_1 \cup x_0)(1 \cup \bar{x}_1 \cup \bar{x}_0)$$

sau:

$$y_2 = (x_1 \cup x_0)(x_1 \cup \bar{x}_0)(\bar{x}_1 \cup x_0),$$

deoarece: $1 \cup \bar{x}_1 \cup \bar{x}_0 = 1$.

Întrucît în FDNP și FCNP fiecare din cei 2^n coeficienți a_i (n este numărul de variabile independente) poate lua valorile 0 sau 1, vor exista în total 2^{2^n} funcții de n variabile. Aceste funcții se notează cu f_k , unde k este valoarea zecimală a numărului format din coeficienții:

$$a_{2^n-1} \ a_{2^n-2} \ \dots \ a_1 a_0$$

Fie funcția f dată prin tabela de mai jos:

x_2	x_1	x_0	f
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

*, ** Aceste forme se numesc perfecte deoarece conțin produse sau sume logice în care intră toate variabilele. Ele se numesc normale întrucît semnul negației nu se extinde decît la expresii formate dintr-o singură variabilă.

În FDNP ea se va scrie astfel:

$$f = \bar{x}_2 \bar{x}_1 x_0 \cup \bar{x}_2 x_1 \bar{x}_0 \cup x_2 \bar{x}_1 x_0 \cup x_2 x_1 x_0$$

în timp ce în FCNP, va avea următorul aspect:

$$f = (x_2 \cup x_1 \cup x_0)(x_2 \cup \bar{x}_1 \cup \bar{x}_0)(\bar{x}_2 \cup x_1 \cup x_0)(\bar{x}_2 \cup \bar{x}_1 \cup x_0).$$

În tabelele 4.2 și 4.3 sînt prezentate funcțiile logice de o variabilă și de două variabile.

Tabela 4.2

FUNCȚII LOGICE DE O VARIABILĂ

$f_k \backslash x$	0	1	Reprezentare	Denumire
f_0	0	0	0	Constanta 0
f_1	0	1	x	Variabila x
f_2	1	0	\bar{x}	Negația lui x
f_3	1	1	1	Constanta 1

Tabela 4.3

FUNCȚII LOGICE DE DOUĂ VARIABILE

$f_k \backslash \begin{matrix} x_1 & 0 & 0 & 1 & 1 \\ x_0 & 0 & 1 & 0 & 1 \end{matrix}$	a_0	a_1	a_2	a_3	Reprezentare	Denumire
f_0	0	0	0	0	0	Constanta 0
f_1	0	0	0	1	$x_1 \cdot x_0$	Conjuncția „SI“
f_2	0	0	1	0	$x_1 \nabla x_0$	Interdicția după x_0
f_3	0	0	1	1	x_1	Variabila x_1
f_4	0	1	0	0	$x_0 \nabla x_1$	Interdicția după x_1
f_5	0	1	0	1	x_0	Variabila x_0
f_6	0	1	1	0	$x_1 \oplus x_0$	Suma modulo 2 (SAU-EX)
f_7	0	1	1	1	$x_1 \cup x_0$	Disjuncția „SAU“
f_8	1	0	0	0	$x_1 \uparrow x_0$	SAU – NU (NOR)
f_9	1	0	0	1	$x_1 \equiv x_0$	Identitatea logică
f_{10}	1	0	1	0	\bar{x}_0	Negația lui x_0
f_{11}	1	0	1	1	$x_0 \rightarrow x_1$	Implicația
f_{12}	1	1	0	0	\bar{x}_1	Negația lui x_1
f_{13}	1	1	0	1	$x_1 \rightarrow x_0$	Implicația
f_{14}	1	1	1	0	$x_1 \downarrow x_0$	ȘI – NU (NAND)
f_{15}	1	1	1	1	1	Constanta 1

4. LOGICA ȘI CIRCUITE LOGICE

4.3. Identități și simplificări

Formele canonice, în mod frecvent, nu reprezintă expresiile cele mai simple ale unei funcții logice. Spre exemplu, forma canonică a disjuncției (FDNP) a două variabile este:

$$\bar{x}_1x_0 \cup x_1\bar{x}_0 \cup x_1x_0 \text{ și nu } x_1 \cup x_0, \text{ conform definiției.}$$

Expresiile logice se pot simplifica folosind identitățile date mai jos:

$$1. x \cup 0 = x$$

$$2. x \cdot 0 = 0$$

$$3. x \cup 1 = 1$$

$$4. x \cdot 1 = x$$

$$5. x \cup x = x$$

$$6. x \cdot x = x$$

$$7. x \cup \bar{x} = 1$$

$$8. \overline{\overline{x}} = x$$

$$10. x_1 \cup x_0 = x_0 \cup x_1$$

$$11. x_1 \cdot x_0 = x_0 \cdot x_1$$

$$12. (x_2 \cup x_1) \cup x_0 = x_2 \cup (x_1 \cup x_0)$$

$$13. (x_2 \cdot x_1)x_0 = x_2(x_1 \cdot x_0)$$

$$14. x_0(x_1 \cup x_2) = x_0 \cdot x_1 \cup x_0 \cdot x_2$$

$$15. x_0 \cup (x_1 \cdot x_2) = (x_0 \cup x_1)(x_0 \cup x_2)$$

$$16. \overline{x_1x_0} = \bar{x}_1 \cup \bar{x}_0$$

$$17. \overline{x_1 \cup x_0} = \bar{x}_1\bar{x}_0$$

Identitățile 1—9 sînt evidente. Identitățile 10—15 exprimă proprietățile de comutativitate, asociativitate și distributivitate ale funcțiilor logice. Identitățile 16 și 17 poartă numele de legile lui De Morgan. Ele se pot demonstra simplu cu ajutorul tabelor de adevăr, arătînd că, pentru aceleași valori ale variabilelor x_0 și x_1 , termenii stîng și drept, din egalitățile 16 și 17, iau valori egale.

4.4. Realizarea fizică a funcțiilor logice

4.4.1. Circuite logice combinaționale.

În capitolul referitor la elementele de aritmetică pentru calculatoarele numerice s-a văzut că operațiile aritmetice de adunare și înmulțire în binar se realizează pe baza unor reguli foarte simple, concretizate în tabele corespunzătoare.

Operațiile de adunare și înmulțire a două numere binare x_i și y_i , de cîte un rang, pot fi descrise cu ajutorul următoarelor reguli, unde: S_i este rangul i al sumei, T_{i+1} este transportul în rangul $i+1$ al sumei, P_i este rangul i al produsului:

a) adunarea:

- dacă x_t și y_t sînt zero, suma S_t este zero, transportul T_{t+1} este zero;
- dacă x_t sau y_t este unu (nu simultan), suma S_t este unu, transportul T_{t+1} este zero;
- dacă x_t și y_t sînt unu, suma S_t este zero, transportul T_{t+1} este unu.

b) înmulțirea:

- dacă x_t sau y_t este zero, produsul P_t este zero;
- dacă x_t și y_t sînt unu produsul P_t este unu.

Sub formă de tabelă, regulile de mai sus devin:

x_t	y_t	S_t	T_{t+1}	x_t	y_t	P_t
0	0	0	0	0	0	0
0	1	1	0	0	1	0
1	0	1	0	1	0	0
1	1	0	1	1	1	1

Folosind cunoștințele din paragraful precedent rezultă că:

$$S_t = x_t \oplus y_t = \bar{x}_t y_t \cup x_t \bar{y}_t$$

$$T_{t+1} = x_t y_t$$

$$P_t = x_t y_t$$

Se poate constata ușor că operațiile de adunare și înmulțire binare se reduc la operații logice asupra argumentelor date.

Presupunînd că potențialul electric de +5 V reprezintă unu, iar cel de 0V reprezintă zero, se poate construi un circuit electronic, cu două borne de intrare (x_t, y_t) și două borne de ieșire (S_t, T_{t+1}), capabil să implementeze operația de adunare a două cifre binare, materializate prin niveluri de potențial. Un asemenea circuit sumator se poate sintetiza cu ajutorul circuitelor electronice, care implementează diferite funcții logice.

În practică au fost realizate, în diverse tehnologii (relee electromagnetice, elemente feromagnetice, elemente cu fluid, elemente optice, tuburi electronice, dispozitive semiconductoare discrete, circuite integrate) circuite care pot implementa funcțiile logice prezentate în tabelele 4.2 și 4.3

Elementele constructive ale calculatoarelor electronice sînt realizate cu ajutorul circuitelor logice integrate.

Analiza și sinteza șchemelor care intră în componența calculatoarelor electronice, se pot efectua folosind aparatul logicii matematice. În această privință este concludentă stabilirea ecuațiilor logice, care descriu funcționarea sumatorului elementar.

Circuitele logice realizate sub formă integrată au practic una sau mai multe intrări și pot implementa diferite funcții logice.

Variabilele de intrare (x, y) și ieșire (z) vor fi materializate prin niveluri de tensiune discrete: 0V și +5V, cărora li se asociază valorile logice „0” și respectiv „1”.

Deoarece aceste circuite nu posedă memorie, ieșirile lor pot fi interpretate ca rezultate ale unor operații logice, atîta timp cît sînt prezente (active) semnalele de intrare. Asemenea circuite poartă numele de circuite combinaționale.

În tabelul 4.4 se prezintă câteva circuite logice integrate realizate în tehnologia tranzistor-tranzistor (TTL)*.

Tabela 4.4

EXEMPLPE DE CIRCUITE LOGICE INTEGRATE – TTL

Circuit logic	Nr. de circuite pe pastilă	Simbol pentru elementul de bază
SI (AND) 7408	4/2	$\begin{matrix} x \\ y \end{matrix} \text{---} \text{D} \text{---} z = x \cdot y$
SAU (OR) 7432	4/2	$\begin{matrix} x \\ y \end{matrix} \text{---} \text{D} \text{---} z = x \cup y$
ȘI – NU (NAND) 7400	4/2	$\begin{matrix} x \\ y \end{matrix} \text{---} \text{D} \text{---} z = \overline{xy}$
SAU – NU (NOR) 7402	4/2	$\begin{matrix} x \\ y \end{matrix} \text{---} \text{D} \text{---} z = \overline{x \cup y}$
NU (NOT) 7404	6/1	$x \text{---} \text{D} \text{---} z = \overline{x}$
SAU – EXCLUSIV (SUMA MODULO 2) (XOR) 7486	4/2	$\begin{matrix} x \\ y \end{matrix} \text{---} \text{D} \text{---} z = x \oplus y$

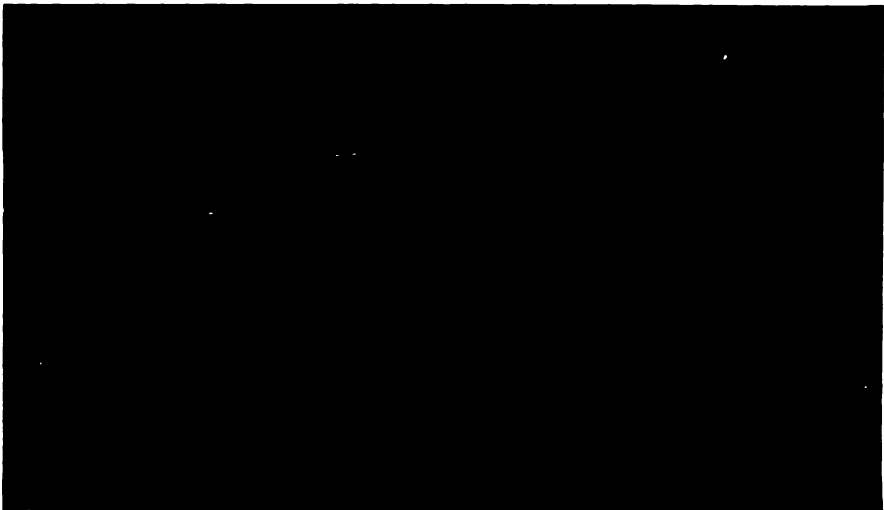


Fig. 4.1. Sumatorul cu două intrări: a) implementare cu circuitele 7486 și 7408 ; b) implementare cu circuitele 7404, 7408, 7432 ; c) schema bloc

* Există mai multe tehnologii de realizare a circuitelor logice integrate: TTL (Logica Tranzistor-Tranzistor), MOS (Metal-Oxid-Semiconductor) în variantele: PMOS (MOS canal P), NMOS (MOS canal N), CMOS (MOS complementar) etc.

II. CALCULATOARE NUMERICE

Cu ajutorul unor asemenea circuite se pot implementa practic ușor diferite funcții logice. De exemplu, plecând de la ecuațiile sumatorului se pot obține implementările din figurile 4.1. a și b, care sub forma generală se pot reprezenta ca în figura 4.1 c.

Se constată că implementarea din figura 4.1. a este mai simplă, însă circuitul din figura 4.1 b poate furniza în punctele 1—4 rezultatele $\bar{x}_i, \bar{y}_i, \bar{x}_i y_i$ și $x_i \cdot \bar{y}_i$, care pot fi utilizate în diverse scopuri, în cadrul schemelor de prelucrare a informației.

Pentru realizarea efectivă a adunării a doi biți, trebuie să se țină seama, pe lângă intrările de date x_i, y_i , și de intrarea de transport T_i , din etajul precedent, conform tabelii de adevăr pentru sumatorul complet.

T_i	x_i	y_i	S_i	T_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Prin examinarea tabelii de mai sus, se constată că un sumator complet cu trei intrări (x_i, y_i, T_i) se poate obține cu ajutorul a două sumatoare simple S și a unui circuit SAU (Fig. 4.2. a) Aceasta se realizează adunând mai întâi x_i

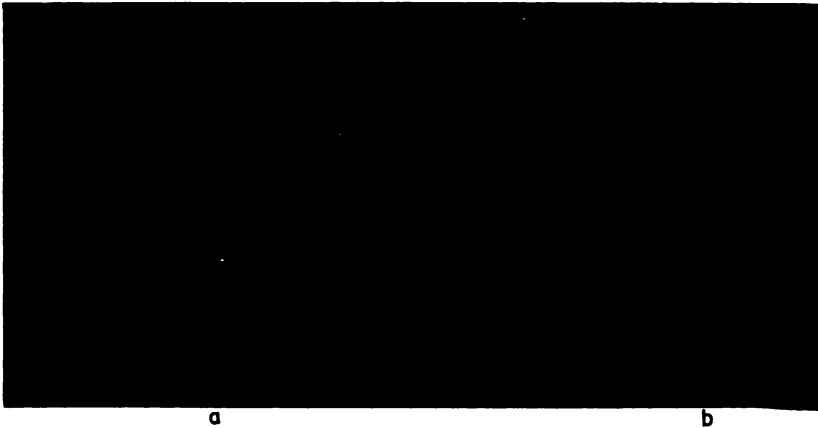
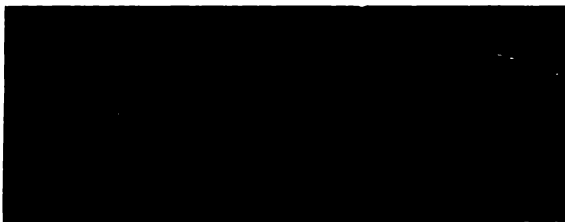


Fig. 4.2. Sumatorul cu trei intrări: a) sumator cu trei intrări realizat cu două sumatoare cu două intrări; b) schema bloc

și y_i pentru a obține rezultatele intermediare S'_i și T'_{i+1} , după care S'_i se va aduna cu T_i , obținându-se suma corectă S_i și transportul T'_{i+1} . Acesta din urmă va fi adunat logic cu T'_{i+1} , pentru a se obține transportul real T_{i+1} . Prin legarea în cascadă a mai multor sumatoare complete, se obțin sumatoare pentru numere binare de mai mulți biți. În figura 4.3 se prezintă un sumator paralel, pentru



numere binare de 8 biți. Prin completarea schemei sumatorului cu circuite logice suplimentare, asupra operanzilor binari $X_{7:0}$ și $Y_{7:0}$ se pot efectua și alte operații aritmetice sau logice.

Fig. 4.3. Sumatorul paralel pentru numere binare de 8 biți

Selecția operațiilor se va face cu un număr binar, cu mai multe ranguri, care se va aplica la un grup de intrări de comandă, înainte de forțarea operanzilor în sumator.

În acest mod se obține o Unitate Aritmetică Logică (UAL).

În figura 4.4 se prezintă o UAL pentru numere binare de 4 biți, asupra căreia se pot efectua diferite operații aritmetice/logice ale căror coduri de operație se aplică la intrările de comandă. La ieșire se obțin, pe lângă rezultatul



Fig. 4.4. Unitate aritmetică-logică pentru numere binare de 4 biți

$Z_{3:0}$ și elemente (indicatorii de condiții) ce caracterizează rezultatul: $Z < 0$, $Z > 0$, $Z = 0$, valoarea transportului T (0 sau 1) etc.

Circuitele de tip UAL joacă un rol important în realizarea unităților de execuție ale calculatoarelor electronice.

Între circuitele logice combinaționale implementate sub formă de circuite integrate se mai pot menționa: decodificatorul (distribuitoarea) și multiplexorul (selectorul).

Decodificatorul are n intrări de date și 2^n ieșiri de date. Pentru fiecare combinație de cifre binare aplicate la intrare, numai una din bornele de ieșire va fi activă.

Decodificatorul 7442 (fig. 4.5) este un decodificator special binar-zecimal. El dispune de 4 intrări, notate cu D, C, B, A, și 10 ieșiri, notate cu Y_9, Y_8, \dots, Y_0 . Pentru fiecare din numerele binare 000—1001, aplicate la intrare, numai câte una din ieșirile Y_0, \dots, Y_9 va fi activă, pe nivel de tensiune coborât (fapt specificat prin ieșirea marcată cu un cerc), celelalte fiind la niveluri ridicate de tensiune.

II. CALCULATOARE NUMERICE

Fig. 45. Decodificator.
binar-zecimal

Fig. 4.6. Multiplexor 2 : 1

Tabela 4.5

TABELA DE LUCRU PENTRU
DECODIFICATORUL 7442

D	C	B	A	Y													
				0	1	2	3	4	5	6	7	8	9				
0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	0	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1
0	0	1	0	1	1	0	1	1	1	1	1	1	1	1	1	1	1
0	0	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1
0	1	0	0	1	1	1	1	0	1	1	1	1	1	1	1	1	1
0	1	0	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1
0	1	1	0	1	1	1	1	1	1	0	1	1	1	1	1	1	1
0	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1
1	0	0	0	1	1	1	1	1	1	1	1	1	1	0	1	1	1
1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1
.
1	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
.
.
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

schema va efectua funcția de demultiplexor pentru intrarea de date C. Astfel, ieșirile Y_0, \dots, Y_3 vor avea următoarele expresii:

$$\begin{aligned}
 Y_0 &= C \cdot (\bar{B}\bar{A}) \\
 Y_1 &= C \cdot (\bar{B}A) \\
 Y_2 &= C \cdot (B\bar{A}) \\
 Y_3 &= C \cdot (BA)
 \end{aligned}$$

Multiplexorul este un circuit cu mai multe intrări de date și cu o singură ieșire de date. Cu ajutorul altor intrări, de selecție, se poate alege intrarea a cărei valoare se va atribui ieșirii.

În figura 4.6 se prezintă multiplexorul 74157, cu două intrări de date A, B, o ieșire de date Y, o intrare de comandă STROBE (activă pe nivelul coborât) și o intrare de selecție SELECT.

Dacă intrarea D este tratată ca intrare de comandă, activă pe nivel coborât, decodificatorul 7442 poate fi considerat ca având trei intrări de date (C, B, A) și opt ieșiri de date (Y_0, \dots, Y_1, Y_7) operând numai în zona marcată cu linie întreruptă în tabela de adevăr 4.5. Cu condiția $D=0$, ieșirile Y_0, Y_1, \dots, Y_7 vor fi date de relațiile:

$$\begin{aligned}
 Y_0 &= CUBUA \\
 Y_1 &= CUBU\bar{A} \\
 &\dots\dots\dots \\
 Y_7 &= \bar{C}\bar{U}\bar{B}\bar{U}\bar{A}
 \end{aligned}$$

Dacă intrarea D, de comandă, este activă pe nivel coborât și dacă intrările A și B sînt folosite ca intrări de selecție a uneia din ieșirile Y_0, Y_1, Y_2, Y_3 ,

Pe un circuit integrat sînt plasate patru asemenea multiplexoare.
 Dacă $STROBE=0$, (activ), stabilirea lui $SELECT=0$, va face $Y=A$, indiferent de B , în timp ce $SELECT=0$, va conduce la $Y=B$, indiferent de A . Compartimentele marcate cu* specifică condiția indiferentă (1 sau 0).

Pe scurt, dacă $STROBE=0$, rezultă:

$$Y = A \cdot \overline{SELECT} \cup B \cdot SELECT.$$

unde $SELECT$ este considerată variabilă logică (Tab. 4.6).

Tabela 4.6

TABELA DE LUCRU A MULTIPLEXORULUI 74157

intrări		ieșire		
STROBE	SELECT	A,	B	Y
1	*	*	*	0
0	0	0	*	0
0	0	1	*	1
0	1	*	0	0
0	1	*	1	1

4.4.2. Circuite logice secvențiale.

În funcționarea reală a circuitelor combinaționale timpul poate interveni în sensul întârzierii apariției semnalelor de ieșire față de momentul aplicării semnalelor de intrare. Aceasta se datorează valorii finite a timpului de propagare a semnalelor, prin circuitele logice din care este construită schema dată.

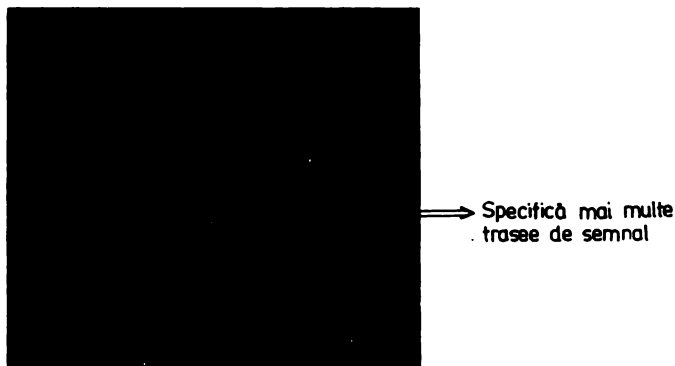


Fig. 4.7. Modelul general al unui circuit secvențial.

Completînd, în mod corespunzător, o schemă constituită din circuite combinaționale, cu elemente capabile să memoreze informația, se pot obține circuite noi, numite secvențiale.

În circuitele secvențiale, ieșirile la un moment dat nu depind numai de intrările aplicate la acel moment, ci și de intrările aplicate la momentele anterioare.

Circuitele secvențiale la care aplicarea semnalelor la intrare și modificarea semnalelor de ieșire se realizează sub comanda unui semnal de sincronizare (tact), poartă numele de circuite secvențiale sincrone. Acestea sînt frecvent folosite în calculatoare.

Modelul general al unui circuit secvențial sincron este dat în figura 4.7, în care:

X reprezintă intrările curențe de date.

Y constituie intrări ce conțin informații referitoare la datele care s-au aplicat la momentele anterioare,

Z este ieșirea curență,

Y' reprezintă informația care va fi furnizată la intrarea Y a circuitului, în ciclul următor de lucru; ea este generată pe baza „istoriei” circuitului și a informației curențe de la intrarea X.

Se constată că circuitul trebuie să funcționeze secvențial, sub controlul unui semnal de sincronizare T , asociat cu timpul.

Informația referitoare la timp (T) este furnizată sub forma discretă — impulsuri, care apar la momente bine definite în timp, purtînd numele de semnale de ceas sau simplu „ceas”.

Un asemenea circuit secvențial este un circuit sincron.

Elementele de memorie poartă numele de circuite bistabile. Ele pot fi realizate fizic sub forme diferite, utilizînd elemente logice SI-NU, SAU-NU, în conjuncție cu inversoare etc. În figura 4.8 se prezintă un bistabil realizat cu circuite ȘI-NU (NAND) și tabela corespunzătoare de adevăr.



Fig. 4.8. Bistabil realizat cu circuite SI-NU (NAND) și tabela de adevăr

El posedă intrările S , R și ieșirile Q și \bar{Q} . Q^n reprezintă ieșirea curență la momentul (pasul) n , al secvenței, iar Q^{n+1} este ieșirea la pasul următor, al secvenței.

Funcționarea lui se poate urmări considerînd că, inițial, sînt îndeplinite condițiile:

$$Q=0; S=1; R=1,$$

care reprezintă o stare stabilă, caracterizată prin ieșirea $Q=0$. Dacă $S=0$ și $R=1$, rezultă $Q=1$. În continuare, dacă $S=1$ și $R=1$, Q se va menține egal cu 1.

Dacă $S=1$ și $R=0$, rezultă $Q=0$. În continuare, dacă $S=1$ și $R=1$, Q se va menține egal cu 0.

Nu este permis ca ambele intrări \bar{S} și \bar{R} să fie zero simultan, deoarece ieșirile Q și \bar{Q} vor fi forțate la nivel ridicat (unu), în timp ce ele, conform definiției, trebuie să aibe valori complementare.

Se constată ușor că ecuația de funcționare a circuitului va fi:

$$Q^{n+1} = \bar{S} \cup \bar{R} \cdot Q^n$$

Ieșirea Q^{n+1} la pasul următor al secvenței este funcție de valorile curente ale semnalelor S , \bar{R} și de valoarea Q^n , memorată la pasul curent.

Pentru a asigura controlul prin semnalul de ceas, în vederea stabilirii cu precizie a momentului memorării informației, se va folosi elementul bistabil din figura 4.9. Acesta posedă o intrare de date, la care se aplică informația binară ce trebuie memorată la forțarea semnalului de ceas. Memorarea se efectuează când semnalul de ceas se afla pe nivel ridicat (valoarea logică unu).

Ecuația de funcționare a circuitului bistabil (de tip D) este următoarea:

$$Q^{n+1} = C \cdot D \cup \bar{C} \cdot Q^n$$

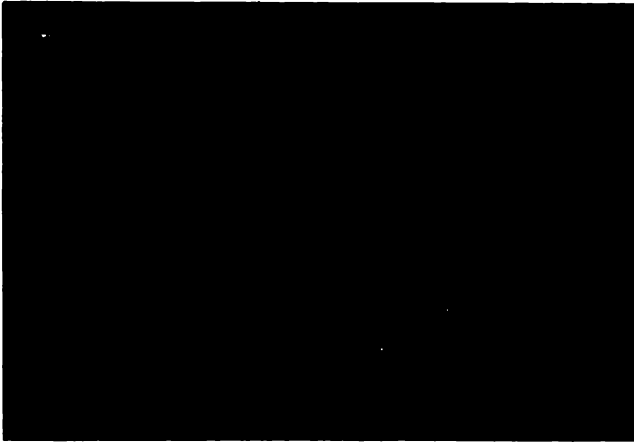


Fig. 4.9. Bistabil de tip D: a) schema logică; b) simbol

Când semnalul de ceas este la nivel coborât (0), $Q^{n+1} = Q^n$, indiferent de intrarea D ; dacă semnalul de ceas este la nivel ridicat (1), $Q = D$, indiferent de vechea valoare Q .

Registre și numărătoare-aplicații.

Registre. Folosind asamblaje de elemente bistabile (de exemplu de tip D) se pot implementa scheme de memorare a informației binare, sub formă de cuvinte cu mai multe ranguri.

În figura 4.10a se prezintă un registru notat cu A , cu patru ranguri, cu intrările de date $X_{3:0}$, intrarea de ceas C și ieșirile $A_{3:0}$, iar în figura 4.10b se prezintă notația simplificată.

Pentru a specifica procesul de încărcare a cuvântului binar $X_{3:0}$ în registrul $A_{3:0}$, la momentul aplicării semnalului de ceas C , se poate folosi notația:

$$C \cdot A \leftarrow X;$$

* Există și alte tipuri de bistabile: J K, T, RS etc.



Fig. 4.10. Registrul cu patru ranguri, cu încărcare paralelă:
a) schema extinsă; b) simbol

Registrul A poate avea mai multe surse de informații. De exemplu, acestea pot fi alte registre sau ieșiri ale unor scheme combinaționale, notate cu $X_{3:0}$, $Y_{3:0}$, $Z_{3:0}$. Dacă se urmărește încărcarea succesivă a cuvintelor X , Y , Z în registrul A , la semnalele de ceas C_t , C_{t+1} , C_{t+2} , se poate imagina schema de principiu, din figura 4.11.

Operațiile se pot descrie sub forma următoare:

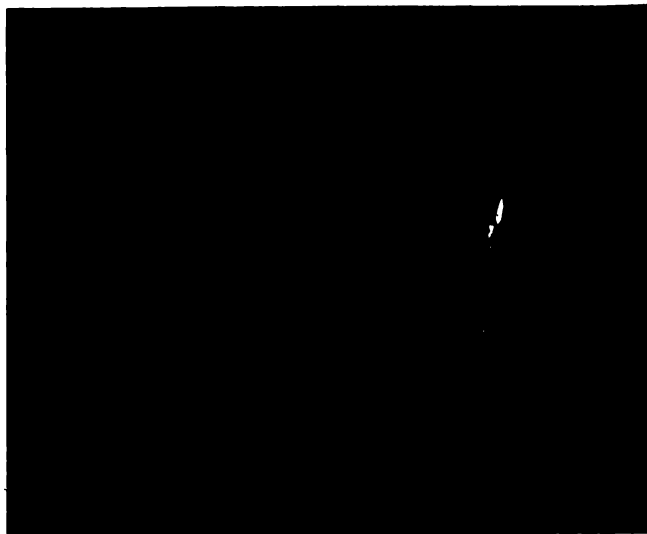


Fig. 4.11. Încărcarea unui registru de la mai multe surse de informații

O *Unitate Aritmetică Logică (UAL)* poate fi completată cu două registre de intrare (A , B), pentru operanzi și cu un registru de ieșire (R), pentru rezultat.

Se considera că:

- la semnalul de cesa C_1 se încarcă operanzii în registrele A și B
- la semnalul C_2 are loc operația de adunare prin activarea unui cod corespunzător de operație la UAL , încărcarea registrului R , cu rezultatul obținut și a registrului CND , cu indicatorii de condiții.

În figura 4.12 este prezentată schemă de principiu a unei unități simple de execuție. Astfel, s-a implementat o schemă simplă de coduri de operații, cu ajutorul cărora se pot efectua operațiile de: adunare, înmulțire logică și sumă logică asupra a doi operanzi, reprezentați prin cuvinte binare de 4 biți.

4. LOGICĂ ȘI CIRCUITE LOGICE

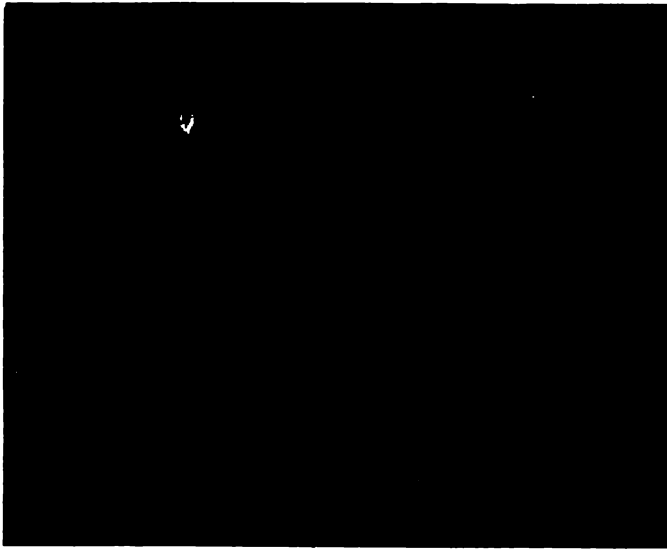


Fig. 4.12. Schema de principiu a unei unități de execuție

Se constată necesitatea existenței unei unități de comandă, care trebuie să distribuie semnalele de ceas și codurile de operație, pentru UAL, într-o secvență dată, în funcție de natura operațiilor solicitate:

- $C_1. A \leftarrow \text{operand 1}; B \leftarrow \text{operand 2}$
- $C_2. R \leftarrow A + B; CND \leftarrow \text{indicatori}$
- $C_3. B \leftarrow R$

Numărătoarele reprezintă circuite secvențiale constituite din bistabile interconectate prin circuite logice. În cazul general, un numărător primește la intrarea sa o serie de impulsuri. Ieșirile bistabililor, care constituie numărătorul, vor reflecta, sub forma unui număr binar, conținutul numărătorului, egal cu numărul impulsurilor aplicate la intrare, pînă în acel moment.



Fig. 4.13. Terminalele numărătorului 74193

Un numărător cu 4 ranguri numără maximum 16 impulsuri, înainte de a reveni în starea inițială. Dacă, inițial, conținutul său era 0000, el va număra în secvența: 0001, 0010, 0011, ..., 1110, 1111, 0000. Se spune că acest numărător operează „modulo 16“.

Pentru a utiliza un numărător este necesar ca el să mai posede și o intrare de comandă care, la primirea unui semnal, va aduce în starea inițială (zero) toți bistabilii.

Numărătoarele realizate sub formă de circuite integrate pot dispune de o serie de facilități, activate prin operații de comandă corespunzătoare:

- anulare conținut (*CLEAR*),
- încărcare (*LOAD*),
- numărare în sens crescător-incrementare (*COUNT-UP*),
- numărare în sens descrescător-decrementare (*COUNT-DOWN*)

Astfel, numărătorul 74193, pe 4 biți, (fig. 4.13) are un caracter reversibil, necesitând, pe lângă semnalele de comandă amintite mai sus și intrările paralele *A*, *B*, *C*, *D*; ieșirile paralele *Q_A*, *Q_B*, *Q_C*, *Q_D*; ieșirea de împrumut (*BORROW*) și ieșirea de transport (*CARRY*).

Numărătoarele, împreună cu registrele, precum și cu alte structuri logice secvențiale constituie resursele fizice (hardware) ale calculatoarelor numerice.

Din cele prezentate mai sus, se pot observa următoarele:

- cuvintele binare (operanzii) sînt stocați în registre, al căror număr de bistabili este egal cu numărul de ranguri ale cuvîntului binar;
- registrele pot fi, atît surse, cît și destinații de operanzi;
- informația se prelucrează în timpul transferului său, de la registrele sursă, spre un registru destinație, prin intermediul unor scheme logice combinaționale (de exemplu: UAL), plasate între surse și destinații;
- la un moment dat un registru poate fi destinație numai pentru un singur transfer;
- pentru implementarea unei secvențe de transferuri între resursele (registre, numărătoare, rețele logice combinaționale, trasee de legătură) unui sistem de prelucrare a informației (unitate de execuție), sînt necesare semnale de comandă, furnizate de o schemă specială numită unitate de comandă, care înglobează și „ceasul“ sistemului.

Partea a III-a

CALCULATORUL PERSONAL HC-85, STRUCTURĂ, COMPONENTE, OPERARE, PROGRAMARE

Capitolul 5 | **Calculatoare numerice.** **Structură și mod de operare.**

5.1. Conceptul de calculator

Un calculator numeric este construit dintr-un ansamblu de resurse fizice (hardware) și de programe de sistem (software de bază), care asigură prelucrarea informațiilor în conformitate cu algoritmi specificați de utilizator, prin programele de aplicații (software de aplicații).

În contextul dat, un algoritm reprezintă o colecție de reguli, o secvență de acțiuni elementare, privind efectuarea unor operații cu caracter aritmetic-logic, asupra unor date, pentru a produce alte date, reprezentând rezultatele — datele de ieșire. Un algoritm trebuie să aibă următoarele caracteristici:

- să fie finit, să se termine după un număr finit de pași;

- să fie determinist, fără ambiguități — aplicarea lui repetată asupra acelorași date de intrare, să conducă la același rezultat;

- să aibă o intrare — un set inițial de date;



Fig. 5.1. Schema execuției unui algoritm de către un operator uman

- să aibă o ieșire — rezultatele;
- să fie eficient, în sensul că operațiile implicate să se execute exact și într-un interval finit de timp.

Un algoritm de calcul poate fi executat manual, de către un operator uman, care dispune de un formular de calcul ce conține datele și metoda de calcul (sub forma unor instrucțiuni), un creion etc. (fig. 5.1).

Operatorul va interpreta instrucțiunile care descriu metoda de calcul (programul) în termenii:

- operațiilor aritmetice care pot fi efectuate cu calculatorul de buzunar;
- operațiilor de decizie, privind execuția instrucțiunii următoare din program, în funcție de rezultatele curente;
- operațiilor de citire a unor date și scriere a unor rezultate în compartimentele corespunzătoare ale formularului de calcul.

Se poate observa că, sub forma cea mai generală, formularul de calcul joacă rolul unei memorii, care dispune de:

- o zonă formată din compartimente ce conțin instrucțiunile, care descriu metoda de calcul; compartimentele vor fi, în general parcurse secvențial;
- o zonă ce conține compartimente în care sînt plasate sau se introduc datele de intrare, rezultatele intermediare și rezultatele finale.

Compartimentele în care sînt plasate instrucțiunile vor fi numerotate (numerele respective vor fi tratate drept adrese de instrucțiuni). Instrucțiunile vor fi executate în ordinea crescătoare a adreselor, atîta timp cît instrucțiunea curentă nu impune altă ordine de parcurgere a secvenței de instrucțiuni — a programului.

Compartimentele în care se introduc sau în care se află deja datele de intrare și compartimentele în care se vor plasa rezultatele finale vor fi numerotate (în continuarea compartimentelor programului), vor avea adrese.

Metoda de calcul va fi descrisă prin instrucțiuni de forma prezentată în exemplul următor:

```

i.   citește data de la adresa j
i+1  adună data de la adresa j+1
i+2  memorează rezultatul la adresa j+2
i+3  dacă rezultatul este mai mare decît zero, execută instrucțiunea
      de la adresa i+7
i+4  stop
      .....
i+7  execută instrucțiunea i
      .....
j    data 1
j+1  data 2
j+2  data 1+data 2 (după execuția programului)

```

Analizînd acest program se pot constata următoarele:

- instrucțiunile și datele sînt plasate în compartimentele aceluiași mediu ce joacă rol de memorie;
- instrucțiunile și datele sînt apelate prin adrese;
- o instrucțiune conține un cîmp care specifică operația de efectuat (codul de operație) și un cîmp ce specifică adresa operandului (adresa);

— codurile de operație se referă la: citirea/stocarea datelor în memorie, efectuarea unor operații aritmetice (adunare, scădere, înmulțire, împărțire etc.), la modificarea secvenței de parcurgere a instrucțiunilor programului, în funcție de anumite condiții sau necondiționat (instrucțiunile de la adresele $i+3$ și $i+7$).

5.2. Structura și operarea calculatoarelor numerice

Un algoritm poate fi executat în mod automat cu ajutorul unui calculator. În acest scop (conform principiilor stabilite de J. von Neumann) un calculator trebuie să posedे următoarele elemente:

— un mediu de intrare (*unitate de intrare — UI*) pentru instrucțiuni și date (operanzi);

— un mediu de ieșire (*unitate de ieșire — UE*) pentru extragerea rezultatelor și prezentarea acestora într-o formă accesibilă utilizatorului;

— o memorie (*unitatea de memorie — UM*) în care se pot stoca programul, datele inițiale, rezultatele parțiale și finale;

— un ansamblu de prelucrare (*unitatea aritmetică — logică sau de execuție — UAL*), capabil să efectueze operații aritmetice și logice, în conformitate cu un algoritm dat, specificat prin program;

— un element de decizie și comandă (*unitatea de comandă — UC*), care, pe baza rezultatelor parțiale obținute, poate selecta una din opțiunile posibile de continuare a calculelor și asigură controlul întregului ansamblu de unități, în vederea funcționării lui automate.

În figura 5.2 se prezintă schema funcțională a unui calculator, constituită din unitățile specificate anterior.

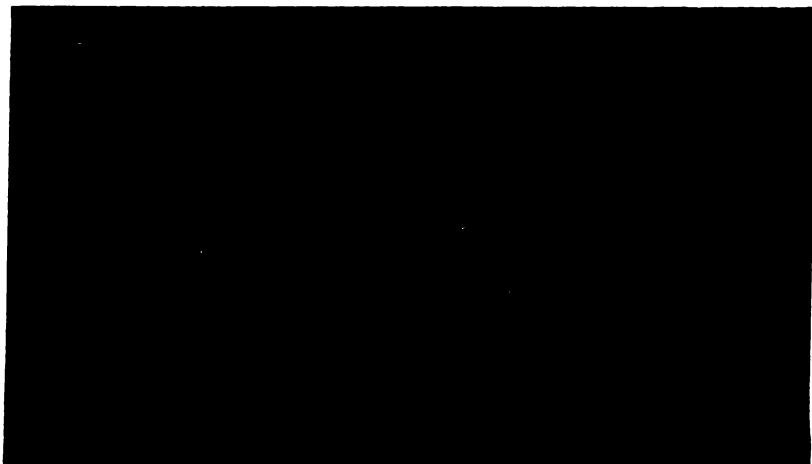


Fig. 5.2. Schema funcțională a unui calculator

Unitatea de intrare preia, sub controlul unității de comandă, informația (instrucțiuni/date) de la o serie de echipamente periferice de intrare (cititor de cartele perforate, cititor de bandă perforată, tastatura unei console operator — display), echipamente de stocare externă a informației pe medii magnetice (discuri, benzi, casete, discuri flexibile) și o aduce la o formă standard de reprezentare, din punct de vedere electric, transferind-o, în continuare, în unitatea de memorie.

Echipamentele periferice se conectează la unitatea de intrare prin așa-numita interfață standard.

Unitatea de ieșire preia, sub controlul unității de comandă, informația corespunzătoare din memorie și o transferă unor echipamente periferice de ieșire: consolă — display, perforator de bandă, imprimată, trasator de curbe (plotter) sau unor echipamente de stocare externă a informației pe medii magnetice: discuri, benzi, casete, discuri flexibile etc.

În cele mai multe cazuri unitățile de intrare/ieșire sînt prezentate împreună, sub forma unității (subsistemului) de intrare/ieșire (I/E) care, la unele calculatoare, mai poartă numele de *Unitate de schimburi* (Felix C-265) sau *Canal* (IBM 360/370). În figura 5.3 se prezintă, sub formă de schemă bloc unitatea de I/E.

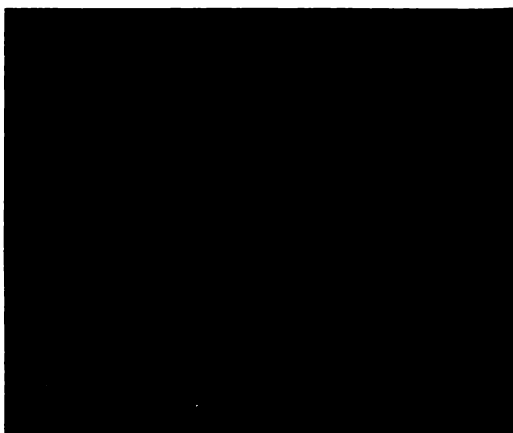


Fig. 5.3. Unitatea de intrare/ieșire

Unitatea de memorie are drept rol stocarea programelor și a datelor. În prezent memoria se realizează cu ajutorul circuitelor electronice integrate pe scară largă, constituite din structuri elementare de memorare a informației (reprezentată sub forma unor niveluri de potențial electric), împreună cu circuitele de acces și comandă.

Elementele de memorare (circuiturile bistabile, în cazul memoriilor statice sau capacitățile de grilă ale unor inversoare, în cazul memoriilor dinamice) sînt organizate sub forma unor registre

capabile să memoreze cîte un cuvînt binar. Fiecare element de memorare stochează un bit al unui cuvînt dat.

Din punct de vedere funcțional, memoria poate fi examinată pe baza modelului din figura 5.4, în care M^i reprezintă cuvîntul i , de la adresa i , din memorie.

Se poate observa că, pe baza adresei de n biți, aplicată la intrarea decodificatorului de adrese, una din cele 2^n ieșiri (0, 1, 2, ..., $2^n - 1$) va fi activată, selectînd un cuvînt din memorie. Selecția se referă atît la circuitul $\$I1$, de aplicare a semnalului de ceas, cît și la circuitul $\$I2$, folosit pentru citirea conținutului registrului respectiv

5. CALCULATOARE. STRUCTURĂ. OPERARE



Fig. 5.4. Modelul funcțional al unei memorii interne

Operația de înscriere a unui cuvânt în memorie, aplicat la intrarea de date, se efectuează în următoarele etape:

1. la intrarea de adrese se aplică informația reprezentând adresa, la intrarea de date se aplica informația care reprezintă datele;
2. se mențin informațiile de la punctul 1 și se mai aplică semnalul de comandă — scriere;

Este evident că informația, care va fi prezentă la intrarea tuturor registrelor, va fi înscrisă numai în registrul selectat de ieșirea decodicatorului și activat de semnalul de scriere.

Operația de citire se va desfășura în următoarele etape:

1. la intrarea de adrese se aplică informația reprezentând adresa;
2. se menține informația de la punctul 1 și se aplică semnalul de comandă-citire.

Pe baza adresei decodificate și a semnalului de citire, se va activa circuitul $\Phi 12$, de la ieșirea registrului selectat. Acesta va permite transferarea conținutului registrului respectiv, la intrarea circuitului SAU, care îl va transmite mai departe, la ieșirea sa.

În cazul examinat, numărul de biți ai adresei este n , iar cel al datelor este m . Memoria va avea o capacitate de 2^n cuvinte binare, de câte m biți fiecare.

În mod curent capacitatea unei memorii se măsoară în Kcuvinte ($1 K = 2^{10} = 1024$). Microcalculatorul HC-85 are o memorie cu o capacitate totală de

64 Koceteți — cuvinte de 8 biți, în timp ce calculatorul FELIX C-256 are o memorie cu capacitatea de 256 Ko.

O altă caracteristică a memoriei o reprezintă timpul de acces, care oferă o imagine asupra vitezei de lucru (citire sau scriere). De exemplu, o memorie cu timpul de acces de 450 ns permite scrierea sau citirea a $2 \cdot 10^6$ cuvinte/s.

Pentru aprofundarea modului de funcționare a unui calculator, se va presupune că memoria M va avea un registru RA pentru stocarea adreselor și un registru RD, pentru intrări/ieșiri de date, conform modelului din figura 5.5.

În aceste condiții funcționarea memoriei poate fi descrisă astfel:
scriere:

1. $RA \leftarrow \text{adresa}; RD \leftarrow \text{data};$
2. $M^{RA} \leftarrow RD;$

citire:

1. $RA \leftarrow \text{adresa};$
2. $RD \leftarrow M^{RA}.$

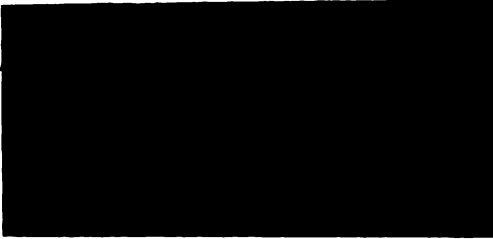


Fig. 5.5. Modelul memoriei interne cu registrele de adresă (RA) și de date (RD)

Notăția M^{RA} trebuie interpretată în sensul că simbolul $\perp RA$ specifică conversia numărului binar — adresa — din RA, într-un număr zecimal i, care reprezintă numărul cuvântului selectat, din memorie, fie în vederea citirii, fie în vederea scrierii.

Unitatea de execuție (Unitatea Aritmetică — Logică) se caracterizează prin aceea că poate efectua operații aritmetice/logice asupra operanzilor aplicați la intrare, în conformitate cu o comandă, un cod de operație, furnizat din exterior.

Unitatea de execuție va avea ca ieșiri:

- rezultatul operației,
- indicatorii de condiții (semnul rezultatului, rezultatul egal cu zero, paritatea rezultatului, transportul în afara rangului de semn etc), indicatorii de eroare (depășirea capacității de reprezentare a numerelor în calculator, de către rezultat).

Sub forma generală o unitate de execuție se poate reprezenta ca în figura 5.6 unde RI1 și RI2 sînt registrele în care se plasează operanzii de prelucrat, iar RE este registrul de ieșire al rezultatului din unitate.

Sub forma cea mai simplă operarea unității de execuție se poate descrie astfel:

$$j. RE \leftarrow RI1 @ RI2; RIND \leftarrow \text{indicatori};$$

unde @ reprezintă una din operațiile pe care le poate efectua unitatea, iar j este semnalul de tact la care are loc operația.

O unitate de execuție modernă poate dispune de mai multe registre generale sau specializate, în care se pot afla operanzi, adrese etc.

Sub forma cea mai simplă o unitate de execuție se poate considera că este constituită dintr-un singur registru (numit și acumulator) și o unitate aritmetică — logică. (fig. 5.7). În această situație unul din operanzi se va afla în acumulator (eventual încărcat în prealabil din memorie sau ca rezultat al unei operații anterioare).

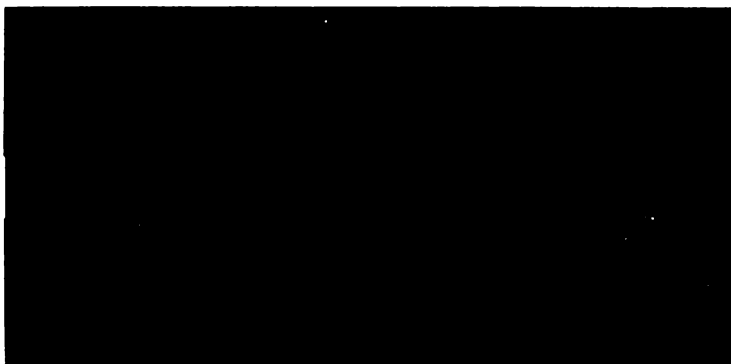


Fig. 5.7. Schema bloc a unității aritmetice-logice cu un singur acumulator

Pentru a aduna două cuvinte aflate în memorie, cu plasarea rezultatului în acumulator (AC), se va încărcă, mai întâi, AC cu primul operand, după care, AC se va aduna cu cel de-al doilea operand, citit din memorie și stocat temporar în registrul de date al memoriei RD , rezultatul fiind în cele din urmă încărcat în AC :

— formatul de reprezentare al operanzilor: virgulă fixă, virgulă mobilă, binar-zecimal;

$$j. AC \leftarrow AC + RD;$$

O unitate de execuție se caracterizează prin următoarele elemente:

— lungimea în biți a operanzilor manipulați;

— viteza de operare.

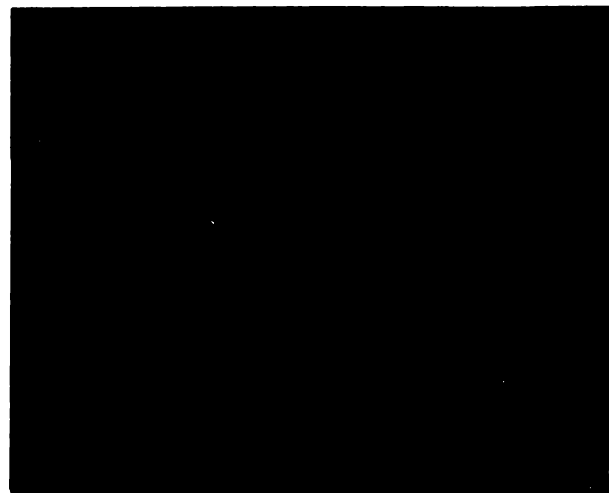


Fig. 5.8. Schema bloc a unității de comandă

Unitatea de comandă asigură citirea instrucțiunilor programului din memorie și execuția lor automată. Ea coordonează, prin semnale de comandă, funcționarea tuturor celorlalte unități ale calculatorului.

În principiu, unitatea de comandă (*UC*) este construită dintr-un registru de instrucțiuni *RI*, un registru contor program *CP* și logica de comandă aferentă (fig. 5.8).

Registrul instrucțiunii (RI) păstrează instrucțiunea curentă, citită din memorie, pe toată durata execuției sale. Instrucțiunea va specifica, de regulă, un cod de operație și una sau mai multe adrese de operanzi.

Contorul programului (CP) va conține adresa instrucțiunii următoare, din program. Conținutul său se poate încărca direct (în cazul unor operații ce modifică secvența de parcurgere a programului: transferuri condiționate sau necondiționate) sau poate fi incrementat (în cazul parcurgerii secvențiale a programului). Logica aferentă *UC* testează indicatorii de condiții și asigură modificarea corespunzătoare a *CP*.

Pe baza codului de operație *UC* furnizează semnalele de comandă pentru controlul *UI/E*, *UM*, *UE*, pe durata fiecărei instrucțiuni, în sincronism cu semnalul furnizat de un oscilator, care joacă rolul de ceas, în sistem.

Secvența de operații, referitoare la citirea unei instrucțiuni din memorie, este următoarea:

1. $RA \leftarrow CP$;
2. $RD \leftarrow M^{RA}$;
3. $RI \leftarrow RD$;

Contorul programului se incrementează la sfârșitul fazei de execuție a instrucțiunii curente, în cazul parcurgerii secvențiale a programului.

Ansamblul unitate de comandă — unitate de execuție poartă numele de unitate centrală de prelucrare sau procesor.

Procesorul împreună cu unitatea de memorie formează unitatea centrală a calculatorului (fig. 5.9).



Fig. 5.9. Structura unui calculator la nivelul unităților funcționale

5.3. Instrucțiunile calculatoarelor numerice

Instrucțiunile calculatoarelor numerice conțin, după cum s-a observat anterior, specificații referitoare la operația care trebuie efectuată de către o unitate dată, din componența calculatorului (adesea unitatea de execuție) și specificații referitoare la adresa unui operand sau a unei instrucțiuni. În unele cazuri o instrucțiune poate conține mai multe adrese: adresa primului operand, adresa celui de-al doilea operand și, eventual, adresa rezultatului. În cele ce urmează se vor considera instrucțiuni simple, avînd o singură adresă (fig. 5.10).

Cîmpul codului de operație specifică una din funcțiile ce se pot executa de către unitățile calculatorului. Dacă acest cîmp posedă m biți, se pot codifica 2^m instrucțiuni diferite.

Cîmpul de adresă specifică o adresă de operand sau de instrucțiune. În cazul în care cîmpul de adresă conține n biți, se poate explora un spațiu de adrese în memorie de 2^n cuvinte.

Totalitatea instrucțiunilor executabile de către calculatorul numeric, la nivelul la care a fost tratat, formează așa-numitul „limbaj mașină”.

Pentru a fi executate, instrucțiunile trebuie să fie transmise unității de comandă sub forma unor cuvinte binare (cod mașina).



Fig. 5.10. Structura unei instrucțiuni simple la nivel de cîmpuri

În vederea simplificării muncii programatorilor, cîmpurile de cod de operație și adrese se înlocuiesc prin simboluri literale mnemotehnice (mnemonice), care pot fi traduse în coduri mașină în mod automat, cu ajutorul unui program numit asamblor.

Limbajul care conține instrucțiunile reprezentate sub formă de mnemonice poartă numele de limbaj de asamblare.

Limbajele mașina și de asamblare sînt limbaje de nivel coborît, deoarece ele sînt interpretate direct de către calculator, la nivelul examinat, numit — nivelul mașinii convenționale. Cu ajutorul lor se scriu programele de sistem, necesare exploataării eficiente a resurselor fizice ale calculatorului.

În general instrucțiunile calculatoarelor se pot împărți în două clase:

- instrucțiuni de transfer de date, de calcul și de schimb de informații cu echipamentele periferice,
- instrucțiuni de decizie, care au ca efect modificarea ordinii de execuție a instrucțiunilor programului.

Cu ajutorul unor mnemonice convenabile se vor da unele exemple de instrucțiuni din prima clasă:

<i>Cod de operație</i>	<i>Adresă</i>	<i>Comentarii,</i>
<i>INC</i>	<i>ADR1;</i>	<i>O încarcă în acumulator operandul de la adresa ADR1, din memorie:</i>
		$AC \leftarrow M^{ADR1}$
<i>MEM</i>	<i>ADR2;</i>	<i>memorează conținutul acumulatorului, la adresa sa ADR2:</i>
		$M^{ADR2} \leftarrow AC$

<i>ADN</i>	<i>ADR3</i> ;	<i>adună conținutul acumulatorului cu conținutul celulei de memorie de la adresa ADR3 și plasează rezultatul în acumulator:</i>	
			$AC \leftarrow AC + M^{\perp ADR3}$
<i>SCD</i>	<i>ADR4</i> ;	<i>scădere:</i>	$AC \leftarrow AC - M^{\perp ADR4}$
<i>ADL</i>	<i>ADR5</i> ;	<i>adunare logică, bit cu bit:</i>	$AC \leftarrow AC \cup M^{\perp ADR5}$
<i>INL</i>	<i>ADR6</i> ;	<i>înmulțire logică, bit cu bit:</i>	$AC \leftarrow AC \cap M^{\perp ADR6}$
<i>SMD</i>	<i>ADR7</i> ;	<i>suma modulo doi:</i>	$AC \leftarrow AC \oplus M^{\perp ADR7}$

Instrucțiunile care se referă la conținutul acumulatorului, nu necesită specificarea adresei:

<i>NEG</i> ;		<i>conținutul acumulatorului se inversează bit cu bit:</i>	$AC \leftarrow \overline{AC}$
<i>DES</i> ;		<i>conținutul acumulatorului se deplasează spre stînga cu un bit:</i>	$AC_{0:7} \leftarrow AC_{1:7,0}$
<i>DED</i> ;		<i>conținutul acumulatorului se deplasează spre dreapta cu un bit, cu extensia rangului de semn:</i>	$AC_{0:7} \leftarrow AC_0, AC_{0:8}$
<i>RST</i> ;		<i>conținutul acumulatorului se rotește spre stînga cu un rang:</i>	$AC_{0:7} \leftarrow AC_{1:7}, AC_0$
<i>RDR</i> ;		<i>conținutul acumulatorului se rotește spre dreapta cu un rang:</i>	$AC_{0:7} \leftarrow AC_7, AC_{0:8}$

Instrucțiunile de intrare/ieșire vor specifica în partea de adresă numerele (adresele) unor registre asociate interfețelor prin care sînt cuplate la unitățile de intrare/ieșire echipamentele periferice. Aceste registre, care se mai numesc și porturi de intrare/ieșire, sînt utilizate pentru a vehicula trei tipuri de informații cu interfața perifericului în cauză: date, comenzi și stări.

Datele sînt emise/recepționate de unitatea centrală/interfață periferic, pentru operațiile de ieșire și recepționate/emise de unitatea centrală/interfață periferic, în cazul operațiilor de intrare.

Comenzile sînt întotdeauna emise de unitatea centrală (sub forma unor cuvinte de comandă, în care fiecare bit are o anumită semnificație pentru periferic) și sînt interpretate — executate de interfață.

Stările sînt emise de interfața perifericului (specifică condițiile în care se află perifericul) și sînt citite de unitatea centrală în vederea luării unor decizii.

Porturile de intrare și de ieșire se vor considera organizate sub forma a două memorii notate cu *PI* și, respectiv — *PE*.

În aceste condiții se pot descrie instrucțiunile de intrare și de ieșire:

<i>INT</i>	<i>ADR16</i> ;	<i>conținutul portului de intrare cu adresa ADR16 se transferă în acumulator:</i>	$AC \leftarrow PI^{\perp ADR16}$
------------	----------------	---	----------------------------------

IES **ADR8;** *conținutul acumulatorului se transferă în portul de ieșire cu adresa ADR8:*
 $PE \stackrel{ADR8}{\leftarrow} AC$

Instrucțiunile din cea de-a doua clasă testează îndeplinirea unor condiții specificate prin indicatorii de condiții. În cazul îndeplinirii condiției specificate se trece la instrucțiunea a cărei adresă este dată în câmpul de adresă, din instrucțiune, în caz contrar, se trece la instrucțiunea următoare din program (prin incrementarea contorului programului: $CP \leftarrow CP+1$).

În cele ce urmează se prezintă exemple de instrucțiuni de transfer al comenzii în program:

<i>Cod de operație</i>	<i>Adresă</i>	<i>Observații</i>
TPL	$\overline{ADR1}$;	<i>testează condiția de rezultat pozitiv (plus) și se trece, în caz afirmativ, la instrucțiunea cu adresa ADR1; în caz contrar se trece la instrucțiunea cu adresa dată de $CP \leftarrow CP+1$.</i>
TMI	ADR2;	<i>testează condiția de rezultat negativ.</i>
TZE	ADR3;	<i>testează condiția de rezultat zero.</i>
TTR	ADR4;	<i>testează condiția de apariție a transportului.</i>
TNC	ADR5;	<i>transfer necondiționat al execuției programului la instrucțiunea cu adresa ADR5.</i>

Implementarea (în sensul de realizare) unei instrucțiuni se efectuează în două etape: citire și execuție.

Etapa de citire este comună pentru toate tipurile de instrucțiuni, în timp ce etapa de execuție diferă de la instrucțiune la instrucțiune.

Etapa de citire a instrucțiunii constă în extragerea instrucțiunii din memorie, pe baza conținutului contorului programului, și aducerea ei în registrul instrucțiunii:

1. $RA \leftarrow CP$
2. $RD \leftarrow M^{CP}$
3. $RI \leftarrow RD$

În realitate această etapă poate fi mai complexă în sensul efectuării unor operații asupra părții de adresă din *RI*, în conformitate cu indicațiile cuprinse în alte câmpuri ale instrucțiunii (indexare, adunare cu conținutul unui registru bază, adresare indirectă etc).

Faza de execuție pentru instrucțiunile de prelucrare implică: aducerea operandului din memorie în registrul de date *RD*, efectuarea operației specificate cu conținutul acumulatorului și plasarea rezultatului în acumulator):

4. $RA \leftarrow RI [ADRESA]$; *partea de adresă din RI trece în RA*
5. $RD \leftarrow M^{RA}$
6. $AC \leftarrow AC @ RD$
7. $CP \leftarrow CP+1$

Instrucțiunile de test și control al secvenței programului afectează conținutul contorului programului după cum urmează:

5. $CP \leftarrow RI [ADRESA]$, în cazul condiției îndeplinite sau
- 5'. $CP \leftarrow CP+1$, în cazul în care nu se îndeplinește condiția testată.

5.4. Software, programare, algoritmi

După cum s-a arătat anterior, un sistem de calcul constă din echipamentele fizice de introducere/extragere, stocare și prelucrare a informației (hardware), pe de-o parte, și din pachete de programe de sistem și aplicații (software), pe de altă parte.

Sub forma cea mai generală software-ul cuprinde totalitatea programelor și a tehnicilor de programare ale unui calculator.

Programarea are ca scop specificarea unei activități de prelucrare a informației, într-un limbaj adecvat, în vederea efectuării ei automate, de către calculator.

Sarcina unui programator, în sensul cel mai larg, constă în descrierea sub forma unui algoritm a procesului de rezolvare a problemei și transcrierea algoritmului într-un program, cu ajutorul unui limbaj de programare. Adesea etapa elaborării algoritmului de rezolvare a unei probleme revine unui specialist de înaltă calificare, numit analist.

Algoritmul reprezintă după cum s-a mai arătat un ansamblu de reguli sau instrucțiuni, cu următoarele caracteristici:

- este finit și se termină după un număr finit de operații;
- este specificat precis, fără ambiguități;
- dacă necesită date de intrare, ele trebuie specificate din punctul de vedere al tipului de aplicații (numere întregi, reale, caractere alfanumerice etc);
- conduce întotdeauna la un rezultat (date de ieșire);
- este eficient: toate operațiile se efectuează exact, într-un timp finit.

Pentru a rezolva o problemă dată, cu ajutorul unui calculator, se parcurg mai multe etape: analiza problemei, rafinarea algoritmului, programarea, execuția programului, interpretarea rezultatului etc.

Analiza constă în definirea unui enunț precis pentru problemă, precizarea datelor de intrare și a celor de ieșire, elaborarea algoritmului de rezolvare.

Rafinarea algoritmului se referă la structurarea și transformarea lui, în vederea obținerii unor programe simple, bine structurate, ușor de înțeles și bine documentate.

Se poate menționa că o problemă dată nu conduce la un algoritm unic. O problemă este decidabilă, dacă există cel puțin un algoritm care să conducă la soluție. În cazul când nu există un asemenea algoritm, problema este nedecidabilă. De asemenea, existența uneia sau a mai multor soluții, pentru diversele cazuri particulare ale problemei, nu implică decidabilitatea acesteia. Invers, o problemă care nu este decidabilă, nu implică absența unor soluții, ci faptul că nu există o metodă care să permită obținerea soluțiilor, în toate cazurile, când ele există. În principiu nu există o metodă pentru descoperirea algoritmilor privind problemele încă nesoluționate. Descoperirea unui algoritm reprezintă un act de creație, bazat pe inteligență, intuiție și, într-o bună măsură, pe experiență.

Programarea se referă la descrierea algoritmului cu ajutorul unui limbaj de programare, care conține instrucțiuni cu semnificații bine precizate pentru calculator, având ca rezultat generarea programului sursă.

Intrucțt, la nivelul calculatorului, instrucțiunile și datele se reprezintă sub forma unor șiruri de cifre binare, este necesară utilizarea unor notații specifice, care să permită scrierea programului într-un limbaj simbolic.

Instrucțiunile simbolice sînt compuse, în general, din parametri de identificare, de specificare a operațiilor sau acțiunilor care trebuie efectuate și de indicare a operanzilor.

Limbajele de programare s-au dezvoltat în două direcții: limbajele de asamblare, apropiate de limbajul calculatorului, în sensul că ele folosesc notații simbolice (mnemotehnice), pentru a codifica instrucțiunile în cod mașină, și limbajele evolute, independente de calculatorul pe care se execută programul scris într-un asemenea limbaj.

Execuția programului necesită, mai întii, o operație de traducere a programului scris în limbaj de asamblare sau limbaj de nivel înalt, numit program sursă, într-un program reprezentat în instrucțiuni „cod mașină”, numit program obiect.

Există două principii de traducere (translatore): traducerea globală sau compilarea și traducerea instantanee în momentul execuției sau interpretarea.

Operația de compilare este executată de un program (compilator), aflat în memoria calculatorului, (fig. 5.11) pentru care programul sursă constituie datele de intrare în vederea generării datelor de ieșire, reprezentînd programul obiect.



Fig. 5.11. Compilarea și execuția unui program

În timpul compilării se detectează erorile sintactice, privind respectarea regulilor de scriere corectă a programului în limbajul dat, și unele din erorile semantice, referitoare la semnificația programelor corecte, la erorile logice detectabile, la compilare. Problema corectitudinii programelor constituie și în prezent un subiect de cercetare.

Prezența compilatorului în memoria calculatorului nu mai este necesară, după compilarea programului sursă. Programul obiect poate fi executat cu diverse seturi de date de intrare. El poate fi stocat pe un suport magnetic și încărcat, în vederea execuției, ori de cîte ori este nevoie.

Interpretarea constă în traducerea programului, scris în limbaj evoluat, instrucțiune cu instrucțiune, în momentul execuției sale. Programul interpretor este rezident în memoria calculatorului, pe durata execuției programului sursă (fig. 5.12). În acest caz nu există program obiect. Limbajele interpretate / interpretative sînt, în general, interactive. Acesta este cazul limbajului BASIC, care permite dezvoltarea și modificarea programelor într-o manieră conversațională, facilitînd astfel punerea la punct a programelor. Limbajele interpretative conduc la programe cu o viteză mai mică de execuție, mai puțin structurate, mai dificil de înțeles și modificat, sub aspect logic.



Fig. 5.12. Interpretare-execuție

Programarea în limbaje evaluate implica existența altor programe: utilitare, biblioteci de subrutine (pentru I/E, funcții matematice: rădăcină pătrată, logaritmi, exponențiale etc), sistemul de exploatare-operare.

Sistemul de exploatare (operare) permite manipularea automată a echipamentelor periferice, secvențierea automată a unor faze ca: citirea programului sursă, compilarea/interpretarea, tipărirea la imprimantă a unor mesaje, vizualizarea rezultatelor pe display, execuția programului obiect, editarea rezultatelor etc.

Între programele utilitare se întâlnește și editorul, care permite scrierea și corectarea unui program sub forma unui text. De asemenea, se pot menționa *programele utilitare pentru depanarea programelor sursă, a programelor pentru gestiunea fișierelor*, în cazul existenței unor memorii externe pe disc sau bandă magnetică.

În cazul calculatorului HC-85, interpretorul pentru limbajul BASIC înglobează facilitățile de editare, de lucru cu echipamentele periferice și subrutinele matematice, astfel încît *acest calculator apare, pentru utilizator, ca o „mașină BASIC“*.

6.1. Configurație

Microcalculatorul HC-85 face parte din categoria calculatoarelor personale, neprofesionale, de uz general.

Performanțele sale superioare, costul redus, fiabilitatea ridicată, portabilitatea, extensibilitatea, disponibilitatea unui software de sistem și aplicații, orientat către utilizator, reprezintă câteva din caracteristicile acestui calculator personal, care recomandă utilizarea lui în diverse domenii: cercetarea științifică, proiectare asistată de calculator, învățămînt, medicină, automatizarea unor procese industriale, controlul stațiilor pilot, al sistemelor de măsurare etc.

Într-o configurație completă (fig. 6.1) microcalculatorul HC-85 (se recomandă folosirea casetei de prezentare) constă din:

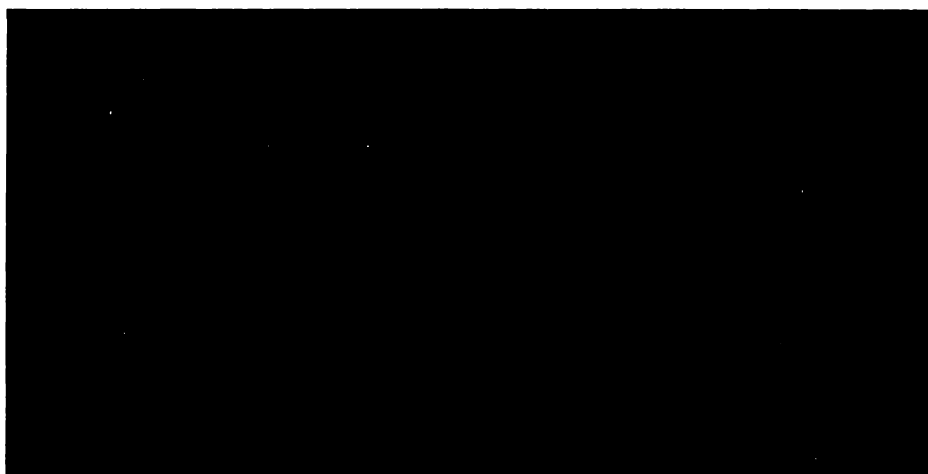


Fig. 6.1. Schema bloc a calculatorului HC-85

- unitatea centrală, pe o singură plachetă, construită cu microprocesorul Z80A, memorii EPROM — 16 Ko și RAM — 48 Ko, circuite integrate pe scară simplă și medie, pentru realizarea interfețelor cu echipamentele de I/E;
- tastatură normală, cu 40 taste, cu semnificații alfabetice și funcționale;
- televizor alb/negru sau color (PAL), monitor color RGB, pentru afișare;
- casetofon audio.

Tastatura și unitatea centrală sînt plasate în aceeași casetă (fig. 6.2)*, care este prevăzută în partea posterioară cu conectori pentru: sursa de alimentare, TV, monitor RGB, casetofon, două manete pentru jocuri (joy-stick), conector pentru interfață.



Fig. 6.2.

Aceasta din urmă asigură extinderea capacităților sistemului, prin posibilitatea de folosire a unei unități simple/duale de disc flexibil (5" 1/4), a unei imprimante seriale de I/E și conectarea într-o rețea de calculatoare personale HC-85.

În eventualitatea că utilizatorul dorește să cupleze la HC-85 echipamente nestandard, se poate folosi conectorul plachetei unității centrale, la care sînt prezente semnalele necesare de date, adrese și comenzi. Descrierea acestor semnale se va face ulterior.

6.2 Unitatea centrală: microprocesorul Z 80, memoria RAM, ROM

Unitatea centrală este prezentată sub formă de schemă bloc în figura 6.3. În componența ei intră următoarele resurse funcționale: unitatea centrală de prelucrare (microprocesorul Z80A), memoria ROM (16 Ko), memoria video și de date (RAM — 16 Ko), memoria suplimentară RAM (32 Ko), sincrogenatorul, blocul de control, interfața video, interfața cu tastatura — casetă magnetică și difuzorul, codificatorul PAL, modulatorul TV și sursa de alimentare.

Unitatea centrală de prelucrare folosește microprocesorul Z80A, pe 8 biți, realizat în tehnologia NMOS și plasat într-o capsulă cu 40 terminale.

Terminalele, în afara celor care asigură tensiunea de alimentare +5V, masa, ceasul și inițializarea (RESET), se pot considera conectate la magistralele de date, adrese și comenzi.

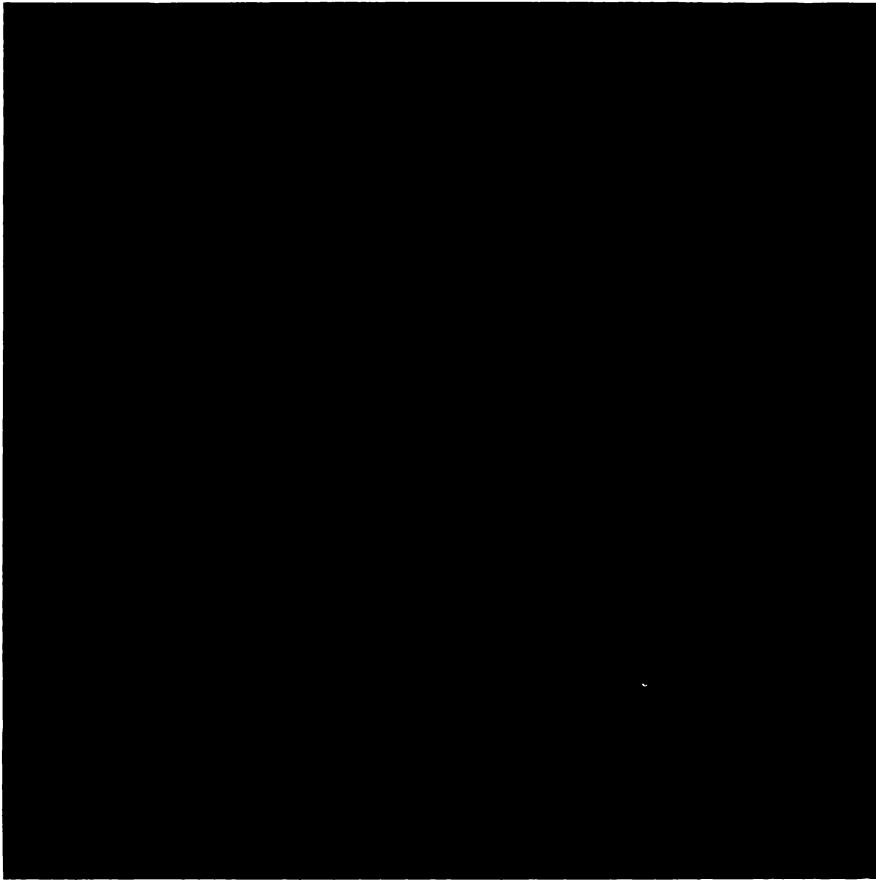


Fig. 6.3. Schema bloc a unității centrale, la nivel funcțional

Magistrala de date (D0—D7) este bidirecțională și poate intra în starea de mare impedanță. Ea este folosită pentru schimbul de informații între procesor, pe de-o parte și memoria/dispozitivele de I/E, pe de altă parte.

Magistrala de adrese (A0—A15) furnizează adresele necesare pentru selecția celulelor de memorie și a porturilor de I/E. Spațiul de adresare permite manipularea a 65536 celule de memorie și respectiv, a 256 porturi de I/E.

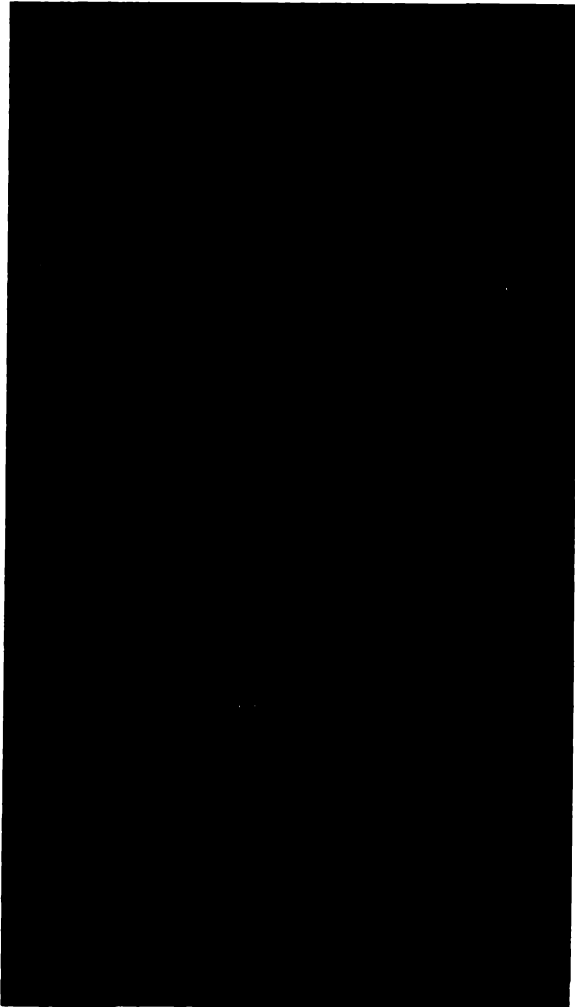
Magistrala de comenzi cuprinde semnalele necesare coordonării transferului de date între microprocesor și memorie/porturi de I/E.

Pe lângă alte operații, asociate cu întreruperile și cererile de magistrală, microprocesorul execută în principal următoarele activități:

- citire/scriere date, din/in memorie,
- citire/scriere date, de la/la un port de I/E,
- operații aritmetice-logice asupra datelor.

Microprocesorul Z80A execută 158 de instrucțiuni distincte. În cazul de față, el operează la o frecvență de tact de 3,5 MHz.

În figura 6.4 se prezintă terminalele microprocesorului, care au următoarele semnificații:



A0 — A15: liniile semnalelor de adrese reprezintă ieșiri cu trei stări, active pe nivel-ridicat. Liniile A0—A7 sînt folosite pentru a selecta unul din cele 256 porturi de I/E. Pe durata perioadei de reîmprospătarea memoriei dinamice, biții A0—A6 conțin adresa de reîmprospătare;

D0 —D7: liniile semnalelor de date au un caracter bidirecțional, sînt active pe nivel ridicat, pot intra în starea de mare impedanță și asigură transferul informației între microprocesor și memorie / porturile de I/E;

M1: linie de ieșire, activă pe nivel coborît, indică ciclul mașină în care se citește primul octet al unei instrucțiuni;

MREQ: linie de ieșire cu trei stări, activă pe nivel coborît, specifică prezența unei adrese de celulă de memorie, pe liniile A0—A15;

IOREQ: linie de ieșire cu trei stări, activă pe nivel coborît, indică prezența unei adrese de port de I/E, pe liniile A0—A7;

RD: linie de ieșire cu trei stări, activă pe nivel coborît, specifică o operație de citire de date, de la memorie sau de la un port de I/E, care trebuie să forțeze datele pe liniile D0—D7;

Fig. 6.4. Terminalele microprocesorului Z80

WR: ieșire cu trei stări, activă pe nivel coborît, indică prezența datelor pe liniile D0—D7, pentru a fi înscrise în memorie sau la un port de ieșire;

RFSH: ieșire, activă pe nivel coborît, specifică prezența, pe liniile A0—A7, a adresei de reîmprospătare pentru memoria dinamică;

HALT: ieșire, activă pe nivel coborît, indică terminarea execuției instrucțiunii HALT;

WAIT: intrare, activă pe nivel coborît, prin care memoria sau portul adresate semnalizează că nu sînt pregătite pentru transferul de date;

INT: intrare, activă pe nivel coborît, reprezintă o cerere de întrerupere din partea unui echipament de I/E;

6. STRUCTURA, COMPONENTE HC-85

NMI: intrare, activă pe front negativ, indică prezența unei cereri de întrerupere nemascabile, cu prioritatea mai mare decât **INT**;

RESET: intrare, activă pe nivel coborât, care forțează în zero contorul programului și inițializează procesorul;

BUSRQ: intrare, activă pe nivel coborât, prin care se solicită, din partea unui dispozitiv extern, controlul asupra liniilor de adrese, date și comenzi;

BUSAK: ieșire, activă pe nivel coborât, care specifică unui dispozitiv extern trecerea liniilor de adrese, date și a unora din liniile de comenzi în starea de mare impedanță, pentru a fi controlate de către dispozitivul în cauză;

Φ : semnal de ceas monofazic, cu frecvența de 3,5 MHz (în cazul de față), generat extern.

Structura internă a microprocesorului Z80. Pentru programator, structura internă a microprocesorului Z80 apare ca în figura 6.5.

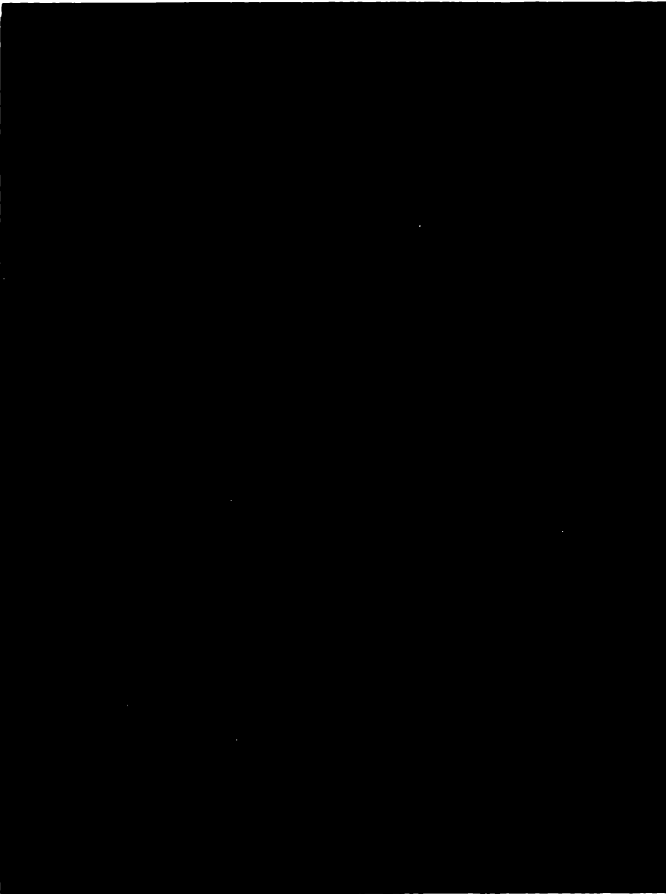


Fig. 6.5. Structura microprocesorului Z80 la nivelul registrelor: a) seturile de registre; b) bistabilele de stare pentru întreruperile mascabile; c) bistabilele modurilor de tratare a întreruperilor mascabile; d) indicatorii de condiții

Microprocesorul Z80 dispune de două seturi de registre, cel de-al doilea fiind alcătuit din dublurile registrelor F, A, B, C, D, E, H, L. Trecerea de la un set la altul se efectuează prin instrucțiunea EXCHANGE (EX AF, AF').

Registrele acumulative (A, A') și registrele indicatoare de condiții (F, F'). Registrele acumulative sînt organizate pe 8 biți și au asociate registrele indicatoare de condiții. Structura informației și registrele de condiții sînt prezentate în figura 6.5. d.

Indicatorii de condiții sînt poziționați automat, ca urmare a operațiilor efectuate în UAL, și pot fi testați prin instrucțiuni de transfer condiționat, în vederea efectuării unor ramificații în program. Indicatorii au următoarele semnificații:

S — *semn*: se poziționează în conformitate cu semnul rezultatului (0 — pentru rezultat pozitiv sau zero și 1 — pentru rezultat negativ);

Z — *rezultat zero*: se poziționează în 1 pentru un rezultat egal cu zero;

H — *transport auxiliar*: se poziționează în 1 ca urmare a apariției unui transport/imprumut spre / de la bitul patru al acumulatorului;

P/V — *paritate/depășire*: indică paritatea rezultatului în acumulator, în cazul operațiilor logice sau depășirea aritmetică, în cazul operațiilor cu numere reprezentate în complementul față de doi;

N — *indicator de adunare/scădere*: specifică tipul instrucțiunii executate înaintea operației de corecție, la operarea în binar-zecimal;

C — *transport*: se poziționează în 1 ca urmare a apariției unui transport/imprumut în afara rangului de semn / în rangul de semn.

Registrele B-L și B'-L' pot fi folosite individual, ca registre de 8 biți, sau asamblate în perechi **B-C**, **D-E**, **H-L** și **B'-C'**, **D'-E'**, **H'-L'**, ca registre de 16 biți. Seturile de registre se pot selecta prin instrucțiunea **EXX**.

Registrul contorului programului PC are 16 biți și conține adresa instrucțiunii următoare, în timpul execuției instrucțiunii curente.

Indicatorul adresei vîrfului stivei SP are 16 biți și indică adresa celei care reprezintă vîrfurile stivei.

Registrele index IX și IY au câte 16 biți; ele conțin constantele de indexare a adresei.

Registrul I, cu o lungime de 8 biți, permite adresarea indirectă a unei locații de memorie, în urma unei cereri de întrerupere; perifericul care solicită întreruperea furnizează primii opt biți mai semnificativi, în timp ce registrul **I** asigură ultimii opt biți mai puțin semnificativi.

Registrul R este folosit pentru reîmprospătarea memoriei dinamice. Conținutul său este transmis pe liniile **A0-A6**, simultan cu semnalul **RFSH**.

Bistabilii IFF1 și IFF2 specifică starea sistemului de întrerupere, al microprocesorului, pentru întreruperile mascabile. **IFF1=0/1** — dezactivat/activat; **IFF2=IFF1** pe durata servirii unei întreruperi nemascabile.

Bistabilii IMF_a, IMF_b specifică modurile programate, pentru răspunsul la cererile de întrerupere mascabile: *modul 0-IMF_a/IMF_b=0/0*, *modul 1-IMF_a/IMF_b=1/0*, *modul 2-IMF_a/IMF_b=1/1*

Memoria adresată de microprocesor are o capacitate totală de 64 Ko, din care 16 Ko — cu conținut fix (EPROM) și 48 Ko — cu conținut variabil (RAM).

Memoria EPROM, care ocupă spațiul de adresare de la 0 la 16383, conține codurile mașină ale interpretorului, pentru limbajul BASIC, și subrutinelor folosite pentru manipularea hardware-lui microcalculatorului HC-85 (subrutina care permite citirea caracterelor introduse de la tastatură, subrutina care controlează sunetul la difuzor, subrutina pentru scrierea/citirea de la/casetofon etc). Această memorie este realizată fizic cu circuitele 2716, avînd fiecare o capacitate de 2 Ko.

Memoria EPROM este conectată direct pe magistrala UCP și operează independent de restul memoriei. Ea poate fi dezactivată forțind semnalul ROMCS, de la conectorul cu 28 de terminale al plachetei, la +5V. În acest mod, la conector, se poate plasa o altă memorie EPROM, cu alt conținut.

Memoria RAM ocupă spațiul de adresare de la 16384, la 65535. Ea este împărțită în două secțiuni.

Prima secțiune, cu o capacitate de 16 Ko, plasată în spațiul de adresare 16384 — 32767, poartă numele de memorie video și de program. Ea stochează, atât informația care se afișează pe ecranul televizorului, cât și informația suplimentară, care se referă la: atributele caracterelor afișate pe ecran, tamponul datelor pentru imprimantă (32 de caractere — o linie ecran), variabilele de sistem, programul BASIC etc.

Memoria video și de program este citită la intervale fixe de către sincrogenerator, pentru a transmite spre TV informația video și atributele de culoare.

Unitatea centrală de prelucrare adresează, de asemenea, memoria video și de program, în scopul modificării imaginii, a atributelor de culoare, a variabilelor de sistem sau pentru a stoca programe BASIC sau date.

Din cele de mai sus rezultă că această memorie este de tip biport, impunându-se rezolvarea conflictelor, care apar la coincidența cererilor de acces din partea sincrogeneratorului și a UCP. Conflictul se va rezolva întotdeauna în favoarea sincrogeneratorului, prin oprirea temporară a ceasului UCP. Această situație provoacă unele neajunsuri în privința programelor care au bucle de temporizare și care sînt stocate în această secțiune de memorie. Pentru controlul exact al temporizărilor, prin program, se recomandă plasarea programelor respective în memoria suplimentară.

Sincrogeneratorul adresează memoria video pentru a citi doi octeți: unul din zona de afișare, în scopul controlării a 8 puncte de pe ecran și altul din zona de atribute, pentru a comanda culoarea.

A doua secțiune a memoriei RAM reprezintă memoria suplimentară, cu o capacitate de 32 Ko, plasată în spațiul de adresare 32768 — 65535. Ea este folosită pentru stocarea programelor BASIC, a datelor etc.

Memoria suplimentară este legată direct pe magistrala de date a UCP și prin intermediul unor multiplexoare, la magistrala de adrese. Aceasta permite



Fig. 6.6. Structura memoriei după conținut

utilizarea ei de către UCP, chiar dacă sincrogeneratorul adresează memoria video, deoarece magistralele lor sînt separate printr-o barieră rezistivă.

Memoria RAM este realizată cu circuite dinamice 4116, cu o capacitate de 16 Ko/circuit.

Organizarea informației în memorie este prezentată în figura 6.6.

6.3. Subsistemele de intrări-ieșiri: interfețe cu televizorul, tastatura, casetofonul, difuzorul; conectorul de extensii

Sincrogeneratorul asigură semnalele de adresă și comandă pentru explorarea periodică a memoriei video, semnalele primare de sincronizare de linie și cadre, în vederea obținerii unei imagini stabile pe ecranul TV.

Imaginea pe ecran este reprezentată prin 256 puncte, pe fiecare din cele 192 linii utile pe cadru. Astfel, se pot afișa pe ecran 24 de rînduri, cu cîte 32 de caractere pe rînd.

Atributele de culoare sînt stabilite la nivel de caracter (matrice de 8×8). În zona de 8×8 puncte a unui caracter, un punct poate avea numai două culori, la un moment dat: „cerneala“ (INK) și „hîrtia“ (PAPER), specificate cu cîte 3 biți, în octetul de atribute. *Formatul octetului unui atribut, pentru un caracter (8×8) puncte este următorul:*

D7	D6	D5	D4	D3	D2	D1	D0
F		*	PAPER			INK	

F — afișare intermitentă a caracterului (flash-clipitor), activ pe 1 (D7);

PAPER — specifică una din cele 8 culori pentru fond (D5—D3)

INK — indică una din cele 8 culori pentru punctele caracterului (D0—D2).

Codurile culorilor sînt următoarele:

000 — negru	100 — verde
001 — albastru	101 — bleu (cyan)
010 — roșu	110 — galben
011 — magenta	111 — alb.

După cum s-a mai arătat, informația privind imaginea TV este stocată în memorie, în două zone adiacente:

— zona de informații video, cu adresele 16384—22527,

— zona de atribute video, cu adresele: 22528—23295.

Zona de informații specifică pentru fiecare punct dacă este de tip „PAPER“ (0) sau de tip „INK“ (1).

Zona de atribute va indica, pentru fiecare matrice de 8×8 puncte culorile.

Blocul de control asigură semnalele de selecție pentru memoria RAM video și program.

Interfața video generează semnalele standard R, G, B și de sincronizare, pentru monitorul color, utilizînd informația din memoria video. La fiecare grup de 8 octeți, reprezentînd un caracter, corespunde un octet de atribute, cu structura dată mai sus. Aceste informații sînt înscrise în mai multe registre, dintre care unele funcționează serial, pentru a asigura controlul imaginii la nivel de punct.

Interfața video mai furniează informația de culoare pentru bordura ecranului, folosind biții D0—D2, ai portului de ieșire cu adresa 254 (FE). De asemenea, generează informația de sincronizare linie/cadru/stingere, pe baza semnalelor primare furnizate de sincrogenerator.

Interfața cu tastatura asigură preluarea informației de la cele 40 de taste cu semnificații alfanumerice și funcționale.

Tastatura este formată dintr-o matrice de 8×5 trasee. La intersecțiile liniilor și coloanelor sînt plasate tastele. Liniile matricei sînt conectate la adresele A8—A15, ale magistralei de adrese. Coloanele matricei sînt legate la liniile D0—D4, ale magistralei de date. Conexiunile respective sînt efectuate prin intermediul unor circuite separatoare. Tastatura este explorată la fiecare 20 ms, prin lansarea unei cereri de intrerupere, către UCP, la sfîrșitul afișării



Fig. 6.7. Matricea de trasee a tastaturii

fiecărui cadru video. Ca urmare a intreruperii, UCP va executa instrucțiuni de intrare (IN) de la portul 254 (FE). Traseele coloanelor sînt conectate la +5 V prin intermediul unor rezistențe de 10 KΩ (fig. 6.7). Pentru a stabili tasta apăsată, care va realiza un contact galvanic la intersecția coloanei cu linia respectivă, este necesar ca, periodic, una din liniile de adrese să fie adusă la un potențial coborît (0 logic), în timp ce liniile celelalte de adrese sînt menținute la un potențial ridicat (1 logic). Bitul poziționat în 0, în grupul D0—D4, va corespunde coloanei conectate la linia de adrese forțată la zero.

În scopul explorării, în maniera arătată, a liniilor de adresă A15,—A8 menținînd pentru A7—A0 valoarea 254, este necesar să se execute instrucțiuni de intrare cu următoarele adrese:

<i>Instrucțiuni</i>	<i>Linie de adrese forțată la zero</i>	<i>Taste selectate.</i>
IN 32766	A15	SPACE, SS, M, N, B
IN 49150	A14	CR, L, K, J, H
IN 57324	A13	P, O, I, U, Y
IN 61438	A12	0, 9, 8, 7, 6
IN 63486	A11	1, 2, 3, 4, 5
IN 64510	A10	Q, W, E, R, T
IN 65022	A9	A, S, D, F, G
IN 65278	A8	CS, Z, X, C, V

Pentru detalii privind specificațiile tastelor se poate apela la caseta de prezentare sau/și la paragraful corespunzător din capitolul referitor la limbajul BASIC.

Interfața cu casetofonul asigură citirea/stocarea programelor și a datelor de pe/pe caseta magnetică.

Conectorul audio are la terminale următoarele semnale:

1,4 — ieșire: nivel 500mV, impedanța de ieșire 500Ω.

3,5 — intrare: 1—4V, impedanța sursei de semnal max. 10 KΩ.

Ieșirea audio este comandată cu ajutorul unui bistabil a cărui stare este controlată prin bitul D3, al portului de ieșire, cu adresa 254 (FE). Forțând alternativ în 0 sau 1 acest bit, se poate genera o formă de undă dreptunghiulară la ieșirea liniei audio. Cu un volum constant, frecvența semnalului va depinde de durata intervalului de timp în care starea bistabilului de ieșire a rămas neschimbată.

Citirea informației de pe suportul magnetic implică amplificarea și filtrarea semnalului, ceea ce va permite, în continuare, comanda bitului D6, al portului de intrare cu adresa 254 (FE).

Pe suportul magnetic, fișierele sînt înregistrate sub forma a două blocuri numite *header (antet)* și *bloc de date*.

Header-ul constă din 19 octeți de date, dintre care numai 17 sînt generați de utilizator (fig. 6.8).



Fig. 6.8. Structura blocului header

Primul octet al header-ului sau al blocului de date, notat cu FLAG, este generat de rutina de salvare SAVE, pentru a face deosebirea între un bloc *header* (FLAG=0) și un *bloc de date* (FLAG=255).

Octetul 19 al header-ului sau ultimul octet al blocului de date conține informația de paritate, ceea ce permite detectarea erorilor la încărcarea informațiilor de pe casetă.

Primul octet și ultimul octet sînt adăugați în mod automat de către rutina SAVE.

Cei 17 octeți ai blocului *header*, furnizați de utilizator, au următoarele semnificații:

TIP — un octet, tipul *blocului de date*:

0 — program BASIC,

1 — tablou numeric,

2 — tablou de tip șir,

3 — cod mașină sau imagine ecran.

NUME FIȘIER — zece octeți. **LUNGIMEA blocului de date** — doi octeți.

Semnificațiile următorilor patru octeți depind de tipul *blocului de date* descris în *header*. Dacă **TIP**=0, atunci octeții 14 și 15 specifică numărul liniei pentru *autostart*, al programului BASIC, iar octeții 16 și 17 indică numărul de

octeți din partea de program a fișierului (salvarea programului BASIC implică salvarea zonei de program și de date). Dacă $TIP=1, 2$, se utilizează numai octetul 15, pentru a specifica numele tabloului. Dacă $TIP=3$, se folosesc numai octeții 14 și 15, pentru a specifica adresa de la care trebuie să se încarce octeții de date. *Blocul de date* este precedat de octetul FLAG (255) și terminat cu octetul de paritate. Ceilalți octeți ai blocului reprezintă imagini ale zonei de memorie salvate. Octetul de paritate este generat înainte de salvarea pe suport magnetic a header-ului și a *blocului de date*, prin aceea că fiecare octet, care este transmis spre ieșire, reprezintă rezultatul unei operații SAU-EXCLUSIV între informația propriu-zisă și octetul curent de paritate. Valoarea inițială a octetului de paritate este chiar valoarea octetului FLAG. La citirea informației de pe suportul magnetic are loc o operație similară. Citirea corectă a blocului conduce la o valoare finală 0, a octetului de paritate.

Semnalul fizic pentru fiecare bloc începe cu un ton rafala, care durează 5 s, pentru header și 2 s, pentru blocul de date (fig. 6.9). Perioada undei dreptunghiu-

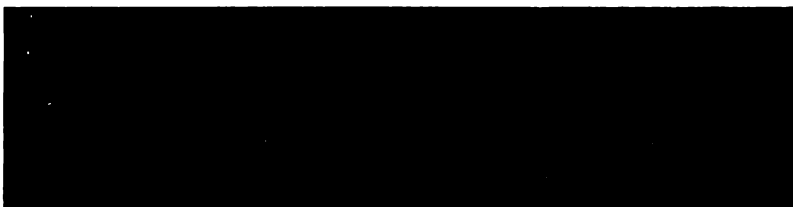


Fig. 6.9. Structura semnalelor folosite pentru stocarea informației pe suportul magnetic

lare corespunzătoare tonului este de 1,2388 ms. Terminarea tonului este marcată de un semnal de sincronizare, cu o valoare coborâtă pe durata a 190,6 μ s și cu valoare ridicată pe durata a 210 μ s. În continuare, fără nici o pauză, se transmit datele, care reprezintă șiruri de biți 0 și 1. Zero este reprezentat printr-un semnal, care are valoarea coborâtă timp de 244,3 μ s și valoarea ridicată tot timp de 244,3 μ s. Unu are drept corespondent un semnal similar, dar cu durate duble pentru cele două stări.

Interfașa pentru difuzor primește un semnal dreptunghiular, corespunzător modificării stării unui bistabil, comandat de bitul D4, al portului de ieșire cu adresa 254 (FE).

Codificatorul PAL are ca intrări semnalele R, G, B și semnalul de sincronizare SY. Pe baza semnalelor R, G, B se obțin semnalele Eu și Ev, reprezentând diferențele între semnalele B și G, respectiv R și G, care, împreună cu semnalul de *burst*, obținut din SY, se aplică la intrările circuitului TCA 650. Acest circuit generează semnalul de cromaticitate, frecvența subpurtătoare PAL obținându-se de la un oscilator cu frecvența de 4,433618 MHz. Un alt circuit generează semnalul de luminanță, pornind tot de la semnalele R, G, B. Prin combinarea într-un etaj final a semnalelor de cromaticitate și luminanță se obține semnalul video complex, codificat PAL. Acesta poate fi utilizat pentru atacarea unui monitor PAL sau, după modulare, a unui televizor color.

Sursa de alimentare este constituită dintr-un alimentator extern, care furnizează o tensiune redresată de minimum 9 V, în sarcină. Tensiunea de +5 V se obține cu ajutorul unui stabilizator integrat LM 7805, plasat pe un radiator. Tensiunile de +12 V și -5 V sînt generate de un convertor cu doi tranzistori

și cu componentele pasive corespunzătoare. Pentru alimentarea eventualelor montaje, plasate pe conectorul de extensie, se recomandă limitarea curentului la maximum 100 mA, de la sursa de +5 V.

Conectorii. Pentru cuplarea unor echipamente periferice convenționale / neconvenționale, în varianta standard a calculatorului HC-85, sînt prevăzuți mai mulți conectori: de extensie, video, pentru manete de tip joy-stick, audio și de antenă.

Conectorul de extensie are 2×28 terminale (fig. 6.10) la care sînt accesibile liniile magistralelor de date, adrese și comenzi ale sistemului. Semnificațiile semnalelor sînt identice cu cele prezentate în cadrul descrierii microprocesorului Z80.



Fig. 6.10. Conectorul de extensie

În continuare se vor face unele precizări privind particularitățile de folosire a unora dintre semnalele accesibile la conector.

13A — $\overline{\text{INT}}$: intrare, întrerupere mascabilă conectată la terminalul $\overline{\text{INT}}$ al microprocesorului și printr-o rezistență de 680Ω la logica generatoare de întreruperi, de pe placa UC. Este activă pe nivel coborît și poate fi utilizată de un dispozitiv extern, pentru a solicita întreruperi.

14A — $\overline{\text{NMI}}$: intrare; întrerupere nemascabilă, activă pe front negativ. Poate fi utilizată de un dispozitiv extern, pentru a forța execuția unei rutine cu adresa de start **102 (66H)**.

17A — $\overline{\text{IORQ}}$: ieșire, semnal de comandă pentru I/E, activ pe nivel coborît. Partea mai puțin semnificativă a magistralei de adrese **A0 — A7**, specifică adresa unui port de I/E. În BASIC, cu o instrucțiune de I/E se poate specifica o adresă de 16 biți, care apare pe liniile de adrese **A0 — A15**, cînd $\overline{\text{IORQ}}$ este zero.

21A — $\overline{\text{WAIT}}$: intrare, activă pe nivel coborît, este folosită de echipamentele mai lente pentru a se sincroniza cu microprocesorul. Linia $\overline{\text{WAIT}}$ nu trebuie activată mai mult de 1 ms, pentru a nu bloca reîmprospătarea memoriei dinamice.

8B — $\overline{\text{CLOCK}}$: ieșire, semnal de ceas 3,5 MHz. Poate fi utilizat de echipamente externe pentru sincronizarea cu microprocesorul; este blocat cînd procesorul solicită accesul la memoria video, în același timp cu sincrogeneratorul.

13B — $\overline{\text{IORGE}}$: semnal de intrare care, conectat la +5 V, blochează semnalul $\overline{\text{IORQ}}$, ce se aplica la placa UC.

19B — $\overline{\text{BUSRQ}}$: semnal de intrare, conectat la terminalul cu același nume al microprocesorului, prin care un echipament extern solicită preluarea controlului asupra magistralor de date, adrese și a unora dintre comenzi.

25B—ROMCS: semnal de intrare activ pe nivel ridicat; conectat la +5 V dezactivează memoria EPROM de pe placheta și permite activarea unei memorii plasate pe conectorul de extensie.

Conectorul video (fig. 6.11) asigură semnalele necesare conectării unor monitoare color **R, G, B** (*Electronica 001*), **PAL** (*Electronica 002*), monitoare alb/negru, cu intrare video compus, Semnalul **FH/2** are frecvența egală cu jumătate din frecvența de linie. Împreună cu semnalele **SYNC, R, G, B** se poate folosi pentru a obține un semnal video codificat, necesar unui alt sistem TV, utilizând o interfață corespunzătoare.



Fig. 6.11. Conectorul video

Fig. 6.12. Conectorii manetelor 1 și 2

Conectorii pentru manete (fig. 6.12) dublează funcțiile primului rând de taste. Manetele asigură închiderea unor contacte (de microinterruptor) normal deschise, atunci când sunt activate în direcțiile stînga, dreapta, sus, jos. Pe conectori sînt notate tastele dublate în momentul închiderii contactului între terminalul respectiv și terminalele notate cu **SEL1** și respectiv **SEL2**

<i>Funcții</i>	<i>Maneta1</i>	<i>Maneta2</i>
Stînga	Tasta1	Tasta6
Dreapta	Tasta2	Tasta7
Jos	Tasta3	Tasta8
Sus	Tasta4	Tasta9
Foc	Tasta5	Tasta0

Conectorul audio servește la cuplarea unui casetofon audio, pentru citirea/stocarea programelor și datelor. Terminalele sale au următoarele semnificații:

- 1,4 — ieșire 500 mV,
- 3,5 — intrare 1—5 V,
- 2 — masă.

Conectorul TV este folosit pentru cuplarea unui televizor, prin mufa de antenă.

7.1. Algoritmi

În activitățile cu caracter practic se întâlnesc numeroase probleme simple sau complexe, pentru a căror rezolvare se utilizează sisteme de reguli bine definite (instrucțiuni sau comenzi), care indică executantului procedeele de operare.

Sistemele de reguli pot fi studiate înainte de rezolvarea problemei sau pot fi elaborate în procesul rezolvării acesteia.

În cazul în care executantul este un automat sau un calculator, sistemul de reguli trebuie descris în limbajul comenzilor sau instrucțiunilor specifice executantului.

Astfel, algoritmul reprezintă o descriere precisă a succesiunii de activități pe care trebuie să le efectueze executantul, pentru atingerea unui anumit obiectiv (scop) sau pentru rezolvarea problemei date. În continuare se vor prezenta câteva exemple de algoritmi.

Exemplul 1. *Algoritmul lui Euclid* pentru găsirea celui mai mare divizor comun a două numere naturale a și b .

Algoritmul se bazează pe proprietatea că *dacă* $a > b$, atunci cel mai mare divizor comun, pentru a și b , este același cu cel al numerelor $a - b$ și b .

Se va proceda după cum urmează:

1. *dacă* $a = b$, oricare din ele reprezintă rezultatul, *în caz contrar se trece la punctul 2*;
2. se determină care din cele două numere este mai mare și se trece la punctul următor;
3. se înlocuiește numărul mai mare cu diferența dintre numărul mai mare și numărul mai mic, se trece la punctul următor;
4. se trece la punctul 1.

Operațiile specificate în acest algoritm se vor repeta pînă la obținerea rezultatului.

Algoritmii pot descrie și alte activități, care nu implică efectuarea unor calcule.

Exemplul 2. *Algoritmul pentru construcția perpendicularei într-un punct dat C , al segmentului de dreaptă \overline{AB} , cu ajutorul compasului și al riglei.* Se consideră $|\overline{AC}| < |\overline{CB}|$.

1. cu vârful compasului plasat în C și cu deschiderea $|\overline{AC}|$ se găsește pe segmentul \overline{CB} un punct D ;

2. cu vârful compasului plasat în D , se trasează un cerc cu raza $r = |\overline{AD}|$;
3. cu vârful compasului plasat în A se trasează un cerc cu raza $r = |\overline{AD}|$;
4. cu ajutorul riglei se unesc punctele de intersecție ale celor două cercuri dreapta astfel obținută este perpendiculară în punctul C , la segmentul \overline{AB} .

Execuția comenzilor în ordinea indicată conduce la rezultatul dorit.

Exemplul 3. Algoritmul „ghicirii“ unui număr x , la care s-a găndit partenerul de joc.

Se propune partenerului de joc efectuarea următoarelor operații asupra numărului ales:

1. se înmulțește numărul cu 5;
2. la rezultat se adună 6;
3. suma se înmulțește cu 2;
4. se comunică rezultatul r obținut.

Pentru a afla numărul x , se procedează astfel:

1. din rezultatul r se scade 12;
2. din rezultatul obținut la punctul anterior, se înlătură cifra cea mai puțin semnificativă, numărul obținut fiind chiar numărul x

Din ultimul exemplu se constată că executantul poate efectua automat cele două operații legate de determinarea numărului x , fără a soluționa ecuația $(5 \cdot x + 6) \cdot 2 = r$.

Aceasta arată că algoritmul permite rezolvarea problemei fără ca executantul să înțeleagă semnificațiile operațiilor efectuate.

Execuția algoritmului are un caracter formal și poate fi efectuată mecanic, de către un automat corespunzător sau un calculator, prevăzut cu un program adecvat.

Elaborarea algoritmului, pentru rezolvarea unei probleme dintr-un domeniu dat, după cum s-a mai arătat, necesită cunoștințe profunde, atât în domeniul respectiv, cât și în cel al matematicilor. Astfel, după formularea problemei, se impune găsirea modelului matematic sau procedural corespunzător. În continuare se elaborează algoritmul, se face verificarea corectitudinii și a complexității sale. După stabilirea algoritmului problema poate fi rezolvată în mod mecanic.

7.2. Limbajul algoritmic (pseudocod)

Una din etapele importante în pregătirea execuției mecanice a unui algoritm o constituie descrierea lui într-un limbaj adecvat.

În numeroase cazuri scopul urmărit prin descrierea algoritmului este legat de necesitățile de comunicare și documentare. Limbajul folosit în acest caz trebuie să fie apropiat de cel natural, să folosească elemente și notații preluate din algebră. El trebuie să se bazeze pe un sistem de notații și reguli pentru descifrarea univocă și precisă a algoritmului și a execuției sale. În această situație descrierea algoritmului este independentă de limbajele concrete de programare, ceea ce ușurează, într-o anumită măsură, studiarea structurii, corectitudinii și complexității sale.

În continuare un asemenea limbaj va fi denumit limbaj algoritmic sau pseudocod.

Limbajul algoritmic posedă un vocabular constituit din cuvinte care sînt folosite pentru scrierea comenzilor necesare execuției algoritmilor.

O comandă reprezintă o propoziție cu caracter imperativ, care specifică o atribuire de valoare, efectuarea unor operații, calcule, verificări de condiții, transferuri ale comenzii etc.

În vocabularul limbajului se întîlnesc cuvinte rezervate, ale căror semnificații sînt bine precizate, ceea ce ușurează înțelegerea algoritmului. Cuvintele rezervate vor fi scrise cursiv.

Algoritmul descris în acest limbaj trebuie să aibe o denumire. Pentru a specifica acest lucru se va folosi cuvîntul *alg* de la *algoritm*, urmat de numele algoritmului, de exemplu:

alg cel mai mare divizor comun

În continuare se va folosi cuvîntul *start* pentru a marca începutul secvenței de instrucțiuni executabile.

Comenzile (instrucțiunile) succesive se pot scrie pe linii succesive sau pe aceeași linie, folosind ca delimitator punctul și virgula (;).

O succesiune de instrucțiuni executabile poartă numele de secvență.

Cuvintele *terminat* și *stop* marchează sfîrșitul unei secvențe, respectiv — sfîrșitul unui algoritm, conform exemplului următor:

alg nume algoritm

start

secvență

terminat

stop

Analiza algoritmilor pune în evidență existența mai multor structuri: liniare, cu ramificații și cu cicluri.

Algoritmii cu structură liniară sînt reprezentați printr-o secvență ce constă din comenzi simple înlănțuite liniar sau din secvențe înlănțuite liniar, conform exemplului de mai jos:

alg nume algoritm

start

secvența1

secvența2

.....

secvența N

terminat

stop

Pentru descrierea ramificațiilor și a ciclurilor se folosesc cuvinte rezervate corespunzătoare.

Ramificațiile se pot descrie în următoarele moduri:

a) *dacă* condiție *atunci*

altfel secvența1

secvența2

terminat

b) *dacă* condiție *atunci*

secvența1

terminat

Ca exemplu se poate descrie algoritmul determinării acidității unei soluții, folosind hîrtia de turnesol.

alg determinarea acidității unei soluții

start

se toarnă într-o eprubetă 5 ml soluție;
se introduce în eprubetă hîrtia de turnesol;

dacă hîrtia se înroșește *atunci*

soluția este acidă

altfel

dacă hîrtia devine albastră

atunci

soluția este o bază

altfel

soluția este neutră

terminat

terminat

stop

Ciclurile conțin secvențe de instrucțiuni care se repetă de un număr de ori, în conformitate cu o condiție impusă. Condiția poate fi testată la începutul său la sfîrșitul secvenței, conform exemplelor de mai jos:

a) *cît timp* condiție *execută*

secvență

terminat

b) *ciclează*

secvență;

cît timp condiție

terminat

Se consideră algoritmul de funcționare a unui regulator de temperatură, bipozițional, care stabilește / întrerupe circuitul de alimentare cu energie electrică al unui radiator în funcție de temperatura mediului și de valoarea prescrisă de 20°C.

a) *cît timp* temp. mediu < 20°C *execută*

stabilește circuitul;

terminat

b) *ciclează*

stabilește circuitul;

cît timp temp. mediu < 20°C

terminat

În cazurile în care numărul de cicluri este dinainte cunoscut, se poate folosi un contor. Contorul are o valoare inițială și trebuie să ajungă la o valoare finală, prin incrementări cu un pas dat. Valorile inițială, finală și pasul sînt cunoscute și reprezintă numere întregi.

Forma generală a ciclului cu contor este următoarea:

pentru contor=val. inițială, val. finală, pas *execută*
secvență
terminat

Ca exemplu se va prezenta calculul valorii funcției $y=5 \cdot x^2$, pentru $5 \leq x \leq 25$, al pasul egal cu 1:

alg calculului valorii funcției $y=5 \cdot x$
start
pentru $i=5, 25, 1$ *execută*
 $y=5 \cdot x^2$
terminat
stop

Tipuri de date. Marea majoritate a problemelor practice sînt legate de prelucrări ale informației. În cadrul descrierii și execuției unui algoritm trebuie să se evidențieze informația inițială (datele de intrare), informația în curs de prelucrare (rezultatele intermediare) și informația finală (rezultatele). Acestea reprezintă mărimi numerice, grafice etc.

Mărimile sînt de două tipuri: *constante*, care nu se modifică pe parcursul execuției algoritmului și *variabile*, care iau diverse valori în timpul execuției algoritmului.

În descrierea unui algoritm, variabilele vor fi reprezentate prin numele lor, scrise în text cu litere cursive.

Mărimile cu care operează un algoritm pot fi numerice (numere naturale, întregi, reale etc) sau nenumerică (cuvinte, tabele, texte, grafice, figuri geometrice etc).

Variabilele numerice vor fi declarate în titlul algoritmului prin precizarea tipului:

întreg i
real x

Variabilele ale căror valori reprezintă cuvinte sau texte vor fi declarate prin
lit:

lit y

*Variabilele numerice pot fi tablouri unidimensionale sau bidimensionale, care reprezintă vectori sau matrici. Elementele acestora vor purta numele vectorului sau matricii precedate de cuvîntul *tab*, de la tablou și vor avea unul sau doi indici, pentru a specifica poziția elementelor date.*

Astfel, un vector v , cu 10 componente reale, va fi declarat ca tablou unidimensional după cum urmează:

real $tab\ v [0 : 9]$

O matrice m , cu 8 linii și 8 coloane, constituită din variabile întregi, va fi declarată ca tablou bidimensional ca în exemplul de mai jos, unde indicii iau valori cuprinse între 0 și 7:

întreg $tab\ m [0 : 7, 0 : 7]$

Precizări privind descrierea algoritmilor în limbajul algoritmic.

Denumirea algoritmului trebuie să conțină declarații ale variabilelor cu care se operează. Este necesar a se preciza variabilele care reprezintă argumen-

tele (datele de intrare) și cele care specifică rezultatele. Pentru acestea se vor folosi cuvintele rezervate: *arg* și *rez*. Pe parcursul execuției algoritmului se mai folosesc și variabile auxiliare, cu caracter temporar, al căror tip trebuie declarat înainte de utilizare.

În cadrul secvențelor de instrucțiuni, în mod frecvent se întâlnește operația de atribuire, cu următoarea formă generală:

variabila ← *expresie*

Variabila este dată prin nume. În urma evaluării expresiei se obține o valoare, care se atribuie variabilei, al cărei nume constituie termenul stîng al formei de mai sus. Simbolul ← specifică operația de atribuire.

În continuare se vor da unele exemple pentru a ilustra elementele prezentate mai sus.

Exemplu 1. Algoritmul pentru rezolvarea ecuației de gradul 2:

$$a \cdot x^2 + b \cdot x + c = 0$$

alg soluția ec.gr. doi (real *a*, *b*, *c*, *x1*, *x2*, lit *y*)

arg *a*, *b*, *c*

rez *x1*, *x2*, *y*

start

real *D*

$D \leftarrow b^2 - 4ac$

dacă $D < 0$ atunci

$y \leftarrow$ „nu există soluție“

altfel

$y \leftarrow$ „există soluție“

$x1 \leftarrow (-b + \sqrt{D})/2a$

$x2 \leftarrow (-b - \sqrt{D})/2a$

terminat

stop

Se constată că *D* reprezintă o variabilă reală de lucru, cu utilizare temporară.

Exemplul 2. Algoritmul lui Euclid pentru aflarea cmmdc a două numere reale *a* și *b*.

alg cel mai mare divizor comun (real *a*, *b*, cmmdc)

arg *a*, *b*

rez cmmdc

start

real *x*, *y*

$x \leftarrow a$; $y \leftarrow b$ cît timp $x \neq y$ execută

dacă $x > y$ atunci

$x \leftarrow x - y$

altfel

$y \leftarrow y - x$

terminat

terminat

cmmdc ← *x*

stop

Exemplul 3. Algoritmul găsirii celui mai mare număr din două numere date x și y , pe scurt *cmddn*

```

alg cmddn (real  $x, y, z$ )
  arg  $x, y$ 
  rez  $z$ 
  start
    dacă  $y \leq x$  atunci
       $z \leftarrow x$ 
    altfel
       $z \leftarrow y$ 
  terminat
stop

```

Exemplul 4. Algoritmul găsirii celui mai mare număr dintr-un șir de n numere date, pe scurt *cmdnn*.

```

alg cmdnn (ntreg  $n$ , real tab  $x[1:n]$ , real  $xmax$ )
  arg  $x, n$ 
  rez  $xmax$ 
  start
    ntreg  $i$ 
     $i \leftarrow 1; xmax \leftarrow 0$ 
    cît timp  $i \leq n$  execută
      dacă  $xmax < x[i]$  atunci
         $xmax \leftarrow x[i]$ 
      terminat
     $i \leftarrow i + 1$ 
  terminat
stop

```

Exemplul 5. Algoritmul pentru scrierea tablei înmulțirii. Tabla înmulțirii va apare ca un tablou cu 9 linii și 9 coloane de forma următoare:

```

prod[1,1], prod [1,2], ...../....., prod[1,9]
.....
prod [9,1], prod [9,2] ..... prod [9,9]

```

unde: $prod [i, j] \leftarrow i \times j$.

alg tabla înmulțirii (ntreg tab $prod [1:9, 1:9]$)

```

arg ntreg  $i, j$ 
rez prod
start
   $i \leftarrow 1$ 
  cît timp  $i \leq 9$  execută
     $i \leftarrow 1$ 
    cît timp  $j \leq 9$  execută
       $prod [i, j] \leftarrow i \times j$ 
       $j \leftarrow j + 1$ 
    terminat
   $i \leftarrow i + 1$ 
  terminat
stop

```

7. ALGORITMI. PSEUDOCOD

Exemplul 6. Algoritmul înmulțirii a două matrici A și B , având ca rezultat matricea produs C . Matricea A are $n \times m$ elemente, matricea B are $m \times p$ elemente, iar matricea produs va avea $n \times p$ elemente. Formula de calcul pentru elementele matricii produs este următoarea:

$$C[i, j] = \sum_{k=1}^m A[i, k] \times B[k, j], \text{ pentru } \text{toți } i=1, n \text{ și } j=1, p.$$

alg înmulțire matrici (întreg m, n, p , real tab $A[1:n, 1:m], B[1:n, 1:p], C[1:n, 1:p]$)

arg A, B, m, n, p

rez C

start

întreg i, k, j ; real suma

pentru $i=1, n, 1$ execută

pentru $k=1, p, 1$ execută

suma $\leftarrow 0$

pentru $j=1, p, 1$ execută

suma \leftarrow suma $+ A[i, k] \times B[k, j]$

terminat

$C[i, j] \leftarrow$ suma

terminat

terminat

stop

Încorporarea unor algoritmi existenți în algoritmi de complexitate mai mare reprezintă o practică curent întâlnită în programare, care se materializează prin chemarea de funcții și subrutine. Aceasta asigură o anumită structurare a algoritmului, permițând o abordare mai simplă a problemelor de mare complexitate.

Exemplu. Se va da un exemplu simplu de folosire a algoritmului *cmmddn*, pentru a găsi cel mai mare număr din trei numere date (*cmmddn*).

alg *cmmddn* (real a, b, c, d)

arg a, b, c

rez d

start

real x

cmmddn (a, b, x)

cmmddn (x, c, d)

stop

Ca urmare a aplicării algoritmului pe perechea a, b , se obține rezultatul x , reprezentând cel mai mare număr dintre a și b . În continuare, operația se repetă cu x și c , rezultatul fiind d .

Se constată că în cele două cazuri de aplicare a algoritmului *cmmddn*, argumentele au fost declarate anterior.

Algoritmi pentru manipularea informației grafice. În practică apar probleme ale căror rezultate sînt sub formă de desene, grafice, diagrame etc. Acestea se pot reprezenta pe ecranul dispozitivului de afișare TV sau pe un dispozitiv de înregistrare.

Se presupune că desenul se efectuează pe un ecran, în cadrul unui sistem de coordonate x, y , cu originea $(0, 0)$ în colțul din stînga-jos, al ecranului.

Desenul este realizat prin comenzi date unei „penițe” imaginare avînd forma unei săgeți, care se poate deplasa pe ecran cu comenzile următoare: *înainte* (*par 1*), *înapoi* (*par 2*), *dreapta* (*par 3*), *stînga* (*par 4*).

Parametrii *par 1* și *par 2* reprezintă numărul de pași elementari efectuați de peniță în sensurile date, pe direcția indicată de vîrfurile săgeții.

Parametrii *par 3* și *par 4* specifică numărul de grade cu care se poate roti penița, la dreapta sau la stînga, față de direcția curentă.

Lista de comenzi trebuie completată și cu instrucțiunile: *desenează* și *nu desena*, pentru a specifica plasarea peniței pe „hîrtie” și ridicarea „peniței” de pe hîrtie.

În starea inițială penița este plasată în punctul de coordonate $0, 0$, avînd vîrfurile în sus.

Exemplu. Desenarea unui triunghi

```
alg triunghi
  start
    desenează
      dreapta (30)
      înainte (5)
      dreapta (120)
      înainte (5)
      stînga (60)
      înapoi (5)
    nu desena
  stop
```

Exemplu. Desenarea unui pătrat cu latura egală cu 5.

```
alg pătrat
  start
    desenează
      înainte (5)
      dreapta (90)
      înainte (5)
      dreapta (90)
      înainte (5)
      dreapta (90)
      înainte (5)
    nu desena
  stop
```

Algoritmul anterior conține grupuri de instrucțiuni care se repetă, ceea ce sugerează rescrierea lui cu ajutorul instrucțiunii de ciclare.

Exemplu. Desenarea unui pătrat cu latura L .

```
alg pătrat (real L)
  arg L
  start
    întreg i
    i ← 1; desenează
```

```

    cît timp  $i \leq 4$  execută
        înainte (L)
        dreapta (90)
         $i \leftarrow i + 1$ 
    terminat
    nu desena
stop

```

Trecerea în revistă a limbajului algoritmic, permite însușirea primelor noțiuni de programare în limbaje de nivel înalt. Fiecare limbaj de programare are particularitățile sale, ceea ce îi poate conferi o serie de avantaje, pentru anumite tipuri de aplicații. În cadrul limbajului algoritmic nu s-au tratat problemele de citire / scriere a informațiilor de la / la un terminal. De asemenea, structurile de date examinate, reprezentînd scalari (întregi, reali), tablouri uni și bidimensionale (vectori și matrici de numere întregi sau reale) și texte constituie structurile cele mai simple întîlnite în practică.

După cum s-a mai menționat, calculatorul HC-85 dispune de un interpretor memorat pentru limbajul BASIC și de o serie de interpretoare și compilatoare, pentru limbajele LOGO, Pascal, C, Microprolog, LISP, Fortran.

În lucrarea de față vor fi prezentate limbajele BASIC și LOGO.

Partea a IV-a

PROGRAMAREA ÎN LIMBAJUL BASIC PE CALCULATORUL HC-85

Capitolul 8 | Caracteristicile și elementele limbajului BASIC

8.1. Noțiuni introductive

La denumirea de BASIC s-a ajuns prin păstrarea inițialelor denumirii complete din limba engleză „Beginners All purpose Symbolic Instruction Code“ (Codul instrucțiunilor simbolice pentru începători, utilizabil în orice scop).

După cum îi spune și numele, el este destinat cu precădere începătorilor și are diverse și multiple utilizări.

Este ușor, simplu, asimilabil într-un timp scurt, fără a necesita o pregătire prealabilă specială.

BASIC este unul dintre cele mai răspândite limbaje de programare din lume.

Limbajul de programare este conceput special în vederea realizării înțelegerii dintre om și mașină. El poate fi sugestiv asimilat cu „limba“ pe care o înțelege un calculator. Numai cunoscând un astfel de limbaj de programare se poate beneficia de toate facilitățile oferite de un calculator personal.

Limbajul de programare BASIC este alcătuit dintr-un set de instrucțiuni și comenzi, alături de toate regulile de formare și folosire a acestora.

O instrucțiune sau o comandă se recunoaște prin „cuvântul cheie“ ce reprezintă denumirea sa. Această denumire este aleasă întotdeauna sugestiv și provine din prescurtarea cuvântului asociat din limba engleză.

O comandă și o instrucțiune pot avea aceeași denumire, dar deosebirea dintre ele sînt fundamentale și constau în modul de memorare, de execuție și de păstrare în memorie.

Imediat după introducere, o *instrucțiune* se memorează, după care se execută imediat sau după un anumit timp, o dată sau de cîte ori se dovedește a fi necesară; ea se păstrează în memorie pînă la eventuala ei ștergere ce survine numai la cerere.

În cazul unei *comenzi*, după introducere urmează imediat execuția sa și aceasta numai o singură dată, după care execuția nu mai poate fi reluată decît printr-o nouă introducere deoarece ea nu se păstrează în memorie.

Instrucțiunile și comenzile se supun unor reguli de sintaxă (de scriere) și unor reguli de logică (de execuție).

Mulțimea ordonată și finită a instrucțiunilor / comenzilor, ce soluționează o problemă, formează un *program*.

Programul se bazează pe un algoritm.

*Algoritm*ul constă în totalitatea raționamentelor matematice și logice, făcute pas cu pas, în scopul rezolvării unei probleme cu ajutorul calculatorului.

Algoritmul se transpune grafic printr-o schemă logică.

Schema logică este mulțimea tuturor simbolurilor standard cu o semnificație predefinită prin care se reprezintă pașii parcurși în vederea transpunerii unei probleme într-un limbaj de programare.

Schema logică este cu atât mai necesară cu cât problema propusă spre rezolvare este mai complexă.

Ansamblul programelor destinate rezolvării unor clase de probleme este cunoscut sub numele de *soft (software) de aplicație*.

În afara acestei categorii, există programe fără de care utilizarea calculatorului nu este posibilă.

Ansamblul acestor tipuri de programe este cunoscut sub numele de *soft de bază*.

BASIC face parte din componenta soft de bază. El este livrat odată cu calculatorul personal și se încarcă automat în memorie prin simpla pornire a sistemului.

Elementele precizate sub denumirea de soft de aplicație și soft de bază se numesc pe scurt, soft. Acesta este un produs intelectual ce nu poate exista în afara echipamentului fizic pentru care este conceput.

Echipamentul fizic sau calculatorul împreună cu toate perifericele (televizor, casetofon, etc.) este cunoscut sub numele de *hard (hardware)*.

Componentele hard și soft alături de factorul uman alcătuiesc un sistem de prelucrare automată a datelor (sistem informatic).

Cele trei componente ale unui sistem de prelucrare automată a datelor sînt într-o continuă interacțiune și dinamică.

Ele alcătuiesc o unitate, ce înmagazinează multă inteligență, experiență și muncă.

Calculatorul este una din uneltele reprezentative ale secolului XX, ce merită să fie explorată cu atenție, perseverență și devotament, pentru a ajunge un instrument popular, de folos în orice domeniu de activitate.

Este clar că, fără cunoașterea unui limbaj de programare, calculatorul, deși oferă multe posibilități, devine un obiect inutilizabil, fără nici o eficiență. În sprijinul depășirii acestui obstacol, capitolul prezent își propune inițierea în limbajul de programare BASIC.

Sînt necesare cîteva precizări.

Pentru că în lume s-au produs diverse tipuri de calculatoare personale, circulă și multe variante ale limbajului BASIC.

La o variantă a limbajului BASIC se ajunge prin preluarea unui subset de instrucțiuni din limbajul BASIC standard. Selectarea și implementarea diferită a instrucțiunilor / comenzilor unui limbaj de programare este generată de multitudinea diferențelor existente între parametrii fizici ai echipamentelor.

Limbajul BASIC prezentat aici este implementat pe calculatorul personal românesc HC-85, compatibil cu calculatoarele din gama SPECTRUM —SINCLAIR.

8.2. Tastatura calculatorului HC-85

Alfabetul utilizat de HC-85 cuprinde 256 simboluri.

Simbolurile pot fi:

- simboluri simple (litere, cifre, simboluri speciale etc.);
- simboluri compuse (cuvinte cheie ale instrucțiunilor sau comenzilor, nume de funcții etc.).

Fiecare simbol se găsește imprimat pe o tastă.

Tastatura calculatorului HC-85 cuprinde 40 de taste și este similară cu cea a unei mașini de scris: literele și cifrele sînt în aceleași poziții cu excepția literelor Q, Z și M. Tastele sînt plasate pe 4 rînduri, cite 10 pe linie.

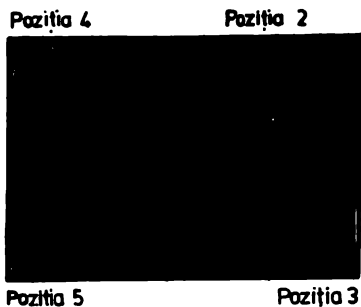


Fig. 8.1.

În centrul tastei apare scris un simbol, în stînga și dreapta sus sau jos alte simboluri, conform pozițiilor din figura 8.1.

Trebuie reținut faptul că, pentru a obține toate simbolurile alfabetului BASIC, unele taste au pînă la șase semnificații.

Aceste semnificații, potrivit celor scrise în pozițiile 1—5 din figură, sînt selectate conform modului de lucru în care se găsește calculatorul, prin apăsarea tastei respective simultan cu tasta:

- CAPS SHIFT (cu prescurtarea CS — prima tastă din rîndul 4);
- SYMBOL SHIFT (cu prescurtarea SS — penultima tastă din rîndul 4).

Dacă o tastă este apăsată mai mult de 2—3 secunde, simbolul începe să se repete.

Semnificația tastelor apăsate apare pe rînd în partea de jos a ecranului, fiecare caracter fiind inserat pe locul cursorului.

Cursorul este săgeata neastîmpărată de pe ecran, care se deplasează singură pe măsură ce alte date se introduc și indică locul unde se face viitoarea tipărire sau înscriere. Cursorul poate fi mutat la stînga cu ← (CAPS SHIFT și 5) sau la dreapta cu → (CAPS SHIFT și 8). Caracterul din stînga cursorului poate fi șters cu DELETE (CAPS SHIFT și 0).

La înscrierea simbolurilor pe tastatură au fost folosite următoarele prescurtări:

RAND	—	în loc de	RANDOMIZE
BRGT	—	în loc de	BRIGHT
INV	—	în loc de	INVERSE
CR	—	în loc de	ENTER
CS	—	în loc de	CAPS SHIFT
SS	—	în loc de	SYMBOL SHIFT
SCR	—	în loc de	SCREEN
CONT	—	în loc de	CONTINUE

8.3. Modurile de lucru

Modul de lucru este modul prin care acționând o tastă, în memorie este introdus unul din simbolurile scrise pe ea în poziția 1—5.

Calculatorul HC-85 are 5 moduri de lucru, desemnate prin literele K, L, C, E și G.

Modul de lucru este inițial afișat în partea stângă jos a ecranului. Litera ce îl desemnează se deplasează automat pe ecran pe măsură ce se introduc date.

În practică, după conectarea la rețea, calculatorul se prezintă (fig. 8.2).

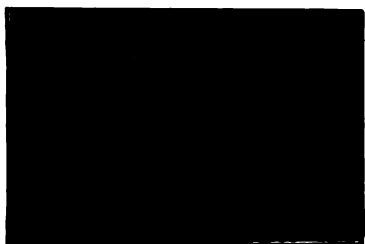


Fig. 8.2.

După această manevră trebuie acționată tasta CR (ultima tastă din rândul 3).

Imperturbabil, pe ecranul curat, în partea stângă jos, se afișează K. Acesta este primul mod de lucru. El poate fi schimbat în alt mod de lucru respectând câteva reguli de tastare.

La rândul lor și celelalte moduri se pot schimba între ele respectând același grup de reguli de tastare.

Prin modul de lucru, calculatorul își atenționează utilizatorul că așteaptă introducerea unui anumit tip de date. Mai mult, dacă nu se introduc datele așteptate (din punct de vedere al logicii limbajului utilizat), pe ecran, acolo unde este greșit, apar semne de întrebare și nu se poate trece mai departe decât dacă se fac corecțiile convenite.

8.3.1. Modul de lucru K

Modul K (keyword=cuvânt cheie) apare atunci când se afișează o comandă (ex: LIST, RUN, SAVE, PRINT etc.) sau o instrucțiune (ex. 100 PRINT, 20 INPUT, 66 LET etc). Aceasta se întâmplă la începutul unei linii program, după simbolul „ : “ (ce separă instrucțiunile de pe aceeași linie) sau după un THEN (cuvânt cheie al unei instrucțiuni de comparație).

În modul K, tastele literale sînt interpretate ca niște cuvinte cheie, potrivit notațiilor din poziția 3, iar tastele numerice sînt interpretate ca simple numere, potrivit notațiilor din poziția 1.

8.3.2. Modul de lucru L

Modul L (letters = litere) alternează cu modul K; imediat ce a fost introdus un cuvânt cheie, modul de lucru K se schimbă automat în L.

În modul L, simbolul scris în poziția 1 pe orice tastă apare prin simpla acționare a acesteia. În cazul unei taste literale apare litera mică a alfabetului.

EXEMPLU:

Introducerea comenzii

PRINT 25

(In limba) BASIC, tipărește numărul 25) se face astfel:

— la început de linie este afișat modul de lucru K, se acționează tasta P ceea ce înseamnă introducerea simbolului din poziția 3, adică PRINT;

— modul de lucru se schimbă automat în L, iar acționarea tastelor numerice 2 și 5 conduce la introducerea simbolurilor scrise în poziția 1 a tastelor numerice specificate.

Atât în modul L cât și în modul K, acționarea simultană a lui SYMBOL SHIFT (SS) și a unei taste numerice este interpretată drept simbolul din poziția 3, iar SYMBOL SHIFT (SS) simultan cu o tastă literală produce simbolul din poziția 2.

EXEMPLU:

Introducerea comenzii

PRINT "25 \$"

(In limba) BASIC, tipărește textul 25 \$) se face astfel:

— la început de linie este afișat modul de lucru K, se acționează tasta P și rezultă PRINT;

— modul de lucru devine automat L, prin acționarea simultană a tastelor SS și P se produce scrierea simbolului " (ghilimele), din poziția 2 a tastei literale P;

— în continuare modul de lucru rămâne tot L, se acționează pe rând tastele numerice 2 și 5 pentru afișarea numărului 25;

— fiind în modul L, pentru afișarea semnului \$, se acționează simultan tasta numerică 4 și tasta SS;

— în final se apasă încă o dată simultan pe tastele SS și P pentru închiderea ghilimelelor.

Acționarea simultană a lui CAPS SHIFT și a unei taste numerice, în modul de lucru L, este interpretată ca simbolul specificat în poziția 4, ca de exemplu EDIT, CAPS LOCK, DELETE ș.a.m.d.

EXEMPLU:

Introducerea comenzii

DELETE

(In limba) BASIC, șterge un număr de caractere) se face astfel:

— pentru ștergerea unui caracter se acționează simultan, o singură dată, tasta 0 (zero) și CS;

— pentru ștergerea unui grup de caractere se acționează simultan tastele 0 (zero) și CS de un număr de ori egal cu numărul caracterelor componente ale acelui grup.

Acționarea unei taste literale simultan cu CAPS SHIFT, în modul de lucru K, nu are nici un efect (sau tasta este interpretată conform poziției 3), iar în modul de lucru L produce conversia literelor mici în litere mari.

EXEMPLU:

Introducerea comenzii

PRINT A+B

(In limba) BASIC, tipărește suma numerelor existente în A și B) se face astfel:

- fiind în modul K se tastează P și pe ecran apare PRINT;
- modul de lucru devine automat L, se acționează simultan tastele CS și A din care rezultă litera mare A;
- tastarea simultană a lui SS și K dă semnul +;
- următoarea tastare se compune din apăsarea simultană a tastelor CS și B din care rezultă litera mare B.

8.3.3. Modul de lucru C

Modul C (capitals=majuscule) este o variantă a modului de lucru L, în care scrierea se face cu litere mari.

Tasta CAPS LOCK (acționarea simultană a tastelor CS și 2) determină trecerea din modul L în modul C. Numai dacă este acționată din nou tasta CAPS LOCK se revine din modul C în modul L.

EXEMPLU:

Introducerea comenzii

PRINT A+B

se poate face și astfel:

- primul pas este identic cu cel din exemplul anterior;
- modul de lucru devine automat L, iar prin tastarea simultană a lui CS și 2 se schimbă în C, după care se apasă direct tasta A pentru tipărirea literei mari A;
- pentru semnul + se procedează ca în exemplul anterior;
- modul de lucru C se păstrează în continuare, iar prin apăsarea directă a tastei B, se obține ultimul caracter dorit, litera mare B.

8.3.4. Modul de lucru E

Modul de lucru E (extended — extins) este utilizat pentru a obține simboluri noi, în special alte instrucțiuni sau comenzi. Pentru a intra în acest mod se acționează simultan ambele SHIFT-uri (CS și SS), iar anularea acestui mod se face automat după prima tastare.

În acest mod, apăsarea unei taste literale generează simbolul scris în poziția 4, iar dacă tasta este apăsată împreună cu o tastă SHIFT se generează simbolul scris în poziția 5 a tastei.

EXEMPLU:

Introducerea comenzii

READ nr

(In limba) BASIC, citește variabila cu numele „nr“) se face astfel:

- în stînga jos pe ecran apare tipărit modul K, el se schimbă în modul E prin apăsarea simultană a tastelor CS și SS;

— în modul de lucru E, apăsarea tastei A conduce la obținerea simbolului din poziția 4 a tastei, respectiv READ;

— modul de lucru se schimbă automat în L, apăsarea pe rând a tastelor N și R conduce la obținerea caracterelor dorite, respectiv variabila cu numele „nr“.

EXEMPLU:

Introducerea comenzii

BEEP 1,3

(în limba) BASIC, sunetul corespunzător unei note muzicale) se face astfel:

— se transformă modul K în modul E prin apăsarea simultană a tastelor CS și SS;

— în modul de lucru E, apăsarea simultană a tastelor Z și CS (sau SS) conduce la obținerea simbolului din poziția 5 a tastei Z, respectiv BEEP;

— modul de lucru se schimbă automat în L, iar pentru obținerea numărului 1 se apasă pe tasta 1;

— modul de lucru nu se schimbă, este tot L, se apasă simultan pe tasta N și SS obținându-se simbolul virgulei;

— ultima tastare, în modul de lucru L, constă în apăsarea tastei 3 pentru introducerea numărului 3.

Apăsarea unei taste numerice în modul de lucru E generează o comandă dacă este acționată împreună cu SYMBOL SHIFT și o secvență de control a culorii dacă este acționată singură.

8.3.5. Modul de lucru G

Modul G (graphics — grafice) se obține prin acționarea tastelor CAPS SHIFT și 9. Anularea acestui mod de lucru se face acționând din nou tastele CAPS SHIFT și 9 sau numai tasta 9.

O tastă numerică dă un mozaic grafic predefinit (în afara tastelor 0 și 9) și orice tastă literală în afara de V, W, X, Y și Z generează un simbol grafic definit de utilizator.

Acționarea lui V, W, X, Y sau Z în modul de lucru G conduce la introducerea unui simbol deja stabilit care reprezintă cea de a șasea semnificație a acestor taste:

V	—	RND
W	—	INKEY \$
X	—	PI
Y	—	FN
Z	—	POINT

EXEMPLU:

Introducerea comenzii

DRAW 10, 50, PI

(în limba) BASIC, trasează conturul unui semicerc) se face astfel:

— la început de linie este afișat modul de lucru K, se acționează tasta W și se obține DRAW;

— modul de lucru devine automat L, acționarea pe rând a tastelor 1 și 0 introduce numărul 10;

- în continuare modul de lucru este tot L, se acționează simultan tasta SS și N pentru obținerea simbolului virgulă;
- modul de lucru fiind tot L se procedează ca înainte pentru tipărirea simbolurilor 5 și 0;
- prin acționarea simultană a tastelor CS și 9 modul de lucru devine G, iar simpla apăsare a tastei X tipărește simbolul PI.

În scopul fixării tuturor posibilităților de transformare a modurilor de lucru între ele în vederea obținerii tuturor simbolurilor specificate pe tastatura calculatorului HC-85, a rezultat necesitatea sistematizării celor prezentate pînă în acest moment. Pentru aceasta s-au realizat fig. 8.3. și tabelul 8.1.



Fig. 8.3.

În figura 8.3 sînt indicate cu ajutorul săgeților toate direcțiile posibile de schimbare a unui mod de lucru în alt mod de lucru, potrivit acționării corespunzătoare a tastei/tastelor specificate chiar de-a lungul săgeților, cu precizările:

- | | |
|--|---|
| <ul style="list-style-type: none"> a) modul de lucru K este dublu încercuit pentru a scoate în evidență că acesta este întotdeauna primul mod de lucru pe care îl afișează calculatorul; b) imediat după tastarea unui simbol, modul de lucru K se transformă automat în L, existînd însă posibilitatea schimbării lui înainte de aceasta, conform săgeților duble; c) numai după tastarea cel puțin a unui simbol într-un anumit mod de lucru, | <ul style="list-style-type: none"> se poate trece în alt mod de lucru, conform săgeților întrerupte. d) fie că s-a tastat sau nu un simbol într-un anumit mod de lucru, se poate trece în alt mod de lucru, conform săgeților pline (continue); e) fiind în modul de lucru C, se poate trece în modul G sau E, dar în momentul schimbării acestora se revine tot în modul C. |
|--|---|

Tabelul 8.1 conține corespondența dintre simbolul precizat pe tastă într-o anumită poziție și un mod de lucru, epuizînd pe rînd atît cele 5 moduri de lucru existente cît și toate cele 5 sau 6 semnificații ale unei taste.

IV. PROGRAMAREA ÎN BASIC, PE HC-85

8.4. Alfabetul limbajului BASIC

Prin acționarea tuturor tastelor calculatorului HC-85 și a combinațiilor posibile potrivit celor 5 moduri de lucru, se obțin toate cele 256 simboluri componente ale alfabetului BASIC.

Alfabetul limbajului BASIC se compune din simboluri simple și simboluri compuse.

Un simbol simplu este format dintr-un singur caracter obținut în urma unei singure tastări.

8. CARACTERISTICI BASIC

Un simbol compus este format dintr-un grup de caractere obținut (atenție!) în urma unei singure tastări.

Diferența dintre cele 2 tipuri de simboluri nu constă numai în numărul de caractere ci și în faptul că un simbol compus nu este echivalent cu simbolul obținut prin tastarea caracter cu caracter.

Din categoria simbolurilor simple fac parte literele, cifrele, semnele de punctuație, semnele operațiilor matematice etc.

Din categoria simbolurilor compuse fac parte cuvintele cheie ale instrucțiunilor sau comenzilor, funcțiile matematice etc.

Cu simbolurile simple se construiesc constante, variabile sau expresii, elemente atât de necesare definirii argumentelor instrucțiunilor și comenzilor BASIC. Cuvintele cheie alături de argumente potrivește generează setul de instrucțiuni și comenzi capabile să realizeze dialogul dintre utilizator și calculator.

Înainte de expunerea efectivă a instrucțiunilor și comenzilor este absolut necesară familiarizarea cu elementele ce au sens în limbajul BASIC.

8.4.1. Setul de caractere

Setul de caractere este format din totalitatea simbolurilor ce pot face parte dintr-o instrucțiune sau comandă în limbajul de programare BASIC.

Setul de caractere este format din:

- a) – litere;
- b) – cifre;
- c) – operatori aritmetici;
- d) – operatori de relație;
- e) – operatori logici;
- f) – semne de punctuație;
- g) – semne speciale.

a) Litere

Literele sînt în număr de 26 și sînt literele cunoscute, de la a la z, mici și mari.

b) Cifre

Cifrele sînt în număr de 10 și sînt cifrele sistemului zecimal, de la 0 la 9.

c) Operatori aritmetici

Operațiile aritmetice executate de calculator au următorii operatori:

- + pentru – adunare și semnul algebric + ;
- – scădere și semnul algebric – ;
- * – înmulțire;
- / – împărțire;
- ↑ – ridicare la putere.

Ordinea de execuție a operațiilor aritmetice este cea cunoscută din matematică:

- prioritatea 1 ridicarea la putere;
- prioritatea 2 înmulțirea și împărțirea;
- prioritatea 3 adunarea și scăderea.

Dacă există numai operații cu aceeași prioritate ordinea de efectuare este de la stînga la dreapta, inclusiv pentru ridicarea la putere.

EXEMPLE:

$$2 \uparrow 3 \uparrow 2 = 8 \uparrow 2 = 64$$

$$3 \uparrow 2 \uparrow 3 = 9 \uparrow 3 = 729$$

Ordinea de execuție poate fi schimbată cu ajutorul parantezelor.

d) Operatori de relație

Relațiile de ordine executate de calculator au următorii operatori de relație:

=	— pentru relația de	— egalitate;
<		— mai mic;
<=		— mai mic sau egal;
>		— mai mare;
>=		— mai mare sau egal;
<>		— diferit de.

Relațiile de ordine într-o mulțime numerică sînt relațiile de egalitate sau inegalitate cunoscute din matematică.

În mulțimea șirurilor de caractere, ca relație de ordine este folosită ordinea alfabetică cunoscută.

e) Operatori logici

Operațiile logice executate de calculator au următorii operatori:

AND	— pentru	— intersecție;
OR		— reuniune;
NOT		— negație.

Operațiile logice simbolizează operațiile din logica matematică obișnuită. Ordinea de execuție a operațiilor logice este cea cunoscută din matematică:

NOT
AND
OR

În expresii complicate ce conțin toate tipurile de operatori ordinea de execuție este următoarea:

- prioritatea 1 — operatori logici;
- prioritatea 2 — operatori de relație;
- prioritatea 3 — operatori aritmetici.

Dacă se dorește să se schimbe această ordine se folosesc parantezele.

Între operatori cu același nivel de prioritate, ordinea de execuție este de la stînga la dreapta.

f) Semne de punctuație

Dintre semnele de punctuație fac parte simbolurile cunoscute pentru:

.	—	punct
:	—	două puncte
;	—	punct și virgulă
,	—	virgulă
'	—	apostrof
"	—	ghilimele
!	—	semnul exclamării
?	—	semnul întrebării

g) Semne speciale

Dintre semnele speciale fac parte simbolurile cunoscute pentru:

—	spațiu liber, blanc (␣)
()	paranteze rotunde
[]	paranteze drepte
CR	retur de car
%	procent
\$	dolar
#	diez

Numărul semnelor speciale este mai mare decât cel specificat. Afirmatia se demonstrează, de exemplu, prin acționarea tastelor numerice de la 1 la 8, în modul de lucru G.

8.4.2. Constante

Constantele sînt valori fixe. Ele sînt de tip numeric sau alfanumeric.



Fig. 8.4.

1°. Constante numerice

Constantele numerice sînt șiruri de cifre negative sau pozitive.

În limbajul BASIC constantele numerice se pot exprima în sistemele de numerație:

- zecimal (numărul este reprezentat cu ajutorul cifrelor de la 0 la 9);
- binar (numărul este reprezentat cu ajutorul cifrelor 0 și 1);
- hexazecimal (numărul este reprezentat cu ajutorul cifrelor de la 0 la 9 și a literelor A, B, C, D, E, F cu semnificația A=10, B=11, C=12, D=13, E=14, F=15).

În cazul particular în care constanta este scrisă în sistemul hexazecimal, s-a stabilit convenția de a fi completată în partea dreaptă cu simbolul &.

Constantele numerice la rîndul lor pot fi, potrivit numerelor pe care le conțin, de două tipuri:

- constante de tip întreg;
- constante de tip real.

Constantele de tip întreg sînt valori întregi.

EXEMPLE: —10
0
1987

Constantele de tip real sînt valori reale în care virgula se reprezintă prin punct.

EXEMPLE:

19.88
0.1988
-0.1988

Reprezentarea constantelor în exemplele anterioare a fost făcută în scriere obișnuită. Ea mai poartă și numele de scriere în virgulă fixă.

Unele probleme practice și științifice impun folosirea unor numere foarte mari sau foarte mici. De exemplu:

- a) distanța de la Pământ la Lună este de 384.000.000 m;
- b) distanța de la Pământ la Soare este de 150.000.000.000 m;
- c) un gram de hidrogen conține un număr foarte mare de atomi de hidrogen și anume:
6023 00000000000000000000, deci 6023 urmat de 20 de zerouri;
- d) un gram de apă conține
33 00000000000000000000 molecule de apă, deci 33 urmat de 21 de zerouri;
- e) masa unei molecule de apă este de
3 : 1000000000000000000000000 kg, deci 3 supra 1 urmat de 26 de zerouri.

După cum se vede este foarte incomod de a scrie și de a lucra cu aceste numere. Pe de altă parte numerele scrise în acest mod ocupă un loc foarte mare în memoria calculatorului. Considerentele specificate au condus la necesitatea găsirii unui alt mod de scriere pentru utilizarea mai comodă a acestor numere.

Acest mod de scriere folosește puterile lui 10, astfel că numerele se pot scrie astfel:

- a) $384 * 10^6$ m;
- b) $15 * 10^{10}$ sau $1,5 * 10^{11}$ m;
- c) $6023 * 10^{20}$ sau $6,023 * 10^{23}$ atomi, (numărul scris sub ultima formă și notat cu N se utilizează în chimie sub denumirea de numărul lui Avogadro);
- d) $33 * 10^{21}$ sau $0,33 * 10^{23}$ molecule de apă;
- e) $3 * 10^{-26}$ kg, notînd $1/10$ cu 10^{-1} .

Această scriere comodă pentru manipulare, dar și economică din punct de vedere al reprezentării numerelor în memorie, se numește scriere cu exponent (pe scurt scriere cu e).

Orice număr se poate reprezenta ca produsul dintre un număr și puteri ale lui 10.

Pentru un număr există mai multe variante de a fi reprodus cu ajutorul puterilor lui 10 și de aceea scrierea cu e se numește și scriere în virgulă mobilă.

În reprezentarea cu e a numerelor, 10 nu apare explicit.

Numerele din exemplele date se reprezintă cu ajutorul scrierii cu e astfel:

- a) 384e6 sau 38.4e7 sau 3.84e8 sau 0.384e9;
- b) 15e10 sau 1.5e11 sau 0.15e12;
- c) 6023e20 sau 6.023e23 sau;
- d) 33e21 sau 0.33e23;
- e) 3e-26 sau 0.3e-27.



Fig. 8.5.

Cel mai mare număr ce poate fi reprezentat în memoria calculatorului este numărul:

$$4294967295 = 2^{32} - 1$$

s. CARACTERISTICI BASIC

2° Constante alfanumerice

Constantele de tip alfanumeric sînt șiruri de caractere. Prin șir de caractere se înțelege un șir format din orice caracter prezent pe tastatura calculatorului încadrat între ghilimele.

Dacă se dorește să se tipărească în text caracterul ghilimele atunci el trebuie dublat.

EXEMPLE de constante alfanumerice:


```
"abc"  
"calculator"  
"linie + coloana"  
"CEL MAI MARE NUMĂR"  
"A 1 + A 2"  
"234"
```

8.4.3. Variabile

Variabila este un nume (o etichetă) care se dă unui număr sau unui șir de caractere. Numele este ales de utilizator. El este format dintr-o succesiune de litere și / sau cifre căruia i se atribuie în diferite momente ale execuției unui program, diferite valori.

Variabilele sînt ca și constantele de două tipuri: numerice sau alfanumerice.

1° Variabile numerice



Versiunea BASIC a calculatorului HC-85 admite pentru variabilele numerice nume formate din oricîte caractere (litere sau cifre), care încep cu o literă. Printre caractere poate fi și blankul, care este însă ignorat. Prezența lui face

variabila mai ușor de citit. Sistemul face filtrarea literelor mari, astfel încît, atît litera mare cît și litera mică corespunzătoare sînt interpretate la fel.

Este recomandat ca numele unei variabile să fie sugestiv fără însă a crea confuzii și cît se poate de scurt pentru a fi ușor de manipulat.

Variabilele numerice sînt simple sau indexate.

Variabila numerică simplă este un nume căruia i se atribuie la un moment dat o valoare număr real.

Fig. 8.6.



Fig. 8.7.

EXEMPLE de variabile numerice:

- a) R — în accepțiunea unui program ea poate reprezenta raza unui cerc;
— poate lua pe rînd valorile:
0
10.3
28.5702
88

b) LAT — în accepțiunea unui program ea poate reprezenta latura unui pătrat;

— poate lua pe rând valorile:

0.5
1
7.88
65

Variabila numerică indexată este un nume căruia i se atribuie la un moment dat un șir finit de valori numerice. Numărul valorilor din șir este dat de indicii variabilei.

Un indice este la rândul său o variabilă numerică. El poate lua valori numere naturale, de la 1, 2, 3. ... pînă la 100 sau mai mare, dar în practică nu este nevoie de un număr așa de mare de indici.

EXEMPLE de variabile numerice indexate:

a) D(I) — în accepțiunea unui program conține cele două dimensiuni ale unui dreptunghi;

I=1,2

— poate lua pe rând valorile:

D(1) = 15 (lungimea)
D(2) = 3 (lățimea)

sau

D(1) = 7.5
D(2) = 2.3

b) s(k) — în accepțiunea unui program conține un șir de 5 numere naturale;

k = 1, ..., 5

— poate lua pe rând valorile:

s (1) = 1 (primul element al șirului)
s (2) = 5 (al doilea element al șirului)
s (3) = 0 (al treilea element al șirului)
s (4) = 12 (al patrulea element al șirului)
s (5) = 2 (al cincilea element al șirului)

sau

s (1) = 18
s (2) = 1
s (3) = 1
s (4) = 1
s (5) = 0

c) M (I, J) — în accepțiunea unui program este un tablou cu 2 linii și 3 coloane ce conține numere între 0 și 1;

I = 1, 2

J = 1, 2, 3

— poate lua pe rând valorile:

	coloana 1	coloana 2	coloana 3
linia 1	M(1,1) = 0.1	M(1,2) = 0.01	M(1,3) = 0.0001
linia 2	M(2,1) = 1	M(2,2) = 0.7	M(2,3) = 0.4

sau

linia 1	M(1,1) = 1	M(1,2) = 1	M(1,3) = 1
linia 2	M(2,1) = 0	M(2,2) = 1	M(2,3) = 0

Este util de reținut că o variabilă numerică simplă poate avea același nume cu o variabilă numerică indexată fără a se genera confuzii în bunul mers al unui program.

2° Variabile alfanumerice

Versiunea BASIC a calculatorului HC-85 admite pentru variabilele alfanumerice nume formate din oricâte caractere (litere sau cifre), ce încep cu o literă și se termină cu caracterul \$.

Variabilele alfanumerice sînt simple sau indexate.

Variabila alfanumerică simplă este un nume căreia i se atribuie la un moment dat o valoare șir de caractere.

Fig. 8.8.

EXEMPLE de variabile alfanumerice:

- a) a\$ — poate lua pe rînd valorile:
ABC
C2M
???
- b) TRA-1 \$ — poate lua pe rînd valorile:
5
DA
-.2

Variabila alfanumerică indexată este un nume format dintr-un singur caracter căruia i se atribuie la un moment dat un șir finit de valori șir de caractere.

Indicii trebuie să îndeplinească aceleași condiții ca cei ai unei variabile numerice indexate.

EXEMPLE de variabile alfanumerice indexate:

- a) T\$(P) — în accepțiunea unui program este un șir de caractere;
P = 1,2,3 — poate lua pe rînd valorile:
T\$ (1) = "ABC" (primul șir al șirului)
T\$ (2) = "DEF" (al doilea șir al șirului)
T\$ (3) = "GHI" (al treilea șir al șirului)
sau
T\$ (1) = "1-A"
T\$ (2) = "2-B"
T\$ (3) = "3-B"
- b) m (u, v) — în accepțiunea unui program este un tablou cu 2 linii și 3 coloane de șiruri de caractere;
u = 1,2,3,4
v = 1,2 — poate lua pe rînd valorile:
- | | coloana 1 | coloana 2 |
|---------|------------------------------|-------------------------|
| linia 1 | m\$ (1,1) = "IONESCU DAN" | m\$ (1,2) = "BUCURESTI" |
| linia 2 | m\$ (2,1) = "IONESCU ION" | m\$ (2,2) = "CONSTANTA" |
| linia 3 | m\$ (3,1) = "IONESCU MIRCEA" | m\$ (3,2) = "IASI" |
| linia 4 | m\$ (4,1) = "IONESCU VLAD" | m\$ (4,2) = "BRASOV" |
- sau
- | | | |
|---------|-----------------------------|-----------------------|
| linia 1 | m\$ (1,1) = "ADAM STELA" | m\$ (1,2) = "LIC. 1" |
| linia 2 | m\$ (2,1) = "BOBESCU RADU" | m\$ (2,2) = "LIC. 21" |
| linia 3 | m\$ (3,1) = "COSTIN MARCEL" | m\$ (3,2) = "LIC. 18" |
| linia 4 | m\$ (4,1) = "DORIN ANA" | m\$ (4,2) = "LIC. 5" |

8.4.4. Expresii

Expresia este un grup de simboluri unite printr-o operație ce poate fi evaluat. Expresiile pot fi numerice sau alfanumerice, după tipul simbolurilor din grup.



Fig. 8.9.



Fig. 8.10.

1° Expresii numerice

Expresiile numerice sînt compuse din constante numerice, variabile simple, variabile indexate sau combinații ale acestora separate prin operatori aritmetici, operatori de relație sau operatori logici.

Expresia aritmetică este expresia numerică care conține operatorii aritmetici cunoscuți : +, -, *, /, ↑

EXEMPLE de expresii aritmetice:

A + B
2 ↑ C
5 ((A + 3/C) - B * D)

Expresia de relație este expresia numerică care conține operatorii de relație cunoscuți : =, <, <=, >, >=, <>.

Valoarea unei expresii de relație este o valoare logică de ADEVĂRAT sau FALS.

EXEMPLE de expresii de relație:

A = B
A < > B
A < = C > = D < > REF

Expresia logică este expresia numerică care conține operatorii logici: NOT, AND, OR.

Operatorii logici într-o expresie cu 2 operanzi au următorul efect:

- A AND B — dacă este îndeplinită condiția A și condiția B se continuă pe ramura DA (sau ADEVĂRAT);
— dacă una din condițiile A și B nu este îndeplinită se continuă pe ramura NU (sau FALS).
- A OR B — dacă este îndeplinită condiția A și condiția B se continuă pe ramura DA;
— dacă este îndeplinită condiția A, dar nu este îndeplinită condiția B se continuă pe ramura DA;
— dacă nu este îndeplinită condiția A, dar este îndeplinită condiția B se continuă pe ramura DA;

- dacă nu este îndeplinită nici condiția A nici condiția B se continuă pe ramura NU.
- NOT A — dacă nu este îndeplinită condiția A se continuă pe ramura DA;
- dacă este îndeplinită condiția A se continuă pe ramura NU.

2° Expresii alfanumerice

Expresiile alfanumerice sînt compuse din constante alfanumerice, variabile simple sau indexate de tip șir de caractere sau combinații ale acestora, separate prin operatori ca:

- operatori de concatenare;
- operatori de relație.

Operatorul de concatenare, marcat prin semnul „+”, alipește două șiruri de caractere.

EXEMPLE de concatenare:

- a) "ABC"+"DEFGH" produce șirul "ABCDEFGH"
- b) "A1A2"+"100" produce șirul "A1A2100"

Operatorii de relație în cazul expresiilor alfanumerice operează asupra șirurilor de caractere cu respectarea ordinii alfabetice. Compararea a două șiruri de caractere se face caracter cu caracter de la stînga spre dreapta.

Operatorii de relație permiși sînt cei cunoscuți: =; <; <=; >; >=; <>.

8.4.5. Funcții

Funcția în accepțiunea limbajului BASIC are aceeași semnificație ca în matematică: unui element dintr-o mulțime i se asociază un element din altă mulțime, respectîndu-se o anumită regulă.

Funcțiile sînt de 2 feluri, potrivit valorilor pe care le manipulează:

- funcții matematice și
- funcții pe șir de caractere.

Funcțiile prezentate în continuare sînt funcții standard adică au un mnemonic (nume) predefinit. Ele sînt apelate prin acest mnemonic (nume) și se introduc într-o linie program printr-o singură acționare a tastei pe care sînt specificate.

1° Funcții matematice

Funcția matematică este funcția ce lucrează cu valori numerice și corespunde unei funcții matematice cunoscute.

Calculatorul HC-85 dispune de următoarele funcții matematice:

- SQR x rădăcina pătrată din x (radical din x, iar $x \geq 0$);
- SIN x sinus de x (x în radiani);
- COS x cosinus de x (x în radiani);
- TAN x tangenta de x (x în radiani);

ASN	x	arcsinusul lui x ;
AGS	x	arccosinusul lui x ;
ATN	x	arctangenta a lui x ;
EXP	x	exponențiala lui x (e^x , unde $e=2,71\dots$);
LN	x	logaritmul natural din x (se poate utiliza la calculul unui logaritm în orice bază folosind formula $\text{LOG}_a x = \frac{\text{LN}x}{\text{LN}a}$, iar $x > 0$);
ABS	x	valoarea absolută a lui x ($ x $);
INT	x	partea întreagă a lui x ($[x]$);
SGN	x	funcția furnizează semnul lui x și are valorile:
		1 pentru $x > 0$
		0 pentru $x = 0$
		-1 pentru $x < 0$

2° Funcții pe șir de caractere

Funcția pe șir de caractere corespunde funcției care se aplică unui șir de caractere și furnizează un număr sau invers, se aplică unui număr și se obține un șir de caractere.

Calculatorul HC-85 dispune de următoarele funcții pe șir de caractere:

LEN	s\$	se aplică șirului de caractere s\$ și generează un număr ce reprezintă lungimea șirului de caractere;
CODE	s\$	se aplică șirului de caractere s\$ și întoarce codul primului caracter din șir (dacă șirul este vid, transmite 0);
VAL	s\$	se aplică șirului de caractere s\$ și convertește șirul în numere;
VAL	\$. s\$	se aplică șirului de caractere s\$ și convertește șirul în alt șir de caractere (este rar folosită);
CHR	n	se aplică numărului n și generează caracterul ce are codul n ;
STR	n	se aplică numărului n și convertește numărul în șir de caractere.

Calculatorul poate fi folosit numai prin intermediul instrucțiunilor pentru care a fost proiectat.

O instrucțiune este alcătuită dintr-un „cuvânt cheie“ și unul sau mai multe argumente:

<cuvânt cheie> <argument 1>, <argument 2> . . .

Cuvântul cheie este mnemonicul unui cuvânt din limba engleză ce definește numele instrucțiunii. Cuvântul cheie este reprezentativ și sugestiv.

Argumentul unei instrucțiuni este o constantă, o expresie, o condiție, un mesaj, un șir de caractere, o funcție sau o combinație a acestora. Toate tipurile de argumente trebuie să respecte regulile din capitolele precedente.

Argumentele unei instrucțiuni sînt delimitate de următorii separatori:

- virgulă;
- punct și virgulă;
- apostrof.

Limbajul BASIC admite două tipuri de instrucțiuni:

- nenumerate și
- numerotate.

Instrucțiunea nenumerotată se numește și *comandă*.

Comanda se execută imediat după apăsarea tastei CR (ultima tastă din rândul 3), după care nu mai are nici un efect.

Instrucțiunea numerotată se numește simplu, *instrucțiune*.

Instrucțiunea nu se execută imediat, ci se stochează ca linie program și se poate executa printr-o comandă RUN ori de cîte ori se dorește acest lucru.

Tastarea lui CR, atît la terminarea unei comenzi cît și la terminarea unei linii program este obligatorie. Ea este manevra prin care datele tastate sînt în-cărcate în memorie și poate fi asimilată cu o „poartă de intrare“.

O linie program poate conține una sau mai multe instrucțiuni; separarea instrucțiunilor dintr-o linie se face prin caracterul „:“ (două puncte).

Numerele date liniilor trebuie să fie întregi și cuprinse între 1 și 9999.

Totalitatea liniilor program ce reprezintă transpunerea unui algoritm în vederea rezolvării unei probleme cu ajutorul calculatorului se numește *program*.

Listarea și execuția unui program se face în ordinea naturală a numerelor atribuite liniilor program componente. De aceea, este indicat ca la scrierea unui program să se numereze liniile din 10 în 10, dînd astfel posibilitatea inserării cu ușurință de linii noi.

Cursorul „>” indică linia curentă asupra căreia se pot face modificări sau după care se pot insera alte linii. De obicei, cursorul se află pe ultima linie introdusă, dar el poate fi deplasat în sus sau în jos prin apăsarea simultană a tastei CAPS SHIFT și a săgeților.

O instrucțiune nenumerotată se deosebește formal față de o instrucțiune numerotată prin inexistența numărului de linie. Deosebirea esențială constă însă în modul de execuție: prima se execută o singură dată și nu mai poate fi reluată, a doua de câte ori este nevoie.

Atât instrucțiunile nenumerotate cât și instrucțiunile numerotate se supun unui grup comun de reguli, drept pentru care nu se va face nici o diferențiere în prezentarea lor.

În cazul particular în care o anumită instrucțiune este utilizată sub formă de comandă se va specifica expres acest lucru.

S-a preferat abordarea instrucțiunilor în ordinea ce urmează, pentru a oferi posibilitatea de a se trece într-un timp cât mai scurt la conceperea de programe în limbajul de programare BASIC.

De asemenea, s-a ținut permanent cont de alternarea prezentării instrucțiunilor simple sau atractive în raport cu prezentarea instrucțiunilor dificile, pentru a se doza efortul de înțelegere a limbajului BASIC. Ultimele instrucțiuni prezentate nu sînt cele mai neutilizate ci sînt cele care au sens într-o anumită dotare (monitor color, imprimantă).

Trebuie subliniat că exemplele alese pentru lămurirea și folosirea instrucțiunilor sînt intenționat simple. Ele scot în evidență utilizarea standard, urmărind familiarizarea eșalonată cu noul mod de a privi lucrurile prin intermediul unui limbaj de programare. Din loc în loc, acolo unde este oportun, se dau și câteva sfaturi practice generale, a căror respectare aduce un plus de ușurare în efortul de a stăpîni calculatorul în momentul elaborării unui program.

9.1. Instrucțiunea PRINT

Instrucțiunea PRINT este utilizată pentru editarea (scrierea) pe ecran a rezultatelor intermediare sau finale ale unei execuții.

Formatul general este:

PRINT listă

unde „listă” este o constantă, o variabilă, o expresie, un șir de caractere separate prin virgulă, punct și virgulă sau apostrof.

EXEMPLE:

1. Tipărirea numărului 12 se face prin comanda:

PRINT 12

2. Tipărirea șirului de caractere "SINT ELEV" se face prin comanda:

PRINT "SINT ELEV"

Trebuie reținut că la terminarea introducerii fiecărei comenzi este obligatorie tastarea lui CR. Această manevră provoacă execuția comenzii de către calculator.

Pe ecran apare:



Fig. 9.1.

Cu ajutorul comenzii PRINT se fac calcule matematice imediate oricât de complicate ar fi, într-un timp record. Aceasta este una din modalitățile de efectuare a calculelor folosind calculatorul personal.

Limbajul BASIC oferă și alte posibilități pentru efectuarea de calcule, dar și pentru multe alte lucruri ce așteaptă numai să fie încercate și / sau descoperite.

EXEMPLE de calcule imediate:

1. Adunarea dintre 2 și 0.2 se realizează prin comanda:

PRINT 2+0.2



Ecranul înainte de CR



Ecranul după CR

Fig. 9.2.

2. Împărțirea dintre 41.6 și 0.80 se realizează prin comanda:

PRINT 41.6/0.80



Ecranul înainte de CR



Ecranul după CR

Fig. 9.3.

3. Calculul expresiei $\frac{5 * 6 - 3 * 4 + 9}{3}$ se realizează prin comanda:

PRINT (5 * 6 - 3 * 4 + 9) / 3



Ecranul înainte de CR



Ecranul după CR

Fig. 9.4.

Comanda PRINT devine instrucțiunea PRINT dacă linia primește un număr de ordine și în acest caz trebuie tastat CR la terminarea introducerii tuturor argumentelor instrucțiunii (dacă linia program conține numai o singură instrucțiune). Altfel spus, tastarea lui CR marchează terminarea oricărei linii program, care este astfel memorată și poate fi executată în urma unei comenzi RUN (execută).

Calculule din exemplele anterioare se pot efectua și prin instrucțiunea PRINT ajungându-se astfel la scrierea celor mai simple programe.

Program 1

```
10 PRINT 2 + 0.2
RUN
```



Ecranul după PRINT urmat de CR



Ecranul după RUN urmat de CR

Fig. 9.5.

Program 2

```
10 PRINT 41.6/0.8
RUN
```



Ecranul după PRINT urmat de CR



Ecranul după RUN urmat de CR

Fig. 9.6.

Program 3

```
10 PRINT (5 * 6 - 3 * 4 + 9) / 3
RUN
```



Ecranul după PRINT urmat de CR



Ecranul după RUN urmat de CR

Fig. 9.7.

Acestea sînt cele mai simple programe nu numai pentru că sînt formate dintr-o singură linie program ci și pentru că și-au propus spre rezolvare o singură problemă, și ea simplă de fapt. De obicei, un program este alcătuit din mai multe linii program. De exemplu, toate calculele anterioare pot fi grupate într-un singur program de forma următoare:

```
10 PRINT 2+0.2
20 PRINT 41.6/0.8
30 PRINT (5 * 6 - 3 * 4 + 9)/3
RUN
```

Pe ecran rezultatele sînt acum afișate ca în figura 9.8.



Fig. 9.8.

Următoarele observații rămîn valabile pentru toate instrucțiunile:

- o linie program se consideră terminată numai după ce s-a tastat CR;
- între numărul de ordine, cuvîntul cheie al instrucțiunii și argumentele sale nu este nevoie să se lase spațiu (blanc), operația fiind făcută automat de calculator.

Programul anterior poate fi completat și cu alte calcule, ca de exemplu împărțirea lui 7 la 2.

Pentru aceasta se introduce linia program:

```
25 PRINT 7/2
```

Respectînd convenția de tastare a lui CR după fiecare linie program, se observă că ordonarea liniilor se face automat, iar noul program arată astfel:



Fig. 9.9.

```
10 PRINT 2+0.2
20 PRINT 41.6/0.8
25 PRINT 7/2
30 PRINT (5 * 6 - 3 * 4 + 9)/3
```

Comanda RUN urmată de tastarea lui CR face posibilă execuția programului ce afișează pe ecran numerele din fig. 9.9.

În concluzie, orice program poate fi îmbogățit cu alte linii program tastînd consecutiv, după ultima linie a lui, noile linii. Numărul de ordine al unei noi linii introduse poate fi mai mic sau mai mare decît ultimul număr de ordine, potrivit locului în care se dorește inserarea. Ordonarea efectivă a liniilor program este făcută automat de către calculator.

Din observația anterioară se deduce modul prin care un program poate fi modificat pe parcursul exploatării sale. Pentru ca orice modificare să se facă cu un minim de efort, se recomandă ca numerele de ordine a liniilor unui program, în prima etapă a validării pe calculator, să respecte următoarele reguli:

- numerotarea să nu se înceapă de la 1 ci de la 10;
- numerotarea să nu se facă din 1 în 1 ci din 10 în 10.

O situație des întâlnită constă în renunțarea la o linie program.

O linie program este ștearsă prin simpla tastare a numărului său de ordine urmat imediat de tastarea lui CR.

Din programul:

```
10 PRINT 2+0.2
20 PRINT 41.6/0.8
25 PRINT 7/2
30 PRINT (5*6-3*4+9)/3
```

dispare împărțirea lui 41.6 cu 0.80 prin introducerea liniei:

```
20 <CR>
```

Între argumentele instrucțiunii PRINT se găsesc diverși separatori. Folosirea acestora are următorul efect:

a) utilizarea caracterului virgulă determină începerea tipăririi fie pe marginea din stînga, fie în mijlocul ecranului, în funcție de ce urmează după virgulă;

b) utilizarea caracterului punct și virgulă determină tipărirea șirului imediat următor după șirul precedent;

c) utilizarea caracterului apostrof determină saltul cursorului la începutul liniei următoare și continuarea tipăririi din acel punct ca și cum elementele despărțite prin caracterul apostrof ar fi fost sub incidența unor instrucțiuni PRINT succesive.

Pentru ca instrucțiunea PRINT să nu determine saltul la linia următoare este necesar ca instrucțiunea PRINT precedentă să se termine cu caracterele virgulă sau punct și virgulă.

Se constată că varianta cu virgulă împarte totul în două coloane, cea cu punct și virgulă scrie totul compact, iar cea cu apostrof scrie un număr pe o linie.

Dacă după argumentele lui PRINT nu se pune nici un semn de punctuație atunci fiecare număr se scrie pe o altă linie.

Dacă se dorește să apară pe ecran o linie fără conținut (o linie liberă) se folosește instrucțiunea PRINT fără nici un argument.



Fig. 9.10.

EXAMPLE:

Program 1

```
10 PRINT "HC-85 are 5 moduri de lucru:";
20 PRINT "1-K, 2-L, 3-C, 4-E, 5-G"
```

afișează pe ecran rezultatele ca în fig. 9.10.

Program 2

```
10 PRINT "HC-85 are 5 moduri de lucru:";
10 PRINT "1-K,";
30 PRINT "2-L,";
40 PRINT "3-C,";
```

```

50 PRINT "4-E.";
60 PRINT "5-G";

```

afișează pe ecran rezultatele ca și programul 1.

Program 3

Programul 3 este conceput pe structura programului 2 modificând liniile 10 –60 prin ștergerea caracterului virgula și a caracterului punct și virgula.

```

10 PRINT "HC-85 are 5 moduri de lucru:"
20 PRINT "1-K"
30 PRINT "2-L"
40 PRINT "3-C"
50 PRINT "4-E"
60 PRINT "5-G"

```

Pe ecran rezultatele programului 3 sînt afișate ca în fig. 9.11

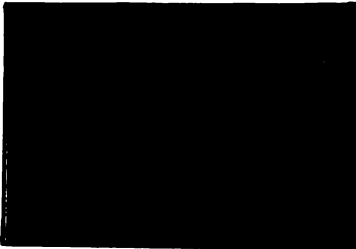


Fig. 9.11.

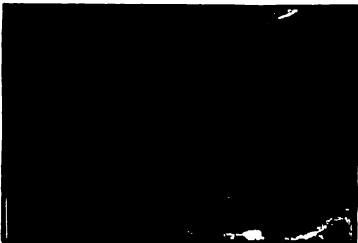


Fig. 9.12.



Fig. 9.13.

Program 4

```

10 PRINT "HC-85 are 5 moduri de lucru:"
20 PRINT "1-K'"
30 PRINT "2-L'"
40 PRINT "3-C'"
50 PRINT "4-E'"
60 PRINT "5-G'"

```

afișează pe ecran rezultatele ca și programul 3 deși liniile 20 –60 sînt modificate față de liniile programului 3 prin adăugarea la extremitatea dreaptă a caracterului apostrof.

Program 5

```

10 PRINT "HC-85 are 5 moduri de lucru:"
20 PRINT "1-K",
30 PRINT "2-L",
40 PRINT "3-C",
50 PRINT "4-E",
60 PRINT "5-G"

```

afișează pe ecran rezultatele ca în fig. 9.12.

Program 6

Programul 6 este identic cu programul 5 cu excepția liniei 40 din care dispăre virgula.

```

10 PRINT "HC-85 are 5 moduri de lucru:"
20 PRINT "1-K",
30 PRINT "2-L",
40 PRINT "3-C"
50 PRINT "4-E",
60 PRINT "5-G"

```

Pe ecran rezultatele programului 6 sînt afișate ca în fig. 9.13.

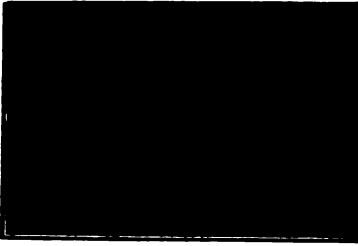


Fig. 9.14.

Program 7

Programul 7 este programul 6 cu linia 40 modificată prin adăugare la sfârșit a semnului punct și virgulă.

```
10 PRINT "HC-85 are 5 moduri de lucru:"
20 PRINT "1-K",
30 PRINT "2-L",
40 PRINT "3-C";
50 PRINT "4-E",
60 PRINT "5-G"
```

Pe ecran rezultatele programului 7 sînt afișate ca în fig. 9.14.

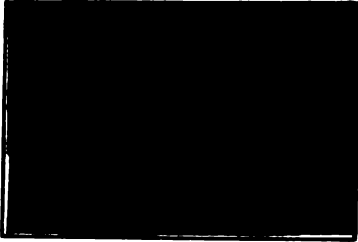


Fig. 9.15.

Program 8

```
10 PRINT "HC-85 au 5 moduri de lucru:"
20 PRINT "1-K"
30 PRINT "2-L",
40 PRINT "3-C";
50 PRINT "4-E",
60 PRINT "5-G"
```

afișează pe ecran rezultatele ca în fig. 9.25

9.2. Instrucțiunea RUN

Din exemplele anterioare se observă că pentru execuția unui program trebuie utilizată comanda RUN.

RUN este des folosită sub forma de comandă.

Formatul general este:

RUN nr. linie

unde

a) dacă „nr. linie“ este specificat, execuția programului cuprinde liniile începînd de la numărul de ordine „nr. linie“ pînă la ultima linie program introdusă;

b) dacă „nr. linie“ nu este specificat execuția programului cuprinde în întregime liniile introduse.

Trebuie reținut că RUN poate fi foarte bine și instrucțiune. Programul care conține o instrucțiune RUN se execută numai după un RUN, care de această dată este comandă. Execuția unui astfel de program durează la nesfârșit. Oprirea se realizează prin tastarea lui BREAK, care înseamnă „înterupe“ și se obține prin acționarea simultană a tastelor CS și SPACE.

EXEMPLU:

Programul

```
10 PRINT "a"
```



Fig. 9.16.

20 PRINT "b"

30 PRINT "c"

afișează pe verticală grupul de litere a, b, c ca în fig. 9.16.

Prin completarea programului anterior cu linia

40 RUN

se obține afișarea continuă a grupului de litere a, b, c (ca o pîlpiire), iar oprirea execuției se realizează prin tastarea lui BREAK.

Pe ecran apare mesajul:

L BREAK into program, 40: 1

Dacă se dorește o nouă afișare, se tastează RUN urmat de CR și se efectuează o nouă execuție a programului.

Execuția unui program, fie că are sau nu o linie program ce-l conține pe RUN, se obține prin comanda RUN.

9.3. Instrucțiunea DELETE

De câte ori este nevoie să se ștergă un simbol se utilizează DELETE. Ea este folosită sub formă de comandă.

Formatul general este:

DELETE

Se impun următoarele precizări:

a) dacă se tastează DELETE (acționarea simultană a tastelor CS și 0) o singură dată atunci se produce ștergerea ultimului simbol introdus;

b) dacă se tastează DELETE de mai multe ori atunci se produce ștergerea unui număr de simboluri egal cu numărul tastărilor efectuate, începînd cu ultimul simbol introdus;

c) introducerea de la tastatură a comenzii DELETE nu se vizualizează pe ecran sau altfel spus numele comenzii DELETE nu se afișează pe ecran ci numai efectul ei (ștergerea efectivă și pe rînd a simbolurilor de la dreapta la stînga);

d) cu DELETE se efectuează ștergerea unor simboluri în cadrul liniei program curente (linia pentru care nu s-a tastat încă CR);

e) în cazul în care se dorește să se renunțe (din anumite considerente) la o linie program deja introdusă se va folosi procedeul de suprapunere al unei linii vide peste respectiva linie (linia formată numai din numărul de ordine al liniei ce trebuie eliminată, terminată brusc prin tastarea lui CR) și nu se va recurge la utilizarea lui DELETE.

9.4. Instrucțiunea GO TO

Instrucțiunea GO TO servește la modificarea ordinii secvenței de execuție a liniilor unui program. Această instrucțiune realizează un salt necondiționat la numărul de linie specificat.

Formatul general este:

GO TO nr linie

unde „nr. linie“ este numărul liniei program de la care se va începe reluarea.

Modificarea programului de la pct. 9.2 prin înlocuirea liniei 40 cu o nouă linie:

40 GO TO 10

face ca programul să se reia mereu de la început. Ca rezultat ecranul se umple pe verticală cu literele a, b și c.

În momentul când ecranul s-a umplut în partea de jos a lui apare mesajul:

scroll ?

(în limba engleză scroll înseamnă „sul de hirtie“).

Dacă se apasă pe orice tastă înseamnă că se dorește „scroll“, echivalent cu continuarea afișării.

În acest moment ultimul rînd tipărit ia locul penultimului rînd, penultimul rînd ia locul antepenultimului rînd ș.a.m.d, astfel că primul rînd tipărit iese din ecran, iar pe ultimul rînd de acum (rămas liber) se poate tipări.

Înlocuirea unui rînd cu alt rînd, de jos în sus, se produce rapid și rularea afișării șirului de numere seamănă cu o „defilare“. Cînd ecranul s-a umplut, apare din nou mesajul:

scroll ?

Dacă se apasă pe tasta N (N de la nu) apare mesajul:

D BREAK — CONT repeats

și afișarea este oprită.

De fapt răspunzîndu-se prin „nu“ s-a produs și oprirea execuției programului.

Trebuie specificat clar că, GO TO are efect de lansare în execuție a unui program (așa cum se vede de fapt și din exemplul ales) la fel ca RUN. Există însă între RUN și GO TO o deosebire esențială:

— RUN șterge valorile tuturor variabilelor rămase dintr-o execuție anterioară;

— GO TO păstrează (nu schimbă) aceste valori.

9.5. Instrucțiunea CLS

Instrucțiunea CLS provoacă ștergerea ecranului.

Formatul general este:

9. INSTRUCȚIUNI BASIC

CLS

(fără nici un argument).

Programul de la pct 9.4 diferă de programul de la pct. 9.2 prin conținutul liniei 40. Cu ajutorul instrucțiunii CLS programul de la pct. 9.4 poate avea același efect cu cel de la pct. 9.2. prin:

— introducerea unei noi linii

```
5 CLS
```

— modificarea liniei 40

```
40 GO TO 5
```

Programul arată acum astfel:

```
5   CLS
10  PRINT "a"
20  PRINT "b"
30  PRINT "c"
40  GO TO 5
```

și afișează continuu pe verticală grupul de litere a, b și c.

Ștergerea ecranului se realizează și cu ajutorul comenzilor RUN și CLEAR (care au și alte funcții), dar numai instrucțiunea CLS poate șterge ecranul în anumite momente ale execuției unui program.

Instrucțiunea CLS șterge ecranul, dar nu șterge și programul din memoria calculatorului.

9.6. Instrucțiunea NEW

Instrucțiunea NEW șterge din memorie programul curent pentru a face posibilă introducerea unui nou program.

NEW este folosită în majoritatea cazurilor sub formă de comandă.

Formatul general este:

NEW

(fără nici un argument).

9.7. Instrucțiunea LIST

Instrucțiunea LIST servește la extragerea pe ecranul televizorului a unei părți sau a întregului program curent (existent în momentul acela în memoria internă).

LIST este des folosită sub formă de comandă.

Formatul general este:

LIST nr. linie

unde

a) dacă „nr. linie“ este specificat, se produce listarea programului de la linia program cu acel număr de ordine pînă la linia curentă (ultima linie program introdusă) care poate fi și sfîrșitul programului;

b) dacă „nr. linie“ nu este specificat, se produce listarea programului de la început (prima linie program introdusă) pînă la linia curentă.

De reținut este faptul că linia curentă (cea cu $>$) apare întotdeauna pe ecran, în mod normal în poziție centrală. Calculatorul memorează numărul liniei curente precum și numărul primei linii din partea de sus a ecranului.

Cînd încearcă să listeze, primul lucru pe care-l face este să compare prima linie de pe ecran cu linia curentă. Dacă prima linie de pe ecran este mai mare decît linia curentă, atunci cursorul va apare pe prima linie a ecranului. Altfel, listarea constă în tipărirea pe ecran în mod defilare a programului cuprins între prima linie și linia curentă.

Oricum, mai întîi se efectuează un calcul aproximativ pentru a vedea cît timp ia listarea și dacă aceasta este prea lungă, linia din vîrf se mută mai jos pentru a fi mai aproape de linia curentă. Acum avînd stabilită linia din vîrf, listarea poate începe. Dacă linia curentă a fost listată, listarea se oprește cînd s-a ajuns la sfîrșitul programului sau în partea de jos a ecranului.

Listarea unui program, după execuția sa, se face și prin simpla tastare a lui CR.

9.8. Instrucțiunea PRINT AT

Ecranul este împărțit în 24 linii de afișare, fiecare cu 32 de caractere.

Ecranul are două părți. Partea de sus de 22 linii este folosită pentru listarea liniilor programului sau a rezultatelor. Cele două linii de jos se folosesc pentru introduceri curente de date.

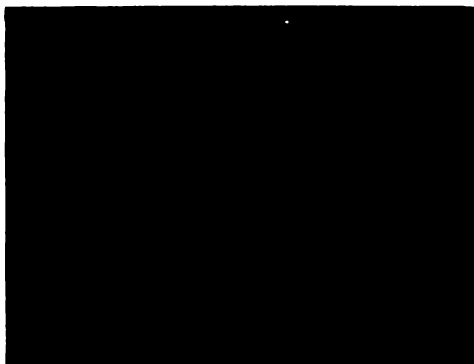


Fig. 9.17.

În acest fel, ecranul are disponibile pentru utilizator: 22 de linii și 32 coloane. Liniile sînt numerotate de la 0 la 21 (de sus în jos), iar coloanele de la 0 la 31 (de la stînga spre dreapta) ca în fig 9.17.

Instrucțiunea PRINT AT deplasează cursorul (locul în care se va tipări) în linia și coloana specificată și tipărește lista de caractere dorită.

Formatul general este:

PRINT AT linie, coloană; listă

unde

— „linie“ este numărul liniei unde se dorește tipărirea;

— „coloană“ este numărul coloanei unde se dorește tipărirea;

— „listă“ reprezintă șirul de caractere ce urmează a fi tipărit în locul stabilit.

9. INSTRUCȚIUNI BASIC



Fig. 9.18.

EXEMPLU:

PRINT AT 11, 16; ". "

Imprimă un punct în centrul ecranului (fig. 9.18).

9.9. Instrucțiunea PRINT TAB

Instrucțiunea PRINT TAB deplasează cursorul în cadrul aceleiași linii, în coloana specificată și tipărește lista de caractere dorită. Cursorul se deplasează pe linia următoare în cazul excepție, când poziția de tipărire specificată se află înaintea poziției de tipărire actuală.

Formatul general este:

PRINT TAB coloană; listă
unde

- „coloană“ este numărul coloanei din cadrul liniei curente unde se dorește tipărirea;
- „listă“ reprezintă șirul de caractere ce urmează a fi tipărit în locul stabilit.

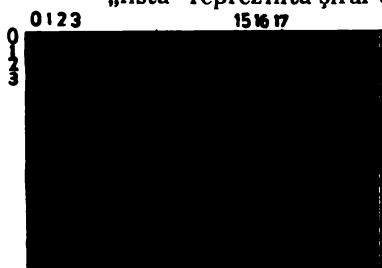


Fig. 9.19.

EXEMPLU:

Programul

10 PRINT AT 3,1; "TITLU"

20 PRINT TAB 16; "AUTOR"

tipărește pe linia 3, coloana 1 textul TITLU și pe aceeași linie, coloana 16 textul AUTOR (fig. 9.19)

De reținut următoarele:

a) după introducerea argumentelor lui PRINT TAB și PRINT AT se folosește „;“. Dacă se folosește alt separator (de exemplu virgulă sau blank), cursorul se poziționează, dar se deplasează imediat astfel încât tipărirea nu se execută acolo unde s-a dorit;

b) liniile ecranului 22 și 23 nu pot fi utilizate pentru tipărire. Ele sînt rezervate pentru introducerea de comenzi, linii program, date, mesaje etc;

c) tipărind cu PRINT AT într-o poziție deja scrisă, ultima tipărire o anulează pe precedentă;

d) calculatorul consideră că există numai 32 coloane, drept pentru care stabilirea coloanei unui PRINT TAB urmat de un număr mai mare decît 31 se face prin scăderea succesivă a lui 32 din el pînă se ajunge la un număr cuprins între 0 și 31. Se spune că PRINT TAB consideră coloanele „modulo 32“.

EXEMPLU:

PRINT TAB 32; 1

PRINT TAB 75; "\$"

este echivalent cu

PRINT TAB 0; 1

PRINT TAB 11; "\$"

fiindcă $32 - 32 = 0$ și

$75 - 32 = 43$; $43 - 32 = 11$

9.10. Instrucțiunea REM

Instrucțiunea REM introduce un mesaj într-un program. Mesajul este un comentariu la adresa programului sau la adresa unui grup de linii, în vederea scoaterii în evidență a unor caracteristici. Un program bine comentat este mai ușor de depanat și înțeles, drept pentru care îl pot adapta și folosi mai mulți utilizatori.

Formatul general este:

REM listă

unde „listă” este un șir de caractere ce aduce unele lămuriri despre program.

Instrucțiunea REM nu este luată în considerare în nici un fel când se trece la execuția programului.

EXEMPLU:

Programul de la pct. 9.2 poate primi numele „Afișează litere” prin completarea cu următoarea linie program:

5 REM Afișează litere

În exemplu, numele dat programului prin introducerea liniei program 5 reprezintă o caracteristică ce lămurește de fapt ceea ce și-a propus să execute acest mic program.

9.11. Instrucțiunea LET

Instrucțiunea LET permite atribuirea de valori unei variabile.

Formatul general este:

LET var = expr.

unde

- „var” reprezintă numele unei variabile simple sau indexate;
- „expr” reprezintă o expresie numerică sau alfanumerică.

Efectul instrucțiunii constă în evaluarea expresiei din dreapta semnului egal și atribuirea valorii astfel determinate variabilei din stînga semnului egal.

EXEMPLU:

Calculul expresiei $\frac{x+1}{2}$ în punctul $x=1$.

```
10 LET x=1
20 PRINT (x+1)/2
```

EXEMPLU:

Concatenarea șirurilor de caractere „Ionescu” și „Ana”

```
10 LET N$ = "Ionescu"
20 LET P$ = "Ana"
30 LET S$ = N$+P$
40 PRINT S$
```

Șirul S tipărit prin linia 40 are valoarea „Ionescu Ana”.

Se observă din exemplul dat că un șir de caractere poate fi atribuit ca valoare unei variabile șir sau poate fi tipărit cu o instrucțiune PRINT.

Fiind dat un șir, un subșir al lui se formează din câteva caractere consecutive ale lui, luate în secvență (unul după altul).

EXEMPLU:

Subșirurile

„eu”

„sint”

„elev”

„eu sint”

„sint elev”

sint subșiruri ale șirului

„eu sint elev”

Subșirul

„eu elev”

nu este un subșir al șirului

„eu sint elev”

pentru că s-a sărit peste „sint”.

Subșirul

„eu sint”

nu este subșir al șirului

„eu sint elev”

pentru că între „eu” și „sint” s-au introdus mai multe blancuri (spații), blancul fiind și el considerat un caracter distinct.

Manipularea subșirurilor în BASIC se face sub forma următoare:

s (n1 TO n2)

unde

— „s” este un șir de caractere sau o variabilă șir;

— „n1”, „n2” sînt numere întregi nenegative ce reprezintă numărul de ordine al caracterului de început, respectiv de sfîrșit, din subșir.

Dacă $n1 > n2$ rezultatul este șirul vid (””).

Dacă nu se precizează începutul subșirului, n1 se ia implicit egal cu 1.

Dacă nu se precizează sfîrșitul subșirului, n2 se ia implicit egal cu lungimea șirului.

EXEMPLE de manipulare de subșiruri de caractere:

„abedef” (2 TO 5) = „bede”

„abedef” (TO 5) = „abedef” (1 TO 5) = „abede”

„abedef” (2 TO) = „abedef” (2 TO 6) = „bedef”

„abedef” (TO) = „abedef” (1 TO 6) = „abedef”

„abedef” (3) = „abedef” (3 TO 3) = „e”

„abedef” (5 TO 7) dă mesaj de eroare deoarece șirul are numai șase caractere.

„abedef” (8 TO 7) = ” ”

„abedef” (1 TO 0) = ” ”

EXEMPLU:

Prin programul

```
10 LET a$ = "123456"  
20 LET a$ (3 TO 4) = "AB"  
30 PRINT a$
```

șirul de caractere „123456” este transformat în șirul de caractere „12AB56”.

9.12. Instrucțiunea BEEP

Calculatorul produce sunete cu ajutorul instrucțiunii BEEP.
Formatul general este:

BEEP d, i

unde

- „d” este o expresie numerică ce indică durata în secunde a sunetului respectiv;
- „i” este o expresie numerică ce reprezintă înălțimea sunetului măsurat în semitonuri relativ la DO central (este admisă și o valoare negativă).

Pentru a transcrie muzică este indicat să se scrie pe marginea fiecărui spațiu și linii a portativului înălțimea corespunzătoare, ținând cont de armura cheii.

Exemplu:



Fig. 9.20.

Trebuie hotărâtă în prealabil și durata notelor, după cum se observă în fig. 9.21.

la tipul de notă

corespunde durata în secunde



Fig. 9.21.

După ce s-au stabilit duratele și înălțimile notelor, oricărei partituri i se atașează un set de numere ce va fi utilizat la reproducerea ei de calculator.



Fig. 9.22.

Notele DO, FA, SOL prevăzute cu # (diez) au înălțimea cu o unitate mai mare decât notele obișnuite.

Și acum programul atașat partiturii din figura 9.22

10	BEEP 1,4	70	BEEP 0.25,11
20	BEEP 1,9	80	BEEP 1,9
30	BEEP 0.75,9	90	BEEP 0.75,8
40	BEEP 0.25,9	100	BEEP 0.25,6
50	BEEP 1,9	100	BEEP 1.5,4
60	BEEP 0.75,13		

Se poate schimba cheia partiturii în următorul mod: se adună la înălțimea fiecărei note o variabilă (de exemplu „cheie“ sau pentru a minimiza efortul de tastare numai „c“) căreia i se atribuie o valoare adecvată, înaintea execuției piesei.

EXEMPLU:

Programul anterior devine atunci:

10	BEEP 1, c +4
20	BEEP 1, c +9
30	BEEP .75, c +9
40	BEEP .25, c +9
50	BEEP 1, c +9
60	BEEP .75, c +13
70	BEEP .25, c +11


```

80 BEEP 1,c +9
90 BEEP .75, c +8
100 BEEP .25, c +6
110 BEEP 1.5, c +4

```

Variabila „c” poate lua diferite valori ca: 0 pentru DO minor, 2 pentru RE minor, 12 pentru DO minor octava superioară etc. (se pot folosi și valori zecimale).

După stabilirea valorii lui „c” este absolut necesar completarea programului cu linia 5:

```
5 LET c=2
```

Folosind procedeul descris, se poate realiza acordarea calculatorului cu un instrument.

De asemenea, este posibil să se execute piese cu viteze diferite. În exemplul ales „o pătrime” a fost programată să dureze un sfert de secundă. Dacă se introduce o variabilă („pătrime” sau „p” analog cu „cheie” sau „c”) careia i se atribuie o anumită valoare (de exemplu 0.5), programul se transformă astfel:

```

5 LET c=2: LET p=.5
10 BEEP p+1, c+4
20 BEEP p+1, c+9
30 BEEP p+.75, c+9
40 BEEP p+.25, c+9
50 BEEP p+1, c+9
60 BEEP p+.75, c+13
70 BEEP p+.25, c+11
80 BEEP p+1, c+9
90 BEEP p+.75, c+8
100 BEEP p+.25, c+6
110 BEEP p+1.5, c+4

```

Prin schimbarea valorilor variabilelor cu numele „cheie” sau „pătrime” este posibilă execuția aceluiași program în numeroase moduri, epuizînd toate dorințele (preferințele) definite prealabil.

9.13. Instrucțiunea PAUSE

Instrucțiunea PAUSE realizează o pauză în program în timpul căreia nu se desfășoară nici o operație.

Formatul general este:

PAUSE const

unde „const” este un număr între 0 și 65535.

Numărul $65535 = 2^{16} - 1$ și este determinat de mărimea memoriei calculatorului, respectiv 64 KO ($64 = 2^8$ și $\text{KO} = 1024 = 2^{10}$).

Instrucțiunea PAUSE oprește execuția programului menținînd activ ecranul pe durata specificată în perioade de baleiaj.

O perioadă de baleiaj este echivalentă cu 20 ms.

Pausa maximă posibilă corespunde instrucțiunii

PAUSE 65535

și înseamnă aproximativ 22 minute.

Pausa minimă posibilă corespunde instrucțiunii

PAUSE 0

care în realitate înseamnă oprirea definitivă a execuției programului.

O pauză obținută folosind PAUSE poate fi scurtată apăsând orice tastă (cu excepția lui SPACE și CAPS SHIFT, taste ce produc BREAK).

EXEMPLU:

Se tipăresc pe ecran pe rând fiecare, 5 numere, cu o pauză de timp între ele de aproximativ 1, 2, 3, 4 secunde.

```
10 PRINT 111
20 PAUSE 50
30 PRINT 222
40 PAUSE 100
50 PRINT 333
60 PAUSE 150
70 PRINT 444
80 PAUSE 200
90 PRINT 555
```

Cu instrucțiunea PAUSE se pot reproduce și pauzele muzicale.

EXEMPLU:

```
10 BEEP .75,4
20 BEEP .25,2
30 BEEP .5,1
40 BEEP .5,2
50 BEEP .5,4
60 BEEP .5,6
70 BEEP .5,7
80 PAUSE 20
90 BEEP .75,4
100 BEEP 25,6
110 BEEP .5,7
120 BEEP .5,7
130 BEEP .5,4
140 BEEP .5,2
150 BEEP 1,1
```

9.14. Instrucțiunea LOAD

Calculatorul HC-85 are posibilitatea să încarce în memorie ceea ce se găsește pe bandă magnetică. Conectarea calculatorului la casetofon se face cu ajutorul unui cablu special.

Instrucțiunea LOAD execută încărcarea.

LOAD este folosită sub forma de comandă.

a) Dacă se dorește încărcarea unui program de pe casetă în memorie se folosește comanda LOAD cu formatul:

LOAD "nume"

unde „nume“ este șirul de caractere ce definește numele programului ce se dorește încărcat.

Cînd se tastează LOAD, înainte de a se încărca noul program se șterge vechiul program împreună cu toate variabilele lui.

Comanda LOAD fără un nume specificat LOAD " " încarcă primul program găsit pe casetă.

b) Dacă se dorește încărcarea unor șiruri de caractere de pe casetă în memorie se folosește LOAD cu formatul:

LOAD "nume" DATA șir ()

unde

- „nume“ este numele sub care se găsește înregistrat șirul pe bandă;
- „șir ()“ este noul nume sub care se va încărca șirul în memorie.

Exemplu:

Comanda

LOAD "test" DATA T ()

caută pe bandă șirul cu numele „test“, cînd îl găsește verifică dacă nu mai există în memorie un șir T, îl anulează și încarcă șirul „test“ în memoria calculatorului sub numele de T.

9.15. Instrucțiunea SAVE

Calculatorul HC-85 are posibilitatea să înregistreze pe bandă magnetică părți din memorie, operațiune care se numește „salvarea“ conținutului respectivei părți din memorie.

Ea se execută cu ajutorul instrucțiunii SAVE.

SAVE este folosită sub formă de comandă.

a) Dacă se dorește salvarea unui program pe casetă se folosește comanda SAVE cu formatul:

SAVE "nume"

unde „nume“ este șirul de caractere ce definește numele sub care va fi salvat programul.

Numele programului este compus din maximum 10 caractere litere — și/ sau cifre — și este fixat de utilizator.

Cînd este gata de salvare calculatorul afișează mesajul:

Start tape then press any key

care înseamnă „pornește casetofonul și apoi apasă orice tastă“.

La terminarea înregistrării apare mesajul:

O.K.

adică „foarte bine“.

Este posibil să se înregistreze un program pe casetă astfel încât atunci când este reincărcat în memorie să se lanseze automat de la o linie anume.

Instrucțiunea are atunci formatul:

SAVE șir LINE număr

și face ca programul încărcat cu LOAD (dar nu și cu MERGE -- vezi 9.17) să fie rulat automat de la linia specificată prin „număr“. Dacă nu este suficient în memorie programul vechi precum și vechile variabile nu sînt șterse și apare un mesaj de eroare:

Out of memory

adică „depășire de memorie“.

b) Dacă se dorește salvarea unui șir de caractere pe casetă se folosește comanda SAVE cu formatul:

SAVE "nume" DATA șir ()

unde

- "nume" este noul nume sub care va fi înregistrat șirul pe bandă;
- „șir ()“ este numele șirului din memorie ce va fi salvat.

EXEMPLU:

Comanda

SAVE "NOU" DATA a ()

înregistrează șirul a () pe banda magnetică sub numele "NOU".

9.16. Instrucțiunea VERIFY

După înregistrarea pe banda magnetică este bine să se facă verificarea corectitudinii înregistrării.

Pentru verificare se reglează volumul casetofonului la nivel mediu și se poziționează banda în punctul în care a început înregistrarea.

Instrucțiunea VERIFY execută verificarea corectitudinii înregistrării.

VERIFY este folosită sub formă de comandă.

a) Dacă se dorește verificarea salvării unui program se folosește comanda VERIFY cu formatul:

VERIFY "nume"

unde "nume" este numele programului ce a fost înregistrat pe bandă și pentru care se verifică corectitudinea înregistrării.

Comanda VERIFY verifică dacă programul și variabilele înregistrate pe casetă sînt identice cu cele din memoria calculatorului. Dacă programul a fost înregistrat și chemat corect, pe ecran apare:

Program "nume"

În timpul căutării programului specificat, calculatorul tipărește numele tuturor programelor pe care le întilnește, iar la sfîrșit apare mesajul:

OK

În cazul unei erori de înregistrare (eroare ce apare la VERIFY) se afișează mesajul:

R tape loading error

care înseamnă „eroare de înregistrare pe casetofon” și se încearcă o nouă înregistrare.

b) Dacă se dorește verificarea salvării unui șir de caractere se folosește comanda VERIFY cu formatul:

VERIFY "nume" DATA șir ()

unde

- „nume” este numele sub care șirul a fost salvat;
- „șir ()” este numele șirului din memorie în raport cu care se verifică corectitudinea salvării.

EXEMPLU:

VERIFY "NOU" DATA a ()

verifică înregistrarea făcută pe casetă sub numele de "NOU" a șirului a () din memorie.

9.17. Instrucțiunea MERGE

Instrucțiunea MERGE servește la încărcarea și intercalarea unui program de pe bandă magnetică cu programul ce se află în memorie în acel moment.

MERGE este folosită sub formă de comandă.

Formatul general este:

MERGE "nume"

unde „nume” este numele programului de pe bandă ce urmează a fi încărcat și intercalat cu programul din memorie.

MERGE este util în cazul testării unui program complex. În momente diferite se păstrează pe bandă diferite variante ale programului, ce vor fi intercalate între ele pînă se va ajunge la varianta finală (cea mai bună).

Comanda MERGE nu poate fi folosită la intercalarea șirurilor de caractere.

9.18. Instrucțiunea INPUT

Instrucțiunea INPUT oferă o modalitate de introducere de valori direct de la tastatură.

Formatul general este:

INPUT listă; var 1, var 2,

unde

- „listă” este o constantă șir de caractere ce punctează acțiunea de introducere de date și reprezintă un mesaj (un comentariu);
- „var 1”, sau „var 2” este numele unei variabile simple sau indexate, numerice sau șir de caractere ce va primi valoarea tastată.

9. INSTRUCȚIUNI BASIC

Argumentul „listă” nu este obligatoriu.
Atribuirea valorilor în urma unui INPUT se consideră terminată în momentul tastării lui CR.

EXEMPLU:

Programul 1 calculează și tipărește perimetrul oricărui pătrat de latură L.

```
10 INPUT L
20 PRINT P=4 * L
30 GO TO 10
```

Imediat după comanda RUN calculatorul așteaptă introducerea unei valori:

— dacă nu se tastează nimic, în partea de jos a ecranului apare un semn de întrebare și calculatorul așteaptă;

— dacă se șterge semnul de întrebare printr-un DELETE și se introduce valoarea 2(L=2), sus pe ecran apare 8 (de la $4 \times 2 = 8$).

Prin linia 30 programul se reia automat și introducându-se alte valori pentru L se obțin și alte valori pentru P.

EXEMPLU:

Programul 1 poate fi modificat astfel:

```
10 INPUT "latura ="; L
20 PRINT "perimetrul ="; P=4 * L
30 GO TO 10
```

În această situație, în partea de jos a ecranului apare mesajul "latura=" care anunță că se așteaptă introducerea valorii ce va fi asignată laturii unui pătrat, iar la apariția mesajului "perimetrul=" se anunță că rezultatul corespunde perimetrului pătratului cu latura stabilită, după care urmează o nouă apariție a mesajului "latura=" și se continuă procesul de calcul.

O linie cu INPUT poate conține o serie de separatori (punct și virgulă, virgulă, apostrof) care au același efect ca într-o linie cu PRINT.

INPUT consideră orice element care începe cu o literă ca pe o variabilă asignabilă (căreia urmează să i se introducă valoarea de la tastatură). Instrucțiunea INPUT poate tipări și mesaje; pentru a tipări un șir de caractere este suficientă introducerea acestuia între ghilimele. Dacă conține și valori de variabile, mesajul se închide între paranteze.

Dacă se dorește citirea unei variabile de tip șir de caractere, a\$, pe ecran apare caracterul ghilimele. Dacă această variabilă trebuie să ia valoarea unei alte variabile de tip șir definită în program, b\$, aceasta se face prin ștergerea ghilimelelor și introducerea numelui variabilei (b\$).

EXEMPLU:

Programul calculează și tipărește perimetrul unui dreptunghi cînd lungimea și lățimea se introduc de la tastatură.

```
10 INPUT "lungime="; L; "latime="; l
20 PRINT "perimetrul="; p=2 * L+2 * l
30 GO TO 10
```

Mesajele "lungime=" și "latime=" apar pe aceeași linie, unul după altul, imediat după ce s-a introdus primul număr.

Dacă se modifică linia 10 așa fel încât după L să urmeze simbolul virgulă și nu simbolul punct și virgulă

```
10 INPUT "lungime=" ; L, "lățime="; 1
```

mesajele "lungime=" și "lățime=" apar pe aceeași linie, primul la începutul rândului, iar după ce se introduce o valoare, cel de-al doilea mesaj la mijlocul aceluiași rând.

Dacă în linia 10 se înlocuiește simbolul punct și virgulă existent după L cu simbolul apostrof

```
10 INPUT "lungime="; L' "lățime="; 1
```

după introducerea valorii corespunzătoare mesajului "lungime=" apărut la începutul unei linii, mesajul "lățime=" își face apariția la începutul liniei imediat următoare.

Instrucțiunea INPUT LINE oferă o modalitate de introducere de valori șir de caractere de la tastatură.

Formatul general este:

```
INPUT LINE listă; var. șir.
```

unde

— „listă“ este o constantă șir de caractere ce punctează acțiunea de introducere de date și reprezintă un mesaj (un comentariu);

— „var. șir“ este numele unei variabile șir de caractere ce va primi valoarea șir de caractere tastată.

Argumentul „listă“ nu este obligatoriu.

Atribuirea valorilor șir de caractere în urma unui INPUT LINE se consideră terminată în momentul tastării lui CR.

EXEMPLU:

Dacă la execuția liniei program

```
10 INPUT LINE a$
```

se introduce de la tastatură linia de date

Carte

variabilei a\$ i se atribuie valoarea constantă tip șir de caractere "Carte".

În contextul instrucțiunii INPUT LINE caracterul virgulă își pierde semnificația de separator și este privit ca o constantă șir de caractere.

EXEMPLU:

Dacă la execuția liniei program

```
10 INPUT a$
```

se introduce de la tastatură linia de date

desen, varianta 1

variabilei a\$ i se atribuie valoarea „desen“.

Dacă se înlocuiește linia 10 anterior descrisă cu linia

```
10 INPUT LINE a$
```

prin introducerea aceleiași linii de date, variabilei a\$ i se atribuie valoarea șir de caractere "desen, varianta 1".

9.20. Instrucțiunea PLOT

Instrucțiunea PLOT desenează un punct pe ecran.

Formatul general este:

PLOT x, y

unde

- „x“ reprezintă distanța față de extrema stîngă a ecranului și se numește abscisa punctului;
- „y“ reprezintă distanța față de baza ecranului și se numește ordonata punctului.

Se știe de la pct. 9.8 că partea utilizabilă a ecranului are 22 de linii și 32 coloane.

$22 \text{ linii} \times 32 \text{ coloane} = 704 \text{ poziții de caracter.}$

Fiecare poziție de caracter este la rîndul său un pătrat format din $8 \times 8 = 64$ puncte. Punctele se numesc și pixeli. Deci pe ecranul televizorului se vor putea desena (utiliza)

$704 \times 64 = 45056 \text{ puncte}$

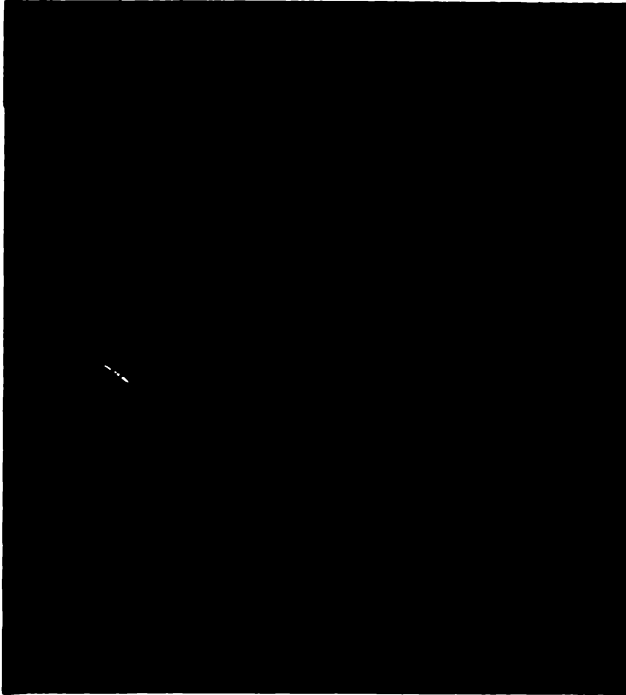


Fig. 9.23.

Poziția unui pixel este specificată prin numerele x și y.

Se mai spune că x și y sînt coordonatele punctului respectiv. Matematic, coordonatele unui punct se descriu ca o pereche de numere între paranteze, astfel că pentru HC-85:

IV. PROGRAMAREA ÎN BASIC, PE HC-85

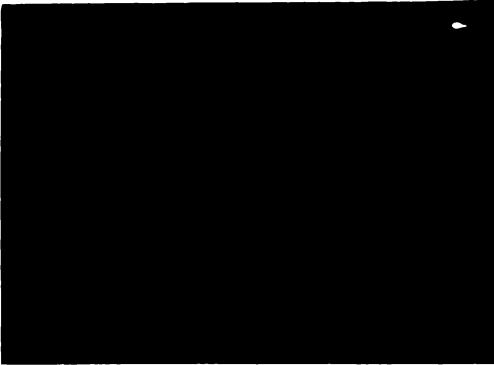


Fig. 9.24.

(0,0) — este punctul din
extrema stînga jos a ecranului;
(255,0) — este punctul din
extrema dreapta jos a ecranului;
(0,175) — este punctul din
extrema stîngă sus a ecranului;
(255,175) — este punctul din
extrema dreapta sus a ecranului.

Trebuie reținut deci că, x și y
trebuie să respecte condițiile:

$$0 \leq x \leq 255$$

$$0 \leq y \leq 175$$

Coordonatele x și y din instrucțiunea PLOT pot fi numere sau expresii aritmetice a căror valoare trebuie să respecte condițiile specificate înainte.

EXEMPLU:

Execuția liniei program

10 PLOT 127,87

Inseamnă desenarea unui singur punct în centrul ecranului.

9.21. Instrucțiunea DRAW

Instrucțiunea DRAW trasează de la ultimul punct desenat un segment de dreaptă sau o porțiune de cerc.

Formatul general este:

DRAW x, y, r

unde

- „ x ” reprezintă distanța pe orizontală față de abscisa (x -ul) ultimului punct desenat;
- „ y ” reprezintă distanța pe verticală față de ordonata (y -ul) ultimului punct desenat;
- „ r ” reprezintă numărul de radiani corespunzător lungimii arcului de cerc.



Fig. 9.25.

Instrucțiunea DRAW în care nu este specificat r trasează un segment de dreaptă.

EXEMPLU:

Programul următor trasează prima diagonală a ecranului (fig. 9.25).

10 PLOT 0,0

20 DRAW 255,175

Argumentele x și y din DRAW pot fi pozitive sau negative potrivit sensului de trasare dorit și trebuie să respecte condițiile:

- $255 < x < 255$
- $175 < y < 175$

Sensul pozitiv pentru argumentul x din DRAW înseamnă dreapta față de x -ul ultimului punct desenat.

Sensul negativ pentru argumentul x din DRAW înseamnă stînga față de x -ul ultimului punct desenat.

Sensul pozitiv pentru argumentul y din DRAW înseamnă sus față de y -ul ultimului punct desenat.

Sensul negativ pentru argumentul y din DRAW înseamnă jos față de y -ul ultimului punct desenat.

EXEMPLE:

În programele următoare se trasează segmentul AB cu sensul de la A la B. Trebuie subliniat că în urma execuției, pe ecran este afișată numai linia continuă AB, celelalte fiind linii ajutoare trasate în sprijinul înțelegerii modalității concrete de funcționare a instrucțiunii exemplificate.

Programul 1 (fig. 9.26)

```
10 PLOT 50, 50  
20 DRAW 100, 100
```

Programul 2 (fig. 9.27)

```
10 PLOT 100, 100  
20 DRAW 50, 50
```

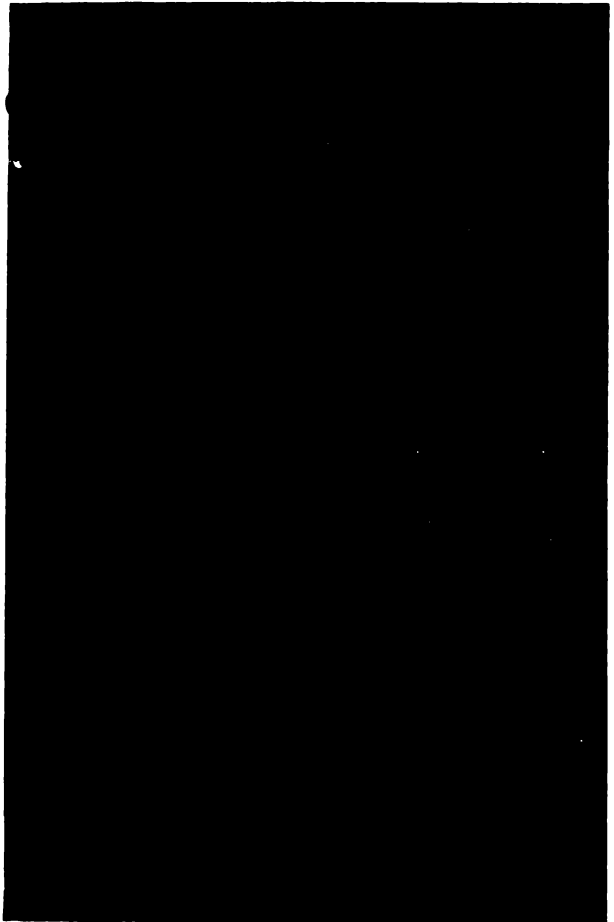


Fig. 9.27.



Instrucțiunea DRAW în care este specificat argumentul r trasează un arc de cerc. Este obligatoriu ca valoarea lui r să fie exprimată în radiani și în acest caz argumentele x , y și r pot fi negative sau pozitive:

- a) dacă $r < 0$ arcul de cerc este trasat în sensul acelor de ceasornic;
- b) dacă $r > 0$ arcul de cerc este trasat în sensul invers acelor de ceasornic.

După cum se știe, unitatea de măsură frecvent utilizată pentru unghiuri (arce de cerc) este gradul sexagesimal.

Un unghi la centru corespunzător unui cerc complet este de 360° .

O altă unitate de măsură pentru unghiuri (arce) utilizată des în știință și tehnică este radianul.

Acesta se definește ca fiind măsura unghiului la centru corespunzător unui arc de cerc egal ca lungime cu raza cercului respectiv.

În cazul în care lungimea arcului $AB = OA = \text{raza}$ se spune că:

măsura $\widehat{AOB} = 1$ radian

Correspondența dintre gradele sexagesimale și radiani este dată de relația:

$$360^\circ = 2\pi \text{ radiani}$$

unde π (se citește „pi”) este un număr care are valoarea aproximativă egală cu 3,1416 Acest număr este introdus în memoria permanentă a calculatorului (în ROM) și se obține prin apăsarea tastei notată cu „PI”. Atenție, tastarea literelor „P” și „I” nu este echivalentă cu acționarea tastei PI!

Folosind această formulă se poate deduce (cu ajutorul unor calcule simple) mărimea în radiani a oricărui unghi, după cum urmează:

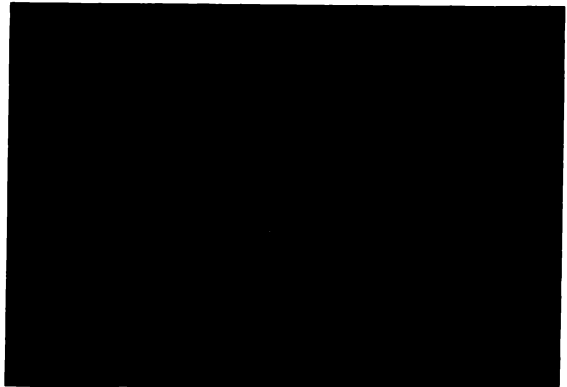
$$a) 180^\circ = \frac{360^\circ}{2} = \frac{2\pi}{2} \text{ radiani} = \pi \text{ radiani}$$

$$b) 60^\circ = \frac{360^\circ}{6} = \frac{2\pi}{6} \text{ radiani} = \frac{\pi}{3} \text{ radiani} = 0.333 \text{ radiani}$$

$$c) 70^\circ = 60^\circ + 10^\circ = \frac{360^\circ}{6} + \frac{360^\circ}{36} = \left(\frac{\pi}{3} + \frac{2\pi}{36}\right) \text{ radiani} = \left(\frac{\pi}{3} + \frac{\pi}{18}\right) \text{ radiani} = (0.333 + 0.055) \pi \text{ radiani} = 0.388 \pi \text{ radiani}$$

EXEMPLE:

În programele următoare se trasează semicercul AB cu sensul de la A la B . Trebuie subliniat că în urma execuției pe ecran apare numai semicercul AB , celelalte fiind linii ajutătoare.



Programul 7 (fig. 9.32)

10 PLOT 20, 30

20 DRAW 100, 60, PI

Fig. 9.32.

Programul 12 (fig. 9.37)

10 PLOT 110, 80

20 DRAW 70, -20, -PI

Dacă pentru π se dau valori diferite de exemplu, $\pi/4$, $\pi/2$, $3\pi/2$, etc., se obțin alte arce de cerc: o optime, o pătrime, trei sferturi de cerc, etc.

Fig. 9.37.

9.22. Instrucțiunea FOR-NEXT

Instrucțiunea FOR și instrucțiunea NEXT sînt nedespărțite.

Prima definește începutul unui ciclu, iar a doua sfîrșitul lui.

Prin ciclu se înțelege execuția de un număr prestabilit de ori a grupului de instrucțiuni cuprinse între FOR și NEXT.

Formatul general pentru FOR este:

FOR var= val_1 TO val_2 STEP val_3

unde

- „var“ este o variabilă numerică simplă (numele său trebuie să fie obligatoriu format dintr-o singură literă) ce reprezintă variabila de ciclu și este un contor (numărător);
- „ val_1 “ este *valoarea inițială* dată lui „var“;
- „ val_2 “ este *valoarea finală* la care poate ajunge „var“;
- „ val_3 “ este *valoare pasului* sau valoarea ce reprezintă diferența dintre două valori consecutive ale contorului.

Argumentele val_1 , val_2 și val_3 pot fi exprimate prin constante, expresii numerice sau variabile ce au valori numerice.

Dacă pasul lipsește se consideră implicit că are valoarea 1.

Formatul general pentru NEXT este:

NEXT var

unde „var“ este variabila contor din FOR.

Un ciclu arată schematic astfel:

FOR var= val_1 TO val_2 STEP val_3

corpul
ciclului { linii program

NEXT var

Execuția unui ciclu constă din etapele descrise în fig. 9.38.

EXEMPLE:

Programul 1 afișează de 5 ori textul "CALCULATORUL HC-85"

```
10 FOR I = 1 TO 5
20 PRINT "CALCULATORUL HC-85"
30 NEXT I
```

Programul 2 afișează în ordine crescătoare primele 10 numere naturale impare.

```
10 FOR I = 1 TO 19 STEP 2
20 PRINT I
30 NEXT I
```

Programul 3 afișează în ordine descrescătoare primele 10 numere naturale.

```
10 FOR n = 10 TO 1 STEP - 1
20 PRINT n
30 NEXT n
```

Programul 4 trasează graficul funcției SIN pentru valori între 0 și 2 PI.

```
10 FOR n = 0 TO 255
20 PLOT n, 88 + 80 * SIN (n/128 * PI)
30 NEXT n
```

Fig. 9.38.

Două cicluri FOR — NEXT pot fi imbricate (intercalate) sau complet separate. Este greșită suprapunerea parțială a două cicluri.

De asemenea trebuie evitat saltul din exterior în interiorul unei bucle FOR—NEXT deoarece contorul nu poate fi inițializat decât printr-o instrucțiune FOR. Pentru a fi siguri că nu se fac salturi în interiorul unui ciclu se pot scrie toate instrucțiunile ciclului pe o singură linie (dacă spațiul permite).

EXEMPLU:

Programul acesta este o exemplificare de două cicluri FOR — NEXT imbricate.

El calculează și tipărește aria dreptunghiurilor cu lățime ce variază între 1 și 100, crescător din 1 în 1 și cu lungime ce variază între 200 și 20, descrescător din 2 în 2.

```

10 FOR I          = 1 TO 100 STEP 1
20 FOR J          = 200 TO 2 STEP -2
30 PRINT "lațimea = "; I; "lungimea = "; J; "aria = "; I * J
40 PRINT
50 NEXT J
60 NEXT I

```

9.23. Instrucțiunea RND

Prin instrucțiunea RND se generează numere aleatoare (la întâmplare) fără a folosi tastatura.

RND nu este o funcție complet aleatoare ci o funcție periodică cu perioadă suficient de mare (65535), astfel încât efectul de periodicitate poate fi neglijat. În cadrul unei perioade, numerele generate sînt complet aleatoare. În anumite privințe, RND se comportă ca o instrucțiune fără argumente: efectuează calcule și produce un rezultat. De fiecare dată cînd e utilizată, rezultatul este un număr aleator nou, cuprins între 0 și 1 (uneori poate lua valoarea 0, dar niciodată 1).

Formatul general este:

RND

Dacă se dorește ca numerele aleatoare să fie într-un anumit domeniu de valori se poate proceda ca în exemplele următoare:

5 * RND	generează numere între 0 și 5;
1.3+0.7 * RND	produce numere între 1.3 și 2;
1+INT (RND * 6)	furnizează numere aleatoare între 1 și 6.

EXEMPLE:

Programul 1 simulează aruncarea zarurilor după fiecare acționare a tastei CR.

```

10 CLS
20 FOR n = 1 TO 2
30 PRINT 1+INT (RND * 6); " ";
40 NEXT n
50 INPUT a$
60 GO TO 20

```

Programul 2 trasează aleator un punct pe ecran după fiecare acționare a tastei CR.

```

10 PLOT INT (RND * 255), INT (RND * 175)
20 INPUT a$
30 GO TO 10

```

9.24. Instrucțiunea RANDOMIZE

Instrucțiunea RANDOMIZE este utilizată împreună cu RND și face ca RND să pornească dintr-un punct anumit al secvenței de generare a numerelor.

Formatul general este:

RANDOMIZE const

unde „const“ este un număr cuprins între 0 și 65535 (perioada lui RND) care reprezintă numărul de ordine al viitorului apel al unei instrucțiuni RND.

```
10 RANDOMIZE 1
20 FOR i = 1 TO n
30 PRINT RND
40 NEXT i
50 GO TO 10
```

Linia program 10 fixează pentru RND condiția de a începe de la numărul de ordine 1. Primul număr generat de RND este întotdeauna 0.0022735596.

Instrucțiunea RANDOMIZE poate fi plasată oriunde în program, dar este indicat să se folosească la începutul lui. Practica recomandă utilizarea instrucțiunii RANDOMIZE după punerea la punct a programului.

RANDOMIZE, ca și RANDOMIZE 0, are efect diferit de RANDOMIZE urmat de un număr. Această formă a instrucțiunii se referă la timpul trecut de la punerea în funcțiune a calculatorului.

```
10 RANDOMIZE
20 PRINT RND: GO TO 10
```

determină tipărirea aceluiași număr. Deoarece timpul de lucru al calculatorului a crescut cu aceeași cantitate la fiecare execuție a lui RANDOMIZE, următorul RND furnizează aproximativ același rezultat.

Pentru a se obține o secvență aleatoare se înlocuiește GO TO 10 cu GO TO 20.

9.25. Instrucțiunea CIRCLE

Instrucțiunea CIRCLE desenează pe ecran conturul unui cerc.

Formatul general este:

CIRCLE x, y, r

unde

- „x“ și „y“ sînt coordonatele centrului cercului și au aceeași semnificație ca la pct. 9.20;
- „r“ este lungimea în puncte (pixeli) a razei cercului.

Argumentul r din CIRCLE trebuie să fie pozitiv (dacă semnul este minus el este ignorat), iar coordonatele x și y pot fi constante, expresii numerice sau variabile cu valori numerice, ce trebuie să respecte condițiile:

$$0 \leq x \pm r \leq 255$$
$$0 \leq y \pm r \leq 175$$

EXAMPLE:

Trebuie reținut că după execuția următoarelor programe concepute în scopul exemplificării instrucțiunii grafice CIRCLE, pe ecran apare numai imaginea desenată cu negru, axele de coordonate și restul liniilor ajutătoare fiind trasate numai pentru ușurarea înțelegerii.

Programul 1 desenează conturul unui cerc, centrul său și una din tangentele sale (fig. 9.39)

```
10 CIRCLE 100, 100, 50
20 PLOT 100, 100
30 PLOT 0,150
40 DRAW 200, 0
```

Prin linia 10 se trasează cercul cu centrul în punctul de coordonate (100, 100) și de rază 50.

Prin linia 20 se trasează punctul ce reprezintă centrul cercului.

Prin liniile 30 și 40 se trasează segmentul de dreaptă AB, tangent la cercul deja realizat.

Programul 2 desenează un disc negru în mijlocul ecranului (de fapt 50 de cercuri concentrice) (fig. 9.40)

```
10 FOR I = 1 TO 50
20 CIRCLE 127, 87, I
30 NEXT I
```

Programul 3 desenează 5 cercuri concentrice (primul cerc este numai un punct ce reprezintă centrul cercurilor) (fig. 9.41)

```
10 FOR I = 0 TO 50 STEP 10
20 CIRCLE 127, 87, I
30 NEXT I
```

9.26. Instrucțiunea OVER

Diferite figuri cu diverse forme se pot trasa folosind cu ingeniozitate instrucțiunile grafice PLOT, DRAW sau CIRCLE. În procesul proiectării apar situații care impun modificarea unei anumite părți dintr-o figură, Pentru asemenea cazuri a fost concepută instrucțiunea OVER.

Instrucțiunea OVER oferă posibilitatea ștergerii unui punct, unei

Fig. 9.41.

drepte, unui arc de cerc sau unui cerc, după cum este folosită alături de PLOT, DRAW sau CIRCLE.

Formatul general este:

OVER 1

Avantajul instrucțiunii OVER constă în faptul că permite ștergerea unei părți dintr-o figură fără a afecta restul figurilor de pe ecran.

Altfel instrucțiunea OVER cît și instrucțiunea CLS efectuează o ștergere, fiecare însă în alt mod, prima afectează o parte din desen, a doua tot ecranul.

Trebuie ținut seama de constatarea că o dreaptă sau un arc de cerc se șterg complet în aceeași direcție și sens în care au fost trasate.

EXEMPLE:

Deși desenul realizat pentru fiecare program constă în mai multe puncte, drepte etc., după execuție, numai imaginea neagră continuă rămîne pe ecran (fără axele de coordonate și celelalte linii ajutoare).

Programul 1 (fig. 9.42)

```
10 PLOT 10, 10
20 DRAW 50, 50
30 DRAW OVER 1, -50, -50
```

Linia 30 șterge dreapta trasată de liniile program 10 și 20 cu excepția punctului de coordonate (60, 60).

Programul 2 (fig. 9.43)

```
10 PLOT 10, 10
20 DRAW 50, 50
30 DRAW OVER 1, 50, 50
```

Prin linia program 30 nu se șterge dreapta de pe ecran, ci se prelungește, pentru că nu s-a respectat convenția referitoare la sensul de ștergere (să fie identic cu sensul de trasare a dreptei).

Programul 3

Dreapta trasată de liniile program 10 și 20 din programul 2 se șterge prin adăugarea liniilor.

```
30 PLOT OVER 1, 10, 10
40 DRAW OVER 1, 50, 50
```

Programul 4 (fig. 9.44)

```
10 PLOT 50, 50
20 DRAW 30, 30, PI
30 DRAW OVER 1, -30, -30, PI
```

Linia 30 nu șterge semicercul desenat de liniile 10 și 20, ci completează desenul cu cealaltă parte a semicercului.

Fig. 9.44.

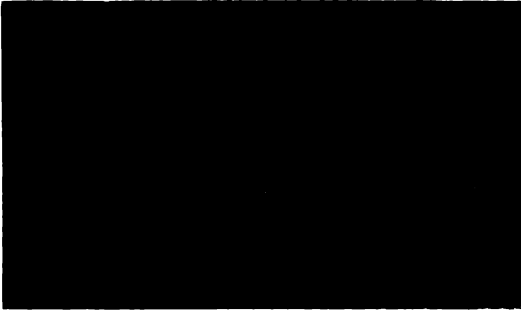


Fig. 9.45.

Programul 5 (fig. 9.45)

```
10 PLOT 50, 50
20 DRAW 30, 30, PI
30 DRAW OVER 1, -30, -30, PI
```

Linia 30 șterge semicercul trasat cu ajutorul liniilor 10 și 20, dar lasă pe ecran urma altorva puncte (în număr de 13) semn că schimbându-se sensul, nu se parcurge chiar același traseu.

Programul 6

```
10 PLOT 50, 50
20 DRAW 30, 30, PI
30 PLOT OVER 1, 50, 50
40 DRAW OVER 1, 30, 30, PI
```

Linile 30 și 40 șterg complet semicercul trasat de liniile program 10 și 20.

Programul 7

```
10 CIRCLE 80, 50, 30
20 CIRCLE OVER 1, 80, 50, 30
```

Linia 20 șterge complet cercul desenat de linia 10.

9.27. Instrucțiunea IF

Instrucțiunea IF transpune în limbaj BASIC luarea unei decizii. Ea servește la realizarea unui transfer al contorului sau la execuția unei instrucțiuni în funcție de valoarea logică a unei condiții.

Formatul general este:

IF cond THEN instr

unde

- „cond“ este o expresie aritmetică, logică sau de relație cu valoarea de adevărat sau fals;
- „instr“ este instrucțiunea ce urmează să se execute dacă condiția este adevărată.

Dacă condiția este adevărată se execută instrucțiunea sau secvența de instrucțiuni scrisă după THEN.

Dacă condiția nu este adevărată se execută instrucțiunea de pe linia următoare a lui IF.

Se pot imagina tot felul de condiții.

Una dintre cele mai simple, compară două numere sau două șiruri de caractere. Se poate testa dacă două numere sînt egale sau dacă unul este mai mare decît celălalt. Se poate testa și egalitatea a două șiruri de caractere sau dacă în ordinea alfabetică, unul apare înaintea celuilalt etc.

EXAMPLE:

Programul 1 compară mereu dacă primul număr introdus de la tastatură este strict mai mare decît al doilea număr introdus tot de la tastatură și tipărește pe cel mai mare dintre ele. (fig. 9.46).

```

10 INPUT "nr. 1", a
20 INPUT "nr. 2", b
30 IF a > b THEN PRINT
"nr. 1 este mai mare", a: GO TO 10
40 PRINT "nr. 2 este mai
mare", b
50 GO TO 10

```

Programul 2 tipărește primele
20 numere naturale. (fig. 9.47)

```

10 LET a = 0
20 LET a = a + 1
30 IF a = 20 THEN GO TO 60
40 PRINT a
50 GO TO 20
60 PRINT a
70 PRINT "SFIRSIT"

```

Programul 3 afișează dacă re-
zultatul unei expresii aritmetice este
negativ, zero sau pozitiv. (fig. 9.48)

În schema logică aleasă se ob-
servă că un al treilea bloc de deci-
zie (în care s-ar compara dacă „e”
este mai mare ca 0) dispăre, deoa-
rece dacă „e” nu este mai mic ca zero
și nici egal cu zero atunci sigur este
mai mare ca zero.

Fig. 9.47.

```

10 INPUT v
20 LET e = (v-v * 5 / (v-1)) + 150.v^2
30 IF e < 0 THEN PRINT "expresia este negativa": GO TO 10
40 IF e = 0 THEN PRINT "expresia este zero": GO TO 10
50 PRINT "expresia este pozitiva"
60 GO TO 10

```

Programul 4 (distractiv) care determină frecvența de apariție a banului sau a stemei
la aruncarea unei monezi.

```

10 LET ban = 0
20 LET stema = 0
30 LET moneda = INT (RN D.2)
40 IF moneda = 0 THEN LET ban = ban + 1

```

9. INSTRUCȚIUNI BASIC

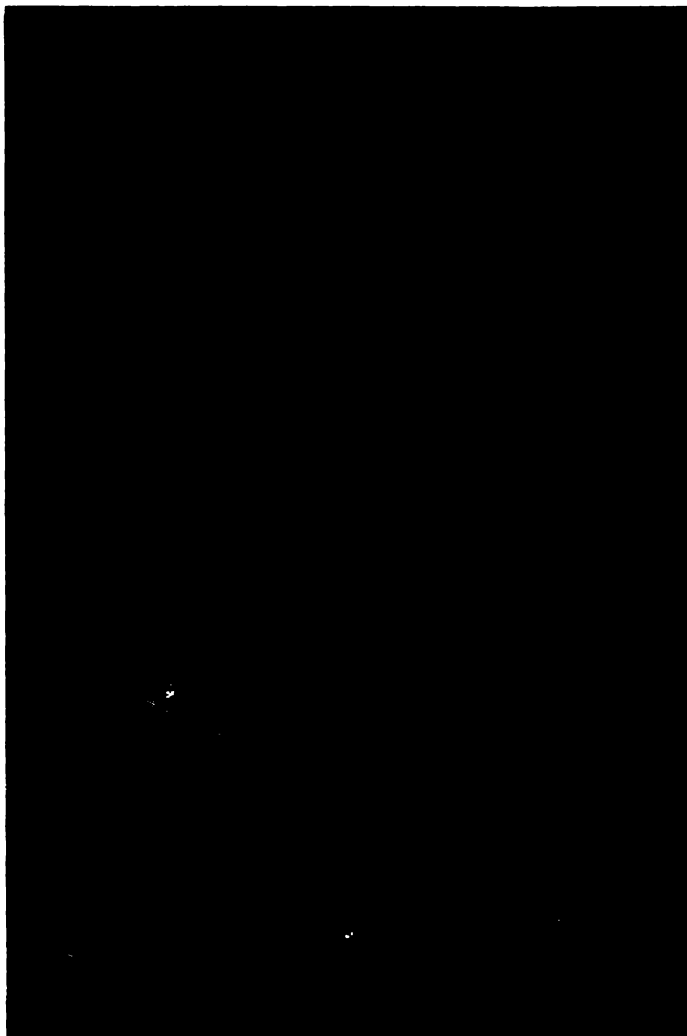


Fig. 9.48.

```
50 IF moneda = -1 THEN LET stema = stema + 1
60 PRINT ban; ", "; stema
70 IF stema > 0 THEN PRINT ban / stema;
80 PRINT
90 GO TO 20
```

Dacă timpul de rulare a programului este suficient de mare, raportul $\text{ban} / \text{stema}$ devine aproximativ 1, deoarece numerele aleatoare generate cu RND sînt uniform repartizate în intervalul (0, 1).

9.28. Instrucțiunea STOP

Instrucțiunea STOP intrerupe temporar execuția unui program. Execuția poate fi reluată începînd cu linia program imediat următoare

opririi prin comanda CONTINUE sau începînd cu linia program specificată printr-o comandă GO TO.

Formatul general este:

STOP

(fără nici un argument).

Instrucțiunea STOP poate să apară în orice punct al programului și pe orice poziție a unei linii program multiple.

Exemplu :

Programul următor verifică dacă se cunosc numerele prime din intervalul [20, 50] și totodată afișează aproximativ în mijlocul ecranului numărul prim introdus.

```
10 INPUT "alege un nr. prim între 20 și 50"; N
20 CLS
30 IF N < 20 THEN PRINT "nr. ales este mai mic ca 20"; N: GO TO 10
40 IF N=51 THEN PRINT "nr. ales este mai mare ca 50"; N: GO TO 10
50 IF N=23 OR N=29 OR N=31 OR N=37 OR N=41 OR N=43 OR N=47 THEN
  PRINT AT 11, 16; N: STOP
60 PRINT "nr. ales nu este prim"
70 GO TO 10
```

Prin linia 10 se introduce un număr, iar linia 20 șterge ecranul (de fapt numărul introdus); numărul va fi tipărit imediat după mesajul de eroare dacă nu este corect ales, în caz contrar în mijlocul ecranului. Linia 30 verifică dacă numărul introdus este mai mic, iar linia 40 dacă este mai mare față de intervalul ales. În program, linia 50 compară numărul introdus cu toate numerele prime posibile și dacă alegerea este corectă se afișează acel număr, iar execuția se oprește la întîlnirea instrucțiunii STOP.

În această situație, în partea de jos a ecranului apare mesajul

```
9 STOP startement, 50: 3
```

care indică că oprirea este cauzată de a treia instrucțiune din linia 50.

Dacă numărul ales nu este corect, se execută și linia 60 care afișează mesajul de eroare, iar prin linia 70 se reia programul de la început.

9.29. Instrucțiunea CONTINUE

Instrucțiunea CONTINUE se folosește împreună cu instrucțiunea STOP pentru a se relua o execuție.

CONTINUE este folosită sub formă de comandă.

Formatul general este:

CONTINUE

(fără nici un argument).

Este util de reținut că STOP și CONTINUE împreună, oferă un mijloc eficient de punere la punct a programelor (în special cele de dimensiuni mari

sau de complexitate deosebită) prin testarea eşalonată a liniilor sau a unui grup de linii.

Ca exemplu, în cazul programului din capitoul anterior, după introducerea unui număr prim corect și afișarea lui, se poate relua execuția (fără a tasta RUN) folosindu-se comanda CONTINUE.

9.30. Instrucțiunea DIM

La pct. 8.4.3. s-a definit noțiunea de variabilă indexată. Unei variabile indexate îi corespunde un număr de șiruri de caractere egal cu numărul indicilor săi. În locul denumirii de variabilă indexată este folosită denumirea de tablou (sau matrice), iar numărul indicilor corespunde numărului dimensiunilor tabloului.

O variabilă indexată poate fi numerică sau alfanumerică drept pentru care elementele tabloului pot fi constante de orice tip.

Un tablou numeric este reprezentat printr-un nume format dintr-o singură literă, pe când un tablou alfanumeric este reprezentat tot printr-un nume format dintr-o singură literă urmată de caracterul \$.

Pentru a putea fi utilizat, unui tablou trebuie să i se rezerve un spațiu în memorie înainte de completarea sa.

Spațiul necesar unui tablou este echivalent cu spațiul total necesar memorării tuturor elementelor sale. Numărul elementelor unui tablou este dat de rezultatul produsului dintre valorile dimensiunilor sale.

Instrucțiunea DIM servește pentru declararea dimensiunilor unui tablou.

Formatul general este:

DIM var (dim1, dim2, ...)

unde

- „var” este numele tabloului;
- „dim1, dim2, ...” sînt dimensiunile tabloului și sînt exprimate prin constante, expresii aritmetice sau variabile ce au ca valoare un număr natural diferit de 0.

Printr-o instrucțiune DIM se definesc numai dimensiunile unui singur tip de tablou.

Ea are următorul efect:

- rezervă spațiu pentru fiecare element al tabloului (adică $\text{const1} \times \text{const2} \times \dots = \text{numărul de locuri}$);
- inițializează toate elementele tabloului cu 0;
- șterge orice tablou cu același nume.



Fig. 9.49

EXEMPLE:

Linia program

10 DIM a\$(5)

rezervă 5 locuri în memorie pentru elementele tabloului a\$ (fig. 9.49).

Linia program

20 DIM P(3, 2)

rezervă 6 locuri (6 de la 3×2) în memorie pentru elementele tabloului P.

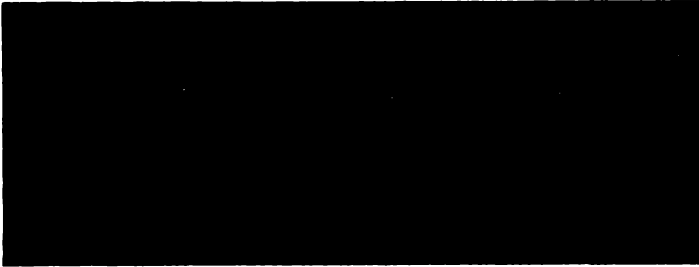


Fig. 9.50.

Pentru tabloul cu numele a din exemplul 1 (prevăzut cu un singur indice) s-a rezervat un șir de 5 locuri consecutive, deoarece este definit ca o succesiune obișnuită de 5 elemente.

Pentru tabloul cu numele P din exemplul 2 (prevăzut cu 2 indici) se rezervă în final tot un șir de locuri, procedându-se astfel: în prima fază se face rezervarea a 3 locuri pentru elementele din prima coloană, după care imediat în continuarea acestora, alte 3 locuri pentru elementele din cea de-a doua coloană. Utilizând acest model se ajunge la rezervarea celor 6 locuri consecutive necesare introducerii tuturor elementelor tabloului P.

Procedeeul descris rămâne valabil în cazul oricărui tablou cu un număr oarecare de indici.



Fig. 9.51.

Linia program

30 DIM r (2, 3, 2)

rezervă 12 locuri (12 de la $2 \times 3 \times 2$) în memorie pentru elementele tabloului r.



Fig. 9.52.

Valorile dimensiunilor se pot exprima fie printr-o constantă, fie printr-o variabilă astfel că efectul liniilor 20 și 30 din exemplele anterioare, este identic cu cel din următoarea secvență de instrucțiuni:

```
5 LET a = 2
10 LET b = 3
20 DIM P(b, a)
30 DIM r(a, b, a)
```

9. INSTRUCȚIUNI BASIC

Programul

```
10 DIM A(4)
20 LET A=0
30 FOR I = 1 TO 4
40 LET A(I)=5 ↑ I
50 PRINT "elementul"; I; "="; A(I)
60 LET A=A+A(I)
70 NEXT I
80 PRINT
90 PRINT "suma elementelor="; A
```

afișează pe rînd cele 4 elemente ale tabloului A (în care s-au introdus diverse puteri ale lui 5), după care se tipărește lăsîndu-se un rînd liber, suma valorilor introduse.

Trebuie reținut faptul că stabilirea dimensiunilor unui tablou prin instrucțiunea DIM se face la începutul unui program și obligatoriu înaintea prelucrării elementelor aceluiași tablou.

9.31. Instrucțiunea DATA

Instrucțiunea DATA introduce date în timpul execuției programului fără intervenția utilizatorului.

Formatul general este:

DATA const1, const2, ...

unde „const1, const2, ...” sînt constante numerice sau alfanumerice.

Instrucțiunile DATA pot fi plasate oriunde în program, ele comportîndu-se ca o listă unică realizată prin concatenarea tuturor instrucțiunilor DATA din program (lista DATA). Practica recomandă gruparea instrucțiunilor DATA la sfîrșitul programului pentru verificare rapidă în momentul testării sau execuției.

Instrucțiunea DATA se folosește împreună cu instrucțiunea READ.

9.32. Instrucțiunea READ

Instrucțiunea READ citește datele dintr-o listă DATA și le atribuie ca valori variabilelor specificate în ea.

Formatul general este:

READ var1, var2, ...

unde „var1, var2, ...” sînt variabile simple sau indexate de tip numeric sau șir de caractere.

Cînd calculatorul citește prima variabilă cu READ, ei îi este asociată prima valoare din lista DATA, și așa mai departe.

Dacă se încearcă citirea mai multor variabile decît numărul valorilor din lista DATA, atunci apare eroare.

Instrucțiunile READ și INPUT au același efect, respectiv acela de a introduce date pentru execuția programului.

Diferența dintre ele rezidă din modul de funcționare: READ ia datele dintr-o instrucțiune DATA, iar INPUT prin tastare directă.

La execuția instrucțiunii READ se face o verificare strictă a corespondenței tipului de variabile cu tipul constantelor disponibile din instrucțiunea DATA.

Instrucțiunile READ pot fi puse oriunde în program, dar pentru ele este neapărat necesară existența unor instrucțiuni DATA.

EXEMPLE:

Programul 1 afișează 5 numere fără a utiliza în execuție tastatura.

```
10 DATA 5, 10, -3, 7, 2
20 READ a, b, c, d, e
30 PRINT "nr. 1 = "; a
40 PRINT "nr. 2 = "; b
50 PRINT "nr. 3 = "; c
60 PRINT "nr. 4 = "; d
70 PRINT "nr. 5 = "; e
```

Programul 2 afișează lunile anului.

```
10 DATA "ian", "febr", "mart", "mai", "iunie", "iulie", "aug", "sept", "oct",
"nov", "dec"
20 DIM l$(12)
30 FOR i = 1 TO 12
40 READ l$(i)
50 PRINT "luna"; i; l$(i)
60 NEXT i
```

9.33. Instrucțiunea RESTORE

Instrucțiunea RESTORE permite refolosirea prin READ a datelor introduse prin DATA.

Formatul general este:

RESTORE nr. linie

unde „nr. linie” este numărul liniei DATA ce se va refolosi.

Instrucțiunea RESTORE face ca instrucțiunea READ următoare să citească datele de la o instrucțiune aflată la linia „nr. linie” sau după aceasta.

Dacă „nr. linie” nu este specificat se ia implicit 1 și se refolosește de fapt linia DATA cu cel mai mic număr de ordine.

EXEMPLU:

```
10 READ a, b
20 PRINT a, b
30 RESTORE 10
40 READ x, y, z
50 PRINT x, y, z
60 DATA 1, 2, 3
70 STOP
```

Pentru că în linia 30 este specificată o instrucțiune RESTORE, citirea variabilelor la liniile 10 și 40 utilizează aceeași linie de date (linia cu numărul de ordine 60) astfel că variabilele primesc valorile:

9. INSTRUCȚIUNI BASIC

a = 1 b = 2
x = 1 y = 2 c = 3

9.34. Instrucțiunea DEF FN

Spre deosebire de funcțiile standard (funcții matematice și funcții pe șir de caractere), limbajul BASIC admite și funcții definite de utilizator. Ele sînt cunoscute sub numele de funcții utilizator.

Funcțiile utilizator au sens numai pe parcursul folosirii programului în care au fost definite.

Instrucțiunea DEF FN face posibilă definirea unei funcții utilizator. Formatul general este:

DEF FN nume (var1, var2, ...) =expresie
unde

- „nume“ este numele dat funcției;
- „var1, var2 ...“ reprezintă parametri formali ai funcției și sînt variabile simple sau indexate;
- „expr“ este expresia ce dă corespondența dintre elementele funcției.

Funcțiile utilizator sînt de două feluri potrivit datelor pe care le manevrează:

- numerice
sau
- alfanumerice.

Tipul funcțiilor utilizator rezultă din argumentul „nume“ din DEF FN și anume:

- o singură literă pentru funcțiile numerice și
- o literă urmată de caracterul \$ pentru funcțiile alfanumerice.

Forma sub care este apelată în program funcția utilizator este:

FN nume (arg1, arg2 ...)

EXEMPLE:

Definirea funcției de ridicare la pătrat se face astfel:

DEF FN f(x) = x * x

Afișarea valorii acestei funcții în punctul 5 generează următoarea secvență de linii program:

```

10 DEF FN f(x) = x * x
20 LET x=5
30 LET t=FN f(x)
40 PRINT t

```

Programul

```

10 DEF FN t(i) = val + i/2
20 LET val = 0
30 PRINT FN t(30)
40 LET val = 10
50 PRINT FN t(30)

```

afișează pe rînd numerele 15 și respectiv 25 ce corespund valorilor funcției în următoarele condiții:

$$\left. \begin{array}{l} \text{val} = 0 \\ \text{i} = 30 \end{array} \right\} \Rightarrow t(30) = 0 + 30/2 = 0 + 15 = 15$$

$$\left. \begin{array}{l} \text{val} = 10 \\ \text{i} = 30 \end{array} \right\} \Rightarrow t(30) = 10 + 30/2 = 10 + 15 = 25$$

9.35. Instrucțiunea GOSUB—RETURN

Practica a generat necesitatea repetării de un număr de ori a unui grup de linii program în cadrul execuției aceluiași program.

Acest grup de linii formează un subprogram sau o subrutină.

Manipularea unei subrutine se face cu ajutorul a două instrucțiuni, GOSUB și RETURN. Prin GOSUB se apelează subrutina, iar prin RETURN se revine din subrutină în program.

Instrucțiunea GOSUB provoacă lansarea în execuție a unei subrutine.

Formatul general este:

GOSUB nr. linie

unde „nr. linie“ este numărul primei linii din subrutină.

GOSUB se comportă ca o instrucțiune GOTO cu excepția faptului că memorează și numărul de ordine al liniei imediat următoare ei, de unde se va relua programul după execuția totală a subrutinei.

Instrucțiunea RETURN realizează revenirea în program și marchează sfârșitul subrutinei.



Fig. 9.53.

Fig. 9.54.

Formatul general este:

RETURN

(fără nici un argument).

Liniile program cuprinse între linia cu numărul de ordine specificat în GOSUB și linia RETURN alcătuiesc subrutina.

Cînd controlul este transferat unei subrutine ea se execută pînă la întîlnirea unui RETURN, după care se revine la linia imediat următoare unui GOSUB.

Transferul controlului generat de GOSUB și RETURN, în cazul unui program ce afișează valoarea unei variabile care este furnizată de o subrutină simplă (formată numai dintr-o singură linie program, linia cu numărul de ordine 40) se vizualizează prin schema (figura 9.53)

O subrutină formată din mai multe linii program se vizualizează ca în figura (9.54)

Nu există posibilitatea declarării explicite a unei subrutine.

Se spune că o subrutină nu are un nume și nu este recunoscută prin el, așa cum se întîmplă în alte limbaje de programare, ci numai prin numărul specificat în instrucțiunea GOSUB; numărul acesta reprezintă numărul de ordine al primei linii program a subrutinei, iar sfîrșitul este considerat ca fiind prima linie program întîlnită în parcurgerea secvențială și crescătoare, ce conține o instrucțiune RETURN.

Subrutinele sînt recursive (se pot folosi de mai multe ori în programul în care au fost definite). Un program poate conține mai multe subrutine. O subrutină poate fi plasată oriunde în corpul programului.

EXEMPLE:

Programul 1 calculează valoarea expresiei

$$\sqrt{a} + \sqrt{a + \sqrt{a}}$$

cînd „a” se introduce mereu de la tastatură.

Programul principal este format din liniile 10—50; se introduce o valoare pentru „a” (linia 10) și se tipărește valoarea lui „a” și rezultatul expresiei pe același rînd, după care se lasă un rînd liber (linia 50) pentru că programul se reia automat și urmează tipărirea altei valori a lui „a” și un alt rezultat pentru expresie ș.a.m.d.

Subrutina din acest exemplu este apelată o singură dată la parcurgerea programului principal și este formată din mai multe linii program (70—100); ea conține toate calculele necesare în vederea obținerii unei valori pentru expresia aleasă.

```
10 INPUT a
20 GOSUB 70
30 PRINT "a="; a,
40 PRINT "expresie="; r
50 PRINT
60 GO TO 10
70 LET b=SQR(a)
80 LET c=SQR(a+b)
90 LET r=b+c
100 RETURN
```

Programul 2 constă în stabilirea naturii rădăcinilor a 10 ecuații de gradul doi numai în cazul în care toți coeficienții sînt pozitivi (a, b, c > 0).

În programul principal se introduc valorile coeficienților (variabilele a, b, c) pe linii diferite, iar în „1” se contorizează de cîte ori a fost reluată execuția întregului program (s-a stabilit inițial că de 10 ori) și se tipăresc diferite mesaje potrivit semnului discriminantului (valoarea expresiei $b^2 - 4 \cdot a \cdot c$).

Subrutina se apelează de trei ori la o singură parcurgere a programului principal și este alcătuită din mai multe linii program (liniile 80—100), prin ea executându-se tipărirea coeficienților ecuațiilor.

```
10 LET i=0
20 LET i=i+1
30 IF i > 11 THEN STOP
40 INPUT a'b'e
50 LET d=b↑2-4*a*c
60 IF d > 0 THEN GOSUB 90 : PRINT „rad. reale distincte“: GO TO 20
70 IF d=0 THEN GOSUB 90 : PRINT „rad. reale egale“: GO TO 20
80 GO SUB 90: PRINT „rad. complexe“: GO TO 20
90 PRINT "a="; a; "b="; b; "c="; c
100 PRINT
110 RETURN
```

9.36. Instrucțiunea PEEK

Instrucțiunea PEEK oferă utilizatorului accesul la memoria fizică.

Formatul general este:

PEEK adr

unde „adr“ este o valoare ce reprezintă adresa locației de memorie ce urmează să fie citită.

EXEMPLU:

Programul următor vizualizează primii 21 octeți din memoria ROM și adresele lor.

```
10 PRINT "Adresa"; TAB 10; "Octet"
20 FOR a=0 TO 20
30 PRINT a TAB 10; PEEK a
40 NEXT a
```

9.37. Instrucțiunea POKE

Instrucțiunea POKE permite utilizatorului scrierea în memoria fizică.

Formatul general este:

POKE adr, val

unde

- „adr“ este adresa locației de memorie;
- „val“ este valoarea ce se atribuie locației de memorie „adr“.

Argumentele instrucțiunii POKE sînt numere sau expresii numerice.

Instrucțiunile POKE și PEEK sînt complementare: prima introduce în memorie o valoare, a doua extrage din memorie acea valoare.

EXEMPLU:**Linia****10 POKE 31000,57**

Incarcă la adresa 31000 valoarea 57, iar linia

20 PRINT PEEK 31000

afișează valoarea 57 găsită în memorie la adresa 31000.

Un alt exemplu de utilizare se referă la faptul că sunetele din gama medie sînt cele mai plăcute la redare, iar sunetele grave se percep ca niște pătănituri. Ele se pot prelungi cu ajutorul instrucțiunii POKE pentru a deveni mai naturale, astfel:

POKE 23609, munde $m=0, \dots 255$.

9.38. Instrucțiunea FLASH

Calculatorul HC-85 oferă diferite facilități în ceea ce privește culoarea ecranului sau a modului de afișare. Pe un televizor alb negru culorile corespund unor tonuri de gri ordonate de la închis la deschis. Orice caracter are asociate 2 culori: culoarea caracterului propriu-zis și culoarea fondului.

La pornirea calculatorului sistemul lucrează cu caractere negre pe ecran alb. Tipărirea poate fi făcută normal, dar există și posibilitatea să apară pe ecran pîlpiind. Pîlpiirea se obține inversînd continuu culoarea caracterului cu culoarea fondului.

Instrucțiunea FLASH realizează pîlpiirea.

Formatul general este:

FLASH nr.

unde „nr“ este 0 sau 1 (0 pentru normal, 1 pentru pîlpiire).

9.39. Instrucțiunea PAPER

Culorile realizate cu HC-85 sînt în număr de 8 și sînt numerotate de la 0 la 7 conform tastelor numerice pe care se găsesc.

Lista culorilor este următoarea:

- 0 — negru
- 1 — albastru
- 2 — roșu
- 3 — mov (magenta)
- 4 — verde
- 5 — bleu
- 6 — galben
- 7 — alb

Atributele de culoare sînt asociate unui caracter, deci ele corespund celor $8 \times 8 = 64$ puncte (pixeli). Nu este posibil ca într-un caracter să se găsească mai mult de două culori (culoarea fondului și culoarea cu care se scrie).

Valorile acestor atribute pot fi modificate pe parcursul execuției unui program.

Instrucțiunea PAPER stabilește culoarea fondului (sau culoarea hîrtiei).

Formatul general este:

PAPER nr.

unde „nr.” este un număr cuprins între 0 și 9.

Cu numerele de la 0 la 7 se stabilește culoarea dorită.

Folosirea numărului 8 face ca fondul să ia culoarea caracterului anterior tipărit, iar numărul 9 indică contrastul dintre fond și ceea ce se afișează.

9.40. Instrucțiunea INK

Instrucțiunea INK stabilește culoarea caracterelor care se afișează pe ecran (sau culoarea cernelii).

Formatul general este:

INK nr.

unde „nr.” este un număr cuprins între 0 și 9.

Folosirea numerelor de la 0 la 7 și 9 au aceeași semnificație ca pentru PAPER. În cazul folosirii numărului 8 caracterul ia culoarea fondului.

EXEMPLE:

Programul 1. Desenează spații divers colorate prin folosirea instrucțiunilor PAPER și INK.

```
20 FOR n=1 TO 10
30 FOR c=0 TO 7
40 PAPER c: PRINT " "; : REM spații colorate
50 NEXT c: NEXT n
60 PAPER 7
70 FOR c=0 TO 3
80 INK c: PRINT c; " ";
90 NEXT c: PAPER 0
100 FOR c=4 TO 7
110 INK c: PRINT c; " ";
120 NEXT c
130 PAPER 7: INK 0
```

Programul 2

```
10 INK 9
20 FOR c=0 TO 7
30 PAPER c
40 PRINT c
50 NEXT c
```

face să contrasteze culoarea cernelii cu cea a hirtiei; cînd una are culoarea negru, albastru, roșu sau mov cealaltă are culoarea verde, bleu, galben, alb.

Programul 3 se obține prin completarea programului 2 cu liniile:

```
60 INK 9
70 PAPER 8
80 PRINT AT 0, 0; FOR n=1 TO 100
90 PRINT n
100 NEXT n
```

În această situație, culoarea cernelii este făcută mereu să contrasteze cu vechea culoare pe care o avea fondul, în fiecare poziție de caracter.

9.41. Instrucțiunea INVERSE

Instrucțiunea INVERSE inversează culoarea fondului cu culoarea cernelii. Datorită acestui efect INVERSE este folosită la ștergerea unor puncte, drepte etc. (atenție la respectarea direcției și sensului) prin redesenarea acestora cu aceeași culoare ca cea a fondului ecranului.

Formatul general este:

INVERSE 1

EXEMPLU:

Utilizînd programul

```
10 PLOT 0, 0
20 DRAW 255, 175
30 PLOT 0, 0
40 DRAW INVERSE 1; 255, 175
```

se constată că diagonala trasată de liniile 10 și 20 este ștearsă prin suprapunere cu diagonala trasată de liniile 30 și 40 care are aceeași culoare cu fondul.

9.42. Instrucțiunea BORDER

Instrucțiunea BORDER stabilește culoarea marginii ecranului.

Formatul general este:

BORDER nr.

unde „nr.” este un număr cuprins între 0 și 7.

Marginea ecranului nu are posibilitatea de pilpire și este de luminozitate normală.



Fig. 9.55.

9.43. Instrucțiunea POINT

Instrucțiunea POINT arată dacă un pixel are culoarea cernelii sau culoarea fondului.

Formatul general este:

POINT (x, y)

unde „x” și „y” sînt coordonatele unui punct.

Instrucțiunea furnizează valorile:

- a) 0 dacă punctul are culoarea fondului (PAPER);
- b) 1 dacă punctul are culoarea cernelii (INK).

9.44. Instrucțiunea ATTR

Instrucțiunea ATTR furnizează un număr în care sînt înglobate atributele unui caracter.

Formatul general este:

ATTR (x, y)

unde „x” și „y” sînt coordonatele unui punct.

Instrucțiunea furnizează o valoare care este suma a patru numere după cum urmează:

- a) 128 dacă poziția pilpiie, 0 dacă este stabilă;
- b) 64 dacă poziția este strălucitoare, 0 dacă este normală;
- c) $8 * n$ unde n este culoarea fondului;
- d) m unde m este culoarea cernelii.

9. INSTRUCȚIUNI BASIC

EXEMPLU: ,

Pentru o poziție plpplitoare, normală cu fond galben și cerneala albastră, ATTR furnizează următoarea valoare:

$$128 + 0 + 8 * 6 + 1 = 177$$

9.45. Instrucțiunea LPRINT

Instrucțiunea LPRINT are același format și efect ca PRINT, cu deosebirea că tipărirea se face la imprimantă.

9.46. Instrucțiunea LLIST

Instrucțiunea LLIST are același format și efect ca LIST, cu deosebirea că listarea se face la imprimantă.

9.47. Instrucțiunea COPY

Instrucțiunea COPY tipărește la imprimantă o copie a ecranului.

LIST urmat de COPY (=LLIST) generează listarea la imprimantă. Tipărirea la imprimantă poate fi oprită în orice moment prin acționarea tastei BREAK.

COPY nu are efect în cazul listărilor automate (o listare automată se obține tastând CR după sfârșitul programului).

PROGRAMAREA ÎN LIMBAJUL LOGO PE CALCULATORUL HC-85

Capitolul 10.

Caracteristicile și elementele limbajului LOGO

În acest capitol se prezintă câteva aspecte prin care se arată că LOGO este un limbaj adecvat pentru utilizarea calculatoarelor în procesul de predare-învățare. În legătură cu calculatoarele, aproape toți oamenii au auzit de viteza și precizia cu care acestea execută șiruri lungi de operații complicate, obositoare pentru om. Retribuția, telefonul, plata energiei electrice consumate, etc., ne vin de la calculator pe bucăți de hîrtie imprimată.

Se pare că putem accepta că, azi, calculatoarele fac parte din viața noastră cotidiană. Dar nu toți sînt mulțumiți în situația aceasta, mai ales atunci cînd calculatoarele, dar mai ales operatorii și programatorii, acestora comit erori. Trebuie înțeles însă un lucru fundamental și anume că utilizarea calculatoarelor este o activitate creativă desfășurată de om.

Orice activitate creativă implică gîndire productivă, rezolvare de probleme, imaginație. Cu ajutorul calculatoarelor activitatea creativă cîștigă un plus de utilitate, originalitate și adaptare la realitate.

Calculatorul face ceea ce face, pentru că cineva l-a instruit ce să facă și cum să facă. Cine poate să-și exprime gîndurile în limbajul calculatorului a dobîndit o nouă cale de a gîndi, scrie și comunica, a descoperit noi modalități de rezolvare a problemelor. Apariția calculatoarelor personale, accesibile și nespecialiștilor în domeniu, este adesea comparată cu apariția materialelor tipărite, cu cîteva secole în urmă. Impactul major pe care tehnologiile tipografice l-au avut asupra vieții de zi cu zi s-a produs abia după alfabetizarea mării majorități a populației. Mulți, altfel bine pregătiți în diferite domenii, consideră calculatoarele niște mașini sofisticate și ostile comunicației directe, cu care se poate comunica numai prin intermediul unor „învățați“ (care știu să „scrie“ și să „citească“ în limbajul calculatorului). Realitatea este însă alta.

Milioane de oameni pot azi comunica direct cu calculatoarele, descoperind noi utilizări ale acestora, pe care nu și le-ar fi putut imagina înainte. Dar ca și lumea înconjurătoare, calculatorul este ceea ce face, nu ceea ce spunem sau credem despre el. Nimic nu poate fi mai convingător decât „făcând, văzînd și crezînd ceea ce se vede”. Pentru aceasta este necesară învățarea unui limbaj de programare.

Oamenii au inventat o mare varietate de limbaje de programare.

10.1. Evoluția limbajelor de programare

Apariția și dezvoltarea limbajelor de programare este strîns legată de evoluția structurii și arhitecturii calculatoarelor. Desigur că viteza de efectuare a operațiilor elementare este un indicator de performanță important, dar din punctul de vedere al limbajelor de programare, capacitatea memoriei interne a calculatorului este mai importantă. Datorită costului ridicat și a posibilităților tehnologice din perioada respectivă, calculatoarele din primele generații aveau o memorie internă de capacitate redusă.

Aceasta a impus ca limbajele de programare să fie simple pentru calculator, chiar cu prețul ca acestea să fie greu de utilizat, neprietenose pentru programatori. Astfel, programatorul trebuia să țină seama de o serie de restricții și precizări privind numele variabilelor utilizate, tipul, structura și dimensiunea datelor prelucrate. Prin convenții implicite (numele variabilelor întregi să înceapă cu una din literele I, J, K, L, M, N sau numele variabilelor de tip șir să se termine cu caracterul \$, variabilele de tip tablou să fie formate din cel mult două caractere începînd cu o literă) și prin declarații explicite compilatorul poate determina numărul de celule de memorie necesar fiecărei variabile, facilitînd astfel alocarea și gestiunea spațiului de memorie limitat. Există și păreri că aceste constrîngerii sînt de fapt avantaje prin faptul că obligînd programatorul să declare tipurile de date ale variabilelor, dimensiunea structurilor de date utilizate, etc., îl obligă pe acesta să fie mai ordonat, mai meticolos în conceperea și scrierea programelor. După alte păreri, aceste neplăceri constituie ceva intrinsec activității de programare.

Mai tîrziu, pe la sfîrșitul anilor 1970, odată cu apariția calculatoarelor personale s-a lărgit considerabil accesul la facilitățile oferite de tehnica de calcul. S-a accelerat astfel schimbarea relației noastre față de cultură, constituită în principal din artă și literatură, avînd un caracter descriptiv, static. Știința în general, informatica în special, oferă posibilitatea unor reprezentări ale lumii înconjurătoare, mai dinamice și predictive, utilizînd legi și modele specifice. Informatica, la fel ca matematica, are o parte de interes comun, dar în același timp fiecare ramură a științei trebuie să-și dezvolte propria ramură a informaticii. Se impune astfel o pregătire de bază, fundamentală în domeniul informaticii utilizînd calculatoarele ca mijloace de realizare a acestui scop.

Trebuie înțeles că însușirea unor „scheme” sau „rețete” de programare nu înseamnă educație în calculatoare. Deși impactul calculatoarelor personale este uriaș în crearea a ceea ce am putea numi o nouă ramură a culturii universale, informatica, trebuie să observăm că acestea au promovat confuzia că ceea ce

este simplu pentru calculator este simplu și pentru utilizatori. Astfel, limbajul BASIC, care constituie „limba maternă“ a calculatoarelor personale este ușor de implementat chiar pe calculatoarele personale cu memorie foarte mică (8—16 Kocteti). Avind un vocabular relativ redus de cuvinte cheie BASIC este ușor de învățat, dar extinderea acestuia cu noi cuvinte cheie, care corespund mai bine necesităților programatorului la un moment dat, este dificilă. Puterea de expresie scăzută a limbajului îl fac greu de utilizat și nu oferă facilități deosebite programatorilor începători pentru a scrie programe simple care fac lucruri interesante (spectaculoase).

Pe de altă parte, oamenii au inventat peste o sută de limbaje de programare și se poate spune că într-un anumit sens toate sînt la fel. Fiecare are un vocabular de aproximativ o sută de cuvinte și simboluri. Fiecare limbaj este bazat pe o gramatică strict definită, fără excepții. „Propozițiile“ dintr-un limbaj pot fi ușor traduse în „propoziții“ similare dintr-un alt limbaj.

Astfel, dacă o problemă poate fi rezolvată într-un limbaj de programare oarecare, ea poate fi rezolvată cumva și în alt limbaj. La ora actuală cei mai mulți programatori „vorbesc“ în BASIC cu calculatoarele lor, datorită răspîndirii calculatoarelor personale. Dar cele mai multe linii de program s-au scris, probabil, în COBOL, un limbaj adecvat aplicațiilor economice.

Marea majoritate a programelor științifice și ingineresti sînt scrise fără îndoială în FORTRAN, unul dintre cele mai vechi limbaje de programare a calculatoarelor. Cînd IBM, cea mai mare firmă producătoare de calculatoare, anunța lansarea limbajului PL/1, s-a spus că acesta o să înlocuiască FORTRAN-ul și COBOL-ul deoarece face tot ce fac acestea, și chiar mai mult. Cei care lucrează în zona inteligenței artificiale (IA) nici nu concep utilizarea altui limbaj decît LISP sau PROLOG.

Esențial este însușirea unui mod de a comunica cu calculatorul, indiferent de limbajul utilizat. Nu se pune problema de a găsi un limbaj „bun“ care să le înlătore pe cele „rele“. Limba Esperanto este mai universală decît limba română, dar Eminescu a scris în limba română ceea ce nimeni nu a scris în Esperanto. Diferențele dintre limbajele de programare sînt relativ mici dar suficiente pentru a justifica o anumită opțiune. Diferența principală dintre diferite limbaje constă în faptul că un anumit tip de probleme se rezolvă mai ușor într-un limbaj decît în altele. Dar care trebuie să fie facilitățile unui limbaj adecvat rezolvării problemelor specifice procesului de învățare?

În continuare sînt prezentate cîteva dintre acestea:

- să fie simplu și puternic în același timp, astfel ca programe simple să facă lucruri interesante, stimulîndu-î astfel pe începători;
- să ofere facilități pentru învățarea experimentală, prin descoperire, mai bine corelată cu gîndirea intuitivă;
- să ofere posibilitatea utilizării puterii de calcul pentru reformularea unor idei tradiționale din matematică și alte științe, pentru a le face mai accesibile oamenilor cu aptitudini matematice dintr-o gamă mai largă;
- să faciliteze controlul individual asupra unor resurse de calcul puternice, utilizate ca unelte universale pentru învățare, recreere și explorare;
- să faciliteze dezvoltarea unor deprinderi în rezolvarea problemelor în general;
- să faciliteze însușirea principiilor programării structurate;
- să constituie un mediu de învățare pentru anumite discipline: matematică, fizică, chimie, biologie, literatură, muzică, etc.

— să ofere posibilitatea de a învăța celor care, dintr-un motiv sau altul nu au succes în clase cu program tradițional;

— să ofere o bază pentru un tip nou de școală, bazată pe principiile Piagetiene de predare-învățare (vezi glosarul de termeni din paragraful 10.10) utilizând calculatoarele ca elemente universale în procesul de învățare;

În continuare se va prezenta LOGO, unul dintre limbajele care răspund în mare măsură acestor cerințe.

10.2. Ce este LOGO? Caracteristici.

În general prin LOGO se înțelege denumirea unui limbaj de programare. Creatorii acestui sistem, denumit LOGO, afirmă însă că aceasta este denumirea unei filozofii privind educația și a unei familii de limbaje de programare în continuă evoluție care constituie suportul pentru implementarea acestei filozofii.

LOGO este un limbaj pentru învățare în general. Mai concret LOGO este pe de o parte, un limbaj de învățare a programării calculatoarelor, iar pe de altă parte, un limbaj de învățare a unui mod de gândire.

Istoric vorbind LOGO își are rădăcinile atât în cercetările din știința calculatoarelor, în special în domeniul inteligenței artificiale, cit și în studiile și cercetările lui Jean Piaget privind dezvoltarea aptitudinilor de a gândi la copii.

LOGO a fost inițial creat de firma Bott, Beranek & Newman, în Cambridge, Massachusetts în anul 1968. De atunci a cunoscut o continuă dezvoltare prin cercetările efectuate la Laboratorul de Inteligență Artificială și la Departamentul de Studii și Cercetări în Educație de la Institutul de Tehnologie din Massachusetts (MIT), precum și alte universități din întreaga lume. Cercetările au fost efectuate utilizând cele mai puternice calculatoare disponibile în acea perioadă.

În 1979 grupul LOGO de la MIT a început activitatea de implementare a limbajului pe calculatorul personal TI99/4 sub conducerea lui Seymour Papert și pe calculatorul personal Apple II sub conducerea lui Harold Abelson. Limbajul a fost apoi implementat și pe alte calculatoare personale. Nu trebuie să se facă confuzia între limbaj și una din aceste implementări supuse unor restricții datorită limitelor performanțelor calculatoarelor personale.

Dar pentru a răspunde mai bine la întrebarea, „Ce este LOGO”, vor fi arătate în continuare, principalele caracteristici prezentate de LOGO privit ca limbaj de programare a calculatoarelor.

LOGO este un limbaj procedural. Programele în LOGO sînt create prin asocierea comenzilor în grupuri denumite proceduri și utilizarea procedurilor drept comenzi în alte proceduri, procesul putînd continua pe oricîte nivele. Fiecare pas într-o procedură poate fi comandă elementară din LOGO sau o procedură definită de utilizatori. Aceasta oferă posibilitatea implementării (și a însușirii) unei strategii foarte puternice de rezolvare a problemelor: spargerea problemei în părți din ce în ce mai mici și scrierea unei proceduri pentru fiecare parte. Procedurile pot comunica între ele prin intrări și ieșiri.

LOGO este interactiv. Orice comandă LOGO fie că este predefinită în limbaj sau este definită de programatori ca o procedură, poate fi executată prin simpla introducere de la tastatură a numelui acesteia.

Introducerea, ștergerea sau schimbarea unei linii de program este simplă.

Editorul integrat în LOGO facilitează definirea, execuția sau modificarea procedurilor fără a utiliza programe separate de încărcare, editare, compilare, etc. Eficiența unui limbaj diferă pentru limbajele interactive (LOGO, BASIC, LISP, APL), față de cele neinteractive (FORTRAN, PASCAL, MODULA, C).

Dezvoltarea și punerea la punct a programelor este mai eficientă în cazul limbajelor interactive, dar un program pus la punct se execută, mai eficient în cazul limbajelor neinteractive.

Orice program scris într-un limbaj de nivel înalt (interactiv sau neinteractiv) trebuie tradus într-un limbaj specific calculatorului, denumit limbaj mașină. Această traducere este efectuată de un alt program denumit interpretor pentru limbajele interactive și compilator pentru cele neinteractive.

Compilarea este un proces complex, care durează mult, dar programul rezultat, când este corect, este obținut într-o formă binară „pură“ (în limbaj mașină) și execuția lui ulterioară este rapidă (nu mai este nevoie de compilator). Prin contrast am putea spune că interpretorul face această traducere linie cu linie, de fiecare dată când se execută programul. De fapt, interpretorul nu traduce programul în formă binară ci el însuși execută pașii necesari în limbaj mașină pentru fiecare pas din programul în limbaj superior.

Cele mai multe interpretoare, inclusiv cel pentru LOGO, produc (la prima execuție) o formă intermediară (o traducere parțială) care face ca execuțiile ulterioare să fie mai rapide.

Limbajele interpretate pot fi interactive. Limbajele interactive oferă și facilități de depanare interactivă a programelor ceea ce duce la creșterea productivității programatorului. De altfel pentru specificul activității de învățare (a programării) un program care nu mai are pene este neinteresant. În general este abandonat și se demarează un nou proiect.

LOGO este un limbaj recursiv. Într-un limbaj procedural o procedură poate apela o altă procedură (denumită subprocedură) pentru a efectua o parte din sarcinile sale. Un limbaj este recursiv dacă permite ca o procedură să fie subprocedură pentru ea însăși. Puterea de expresie a acestui concept constă în faptul că permite ca o problemă complicată să fie descrisă în termenii unei versiuni mai simple a ei însăși ceea ce conduce la posibilitatea de descriere (în limbaj recursiv) într-o formă foarte compactă a unor probleme foarte complexe. Recursivitatea este unul dintre conceptele greu de înțeles din programare, asupra căruia se va reveni, cu exemplificări, în capitolul următor.

Logo este extensibil. Într-un anumit sens, toate limbajele procedurale sînt extensibile deoarece ele permit extinderea facilităților limbajului cu noi operații, pentru care s-au scris procedurile adecvate.

În mod strict însă, un limbaj este extensibil dacă procedurile definite de utilizator „arată la fel“ ca procedurile primitive, predefinite ale limbajului.

LOGO prezintă o singură abatere de la această regulă de extensibilitate. Acesta constă în faptul că unele operații aritmetice primitive (adunare, scădere, ...) pot fi scrise în forma infixată, cu operatorul între cei doi operanzi (Ex: $5 + 3$), iar operațiile noi, definite de utilizator (prin proceduri adecvate), trebuie neapărat să fie în forma prefixată, operatorul precede operanzii (Ex: SUM 5 3). Asupra modului în care se scrie și se citește o linie în LOGO se va reveni în capitolul următor.

În ceea ce privește obiectele reprezentînd date (acele entități care pot fi denumite cu nume de variabile independente, pot fi transmise ca intrări sau ieșiri ale unor proceduri), LOGO are câteva caracteristici distinctive.

10. CARACTERISTICI LOGO

LOGO nu este orientat pe tipuri de date. În cele mai multe limbaje tipul unui obiect reprezentat printr-o variabilă trebuie specificat explicit prin declarații sau este specificat implicit prin restricții în alcătuirea numelui variabilei (Ex: — să înceapă cu una dintre literele I-N pentru a reprezenta un întreg în FORTRAN, să se termine cu \$ pentru a reprezenta un șir în BASIC). În LOGO ca și în LISP (din care este de fapt inspirat) variabilele nu au asociat un anumit tip. Orice variabilă poate lua orice valoare. Aceeași variabilă poate reprezenta un întreg la un moment dat și un cuvânt (un șir de caractere) la alt moment de timp dat. Inițial variabilele au fost tipizate pentru a ușura munca descriere a compilatoarelor care trebuie să stabilească corespondența dintre funcțiile primitive din limbajul de nivel înalt și cele din limbaj mașină. Deoarece în limbaj mașină adunarea unor întregi se face cu altă operație decât adunarea numerelor reale, iar operațiile în simplă precizie sînt diferite de cele în precizie extinsă, traducerea în limbaj mașină este mai simplă dacă limbajul superior obligă la aceleași restricții. Mai tîrziu, filozofînd pe marginea limbajelor de programare, s-a emis ideea că tipizarea datelor are marele avantaj că obligă programatorul să fie mai ordonat în utilizarea variabilelor (o variabilă este utilizată pentru un singur scop într-un program).

LOGO realizează folosirea disciplinată a variabilelor, prin utilizarea unei variabile pentru un singur scop într-o manieră diferită.

LOGO este un limbaj procedural în care variabilele (netipizate) sînt asociate unei proceduri și nu întregului program. Asupra acestui aspect se va reveni în capitolul următor.

LOGO permite prelucrarea listelor. Limbajele de nivel înalt și unele limbaje de asamblare mai evaluate permit gruparea datelor și structurilor de complexitate mai ridicată (tablouri, înregistrări, structuri). În limbajele BASIC și FORTRAN de exemplu se pot defini tablouri de variabile prin declarații în care se specifică numele și dimensiunea acestora. În LOGO se pot defini liste, ca modalitate principală de grupare a datelor în structuri complexe, structurile de date din LOGO fiind inspirate din LISP.

Listele se deosebesc de tablouri atît prin structură cît și prin conținut. Tablourile sînt structuri fixe, statice cu dimensiunile fixate prin declarații la începutul programului. Prin contrast, listele au o structură dinamică, structura lor nu trebuie specificată inițial, ci poate evolua după necesități, în timpul execuției programului. În ceea ce privește conținutul, tablourile au un conținut omogen (conțin numai date de același tip). Pot fi definite tablouri de numere întregi, și numere reale, de informații alfanumerice, dar separat. În același tablou nu pot fi definite date de tipuri diferite. În Logo listele nu sînt tipizate. Fiecare element dintr-o listă poate conține orice obiect acceptat de limbaj, un număr (întreg sau real), un cuvînt (șir de caractere alfanumerice) sau o altă listă (și aceasta pe oricîte nivele). Listele oferă o flexibilitate mult mai mare în structurarea datelor după necesitățile fiecărui proiect, dar accesul la un element oarecare al structurii se face mult mai ușor la tablouri. Cunoșcînd adresa de început a tabloului și faptul că fiecare element ocupă același număr de celule de memorie se ajunge imediat la elementul dorit. Memorarea listelor, avînd în vedere că au o structură dinamică și că fiecare element poate ocupa un număr diferit de celule de memorie, se face într-o manieră mai complicată. Pentru a ajunge la un element oarecare din listă, aceasta trebuie parcursă element cu element, într-un sens sau altul. Căutarea unui element într-o listă este mai lentă, dar nu este mai dificilă pentru programator decît căutarea într-un

tablou. Alte limbaje, cum ar fi PASCAL, MODULA, C, nu pot implementa liste direct, dar cu ajutorul variabilelor de tip indicator (pointer) se pot prelucra structuri de tip listă. Din considerente metodologice am privit LOGO din două puncte de vedere: limbaj de programare și limbaj de învățare. Ca limbaj adecvat procesului de predare — învățare LOGO are câteva caracteristici distincte dintre care vor fi prezentate câteva în continuare.

Este bazat pe ideea lui Jean Piaget că pînă și copiii (chiar mici) au teorii. Procesul de predare-învățare nu este o chestiune de a fi bun sau rău. Învățarea și predarea se împletesc într-un proces continuu de „depanare“, de umplere a golurilor în cunoaștere de îndată ce sînt descoperite. LOGO facilitează învățarea prin descoperire. Penele din programele în LOGO nu trebuie privite neapărat ca greșeli; analizate, acestea pot sugera sau chiar pot duce la rezultate mai interesante decît obiectivele propuse inițial. Prin analiza empirică a sute de proiecte de utilizare a calculatoarelor în școli s-a ajuns la concluzia că prin învățarea și exersarea unui limbaj de programare (în special LOGO) elevii dobîndesc deprinderi cognitive cum ar fi:

- gândire riguroasă și posibilitate de exprimare cu acuratețe și precizie a ideilor;

- însușirea unor metode euristice cum ar fi planificarea activităților, de compoziția unei probleme în subprobleme, alcătuirea unor modele de reprezentare, stabilirea unor analogii;

- abilitate în descoperirea și înlăturarea erorilor din soluțiile la o problemă dată;

- înțelegerea faptului că cele mai multe probleme au mai multe strategii de elaborare a soluțiilor;

- înțelegerea și posibilitatea de utilizare a unor concepte de bază cum ar fi: procedură, funcție, variabilă, recursivitate, etc.

Exprimarea în LOGO este clară, explicită. LOGO diferă de LISP, în care își are rădăcinile, prin caracteristici rezultate din efortul de a-l face cît mai adecvat procesului de predare — învățare. Ideile fundamentale din programare sînt explicite. Astfel procedurile sînt împărțite în operații și comenzi. Operațiile sînt proceduri care calculează (generează) diferite valori transmise la ieșire și care pot fi apoi preluate de alte proceduri. Spre deosebire de operații, comenzile sînt proceduri care nu transmit ceva la ieșire ci au un efect imediat (vizibil, într-un fel), de ex: trasează un segment de dreaptă, afișează pe ecran o valoare, un text, sau modifică valoarea unei variabile. Prin notații adecvate se elimină un mare număr de confuzii în identificarea procedurilor, variabilelor și a valorilor acestora.

Astfel, orice nume format din caractere alfanumerice fără semne speciale reprezintă o procedură (X reprezintă o procedură, „X reprezintă numele variabilei X iar :X reprezintă valoarea variabilei X). Unei variabile i se atribuie o valoare în mod clar, explicit, utilizînd operatorul MAKE care sugerează o schimbare. Astfel $X = X + 1$ pare aberant din punct de vedere algebric. În LOGO aceasta se scrie MAKE „X :X + 1 adică variabila cu numele X, notată „X va primi (va fi făcută egală cu) valoarea actuală a lui X, notată :X plus 1.

LOGO este stimulatîv. Cele mai multe limbaje de programare suferă prin lipsa facilităților de rezolvare cu ușurință a unor probleme interesante, chiar de către începători. Astfel, deseori un curs de programare pentru începători începe cu rezolvarea și discuția ecuației de gradul doi și continuă apoi cu operații cu matrici, etc.

10. CARACTERISTICI LOGO

Unul dintre aspectele cele mai atractive privind programarea în LOGO constă în facilitățile grafice și de prelucrare a textelor în limbaj natural. Utilizând un instrument conceptual denumit *penel* (cu sensul cunoscut pentru cei ce iubesc pictura, sau cu sensul de *peniță electronică* pentru cei fascinați de tehnică) și operațiile și comenzile de lucru cu penelul, se pot obține desene complexe, se pot rezolva probleme de geometrie sau se pot simula fenomene și legi din fizică într-o manieră directă și atractivă. Limbajele de programare consideră în general textele ca șiruri de caractere. În LOGO, literele, cuvintele și propozițiile sînt obiecte cu o ierarhie naturală, ceea ce oferă facilități interesante de analiză a limbajului uman. Între multiplele proiecte și experimente privind utilizarea lui LOGO în școli, prezentate în literatura de specialitate se află și lucrul cu diferite categorii gramaticale. O elevă de 13 ani încercînd să scrie un program care compune poezii a făcut o descoperire interesantă, exprimată astfel: abia acum am înțeles „de ce avem substantive și verbe“.

Deși nu părea să aibă dificultăți în lucrul cu categoriile logice avea probleme în înțelegerea gramaticii, a diferenței dintre substantive verbe și adverbe.

Nu înțelegea de fapt la ce este bună gramatica. Încercînd să scrie programul a constatat că ea însăși clasifică cuvintele în categorii, nu pentru că cineva i-a cerut acest lucru ci pentru că trebuia.

Ea trebuia să învețe calculatorul să aleagă cuvintele, din clasa corespunzătoare. Ea a înțeles că și cuvintele, ca și lucrurile în general, pot fi plasate în diferite grupuri, mulțimi, în conformitate cu obiectivele sale. Experiența ei a fost profundă și plină de sens. Nu numai că a înțeles gramatica, dar și-a schimbat relația față de aceasta.

Pe parcurs vor fi prezentate mai multe exemple de programe de acest gen.

LOGO este adecvat în diferite etape ale procesului de predare învățare. LOGO este în general privit ca un limbaj pentru copii deoarece cei mai mulți au un model al procesului de învățare, ca un proces în care numai cei mici învață. Desigur, au fost proiecte de grafică cu penelul în LOGO pentru copii de 4 ani, utilizînd o claviatură adecvată, cu butoane mari, marcate cu poze în loc de cuvinte. S-a introdus chiar și conceptul de procedură, existînd butoane de „start memorare“ și „stop memorare“ pentru definirea unei proceduri. Pentru a defini mai multe proceduri s-au utilizat butoane de diferite culori. Dar profesorul Harold Abelson a utilizat la MIT facilitățile de grafică cu penelul pentru predarea fizicii la facultate pentru a demonstra principiul din mecanica newtoniană și din teoria generală a relativității. Deasemenea, LOGO a fost utilizat pentru grupuri de elevi cu inteligența normală sau superioară, dar handicapați în ceea ce privește posibilitatea de comunicare. Calculatoarele pot fi utilizate, atît ca mediu de comunicare în astfel de situații, cît și ca mijloace de obținere a autonomiei în realizarea unor scopuri.

6.3. LOGO în școli

Implementarea limbajului pe cele mai răspîndite calculatoare personale a permis depășirea stadiului de experimente izolate și trecerea la utilizarea sistematică și generală în procesul de învățămînt în diferite țări ale lumii. Dacă oportunitatea utilizării limbajului în școli este demonstrată prin multiple

proiecte și experimente, modalitățile concrete de implementare sînt diverse și depind de context.

Calendarul de lucru, organizarea accesului la calculatoare, etc., depind în mare măsură de dotare, de asigurarea cu cadre de specialitate, de modul cum principiile solide ale învățămîntului tradițional sînt corelate cu noile posibilități oferite de calculatoare. În orice caz, pentru începători vor trebui fixate următoarele idei:

- programarea este un proces de angajare a calculatorului într-o conversație utilizînd numai cuvinte dintr-un vocabular restrîns pe care îl înțelege;

- acest vocabular poate fi ușor extins prin definirea de noi proceduri care apoi pot fi utilizate exact ca și operațiile și comenzile primitive din vocabularul inițial;

- prin definirea de noi proceduri și denumirea acestora cu noi cuvinte se dă sens acestor cuvinte, care apoi pot fi folosite în definirea altor proceduri, etc.;

- extînderea domeniului de cunoaștere al calculatorului prin noi proceduri implică un proces de definire și depanare a acestora;

- penele descoperite în procesul de punere la punct a programelor trebuie bine analizate și înțelese.

Ele pot constitui idei pentru noi explorări:

- substituindu-se calculatorului sau penelului și parcurgînd pașii respectivi, programatorul poate ajunge mai repede la detectarea și înlăturarea penelor;

- primele programe trebuie să fixeze relația dintre obiectiv și program, apoi dintre obiective și subobiective, proceduri și subproceduri;

- facilitarea schimbului de idei, informații și experiențe între elevi poate duce la rezultate spectaculoase.

Programele de lucru cu penelul sînt adecvate acestei faze de început. Învățarea-programării, ca orice proces de învățare se desfășoară conform cu un stil propriu care reflectă personalitatea intelectuală a elevului. Calculatorul oferă o mai mare flexibilitate fiecărui elev de a progresa în stilul și ritmul propriu, respectîndu-i-se personalitatea. Totuși, observînd anumite similitudini în abordarea programării în LOGO s-au degajat trei stiluri importante de lucru:

- stilul timid, plecînd de la o micro-explorare. Elevii încep cu timiditate încercînd o dată, de două sau mai multe ori aceeași comandă pentru a se convinge că așa este. Ei execută $2+3$, apoi vor să se convingă că $3+2$ duce la același rezultat etc. Numai după aceea trec la o explorare planificată, direcționată de anumite obiective;

- stilul curajos, plecînd de la o macro-explorare. Elevii exersează diferite proceduri, studiază efectele apoi trec la utilizarea acestora în conformitate cu obiective precise;

- stilul organizat, planificat. Elevii abordează orice problemă în mod structural, de jos în sus sau de sus în jos, dar întotdeauna în mod organizat, după un plan riguros.

Aceste stiluri de lucru se regăsesc la orice începător în LOGO, indiferent de vîrstă și ele trebuie încurajate în mod corespunzător, cunoscînd faptul că ele se întrepătrund în mod natural.

Ce trebuie să știe un profesor de LOGO?, este o întrebare la care este dificil de răspuns. În orice caz el trebuie să aibă în vedere următoarele:

- trebuie să dispună de o colecție de proiecte de programare care pun în evidență conceptele și tehnicile de programare în LOGO;

— să dispună de un vocabular adecvat discuțiilor și dezbaterilor despre programare;

— o subtilă cunoaștere a diferitelor stiluri și strategii de învățare a programării, pentru a le putea manipula;

— stăpânirea și contracararea rezistenței pe care unii copii sau adulți o opun lucrului cu categorii matematice și logice.

În final trebuie accentuate faptul că ideea utilizării calculatoarelor în educație nu este susținută pentru motivul ca fiecare să știe ceva despre calculatoare, ci faptul că pentru mulți oameni, programarea calculatoarelor este o modalitate foarte importantă de a învăța să învețe, este un mod de explorare intelectuală prin construcția de modele din ce în ce mai perfecționate.

Calculatorul oferă toate aceste facilități pentru că este o unealtă universală care poate desena, poate să compună muzică sau poezii, sau să conducă mașini unelte sau roboți sofisticăți.

Microcalculatoarele produse în țara noastră HC-85, FELIX PC și calculatoarele personale compatibile cu APPLE II oferă posibilitatea utilizării limbajului LOGO.

10.4. LOGO pe HC-85

Pentru a utiliza LOGO pe HC-85 este nevoie de:

— calculatorul personal HC-85, Sinclair Spectrum cu 48K sau un calculator personal compatibil;

— televizor sau monitor alb-negru sau color;

— casetofon și caseta cu limbajul LOGO sau

— disc magnetic și discul cu limbajul LOGO.

Încărcarea programului se face normal, iar pe ecran apare un mesaj de bun-venit programatorilor în LOGO.

Apoi, pe ecran, la început de rînd LOGO trimite? pentru a anunța utilizatorul că este în regim de așteptare comenzi de la tastatură. Acest regim corespunde nivelului cel mai de sus, în terminologia LOGO (top level). În continuare se prezintă câteva aspecte privind utilizarea tastaturii:

ENTER sau CR — Apăsarea tastei ENTER specifică terminarea introducerii unei linii de program, a unei instrucțiuni etc. (care a început cu?) și comunică limbajului că poate trece la interpretarea ei. O linie de programare în LOGO poate avea pînă la 250 de caractere, deci mai multe linii pe ecran;

BREAK/SPACE — Introduce un spațiu liber;

SS — Prescurtarea pentru tasta SYMBOL SHIFT;

CS — Prescurtarea pentru CAPS SHIFT;

CMODE — Tastînd CAPS și 2 (în același timp) tastatura trece în mod „litere mari“;

LMODE — Tastînd CAPS a doua oară tastatura trece în mod „litere mici“;

EMODE — Tastînd CAPS și SYS tastatura trece în mod editare. În colțul stînga jos al ecranului se afișează caracterul E.

	Cînd se apasă o a treia tastă, SYS trebuie ținută apăsată;
Delete	— Tastînd CS și 0 se șterge un caracter spre stînga;
←	— Tastînd CS și 5 se deplasează cursorul cu un caracter spre stînga, fără să se șteargă nimic;
↓	— Tastînd CS și 6 (în mod editare) se deplasează cursorul în jos cu o linie;
↑	— Tastînd CS și 7 se deplasează cursorul în sus cu o linie;
→	— Tastînd CS și 8 se deplasează cursorul cu un caracter spre dreapta;
SS P	— Tastînd SS și P se afișează apostrof ' ;
SS Y	— Tastînd SS și Y se afișează paranteza dreaptă deschisă [;
SS U	— Tastînd SS și U se afișează paranteza dreaptă închisă] ;
	LOGO tratează în mod diferit parantezele drepte [] și rotunde ().
CS	— BREAK/SPACE — Tastînd CAPS și BREAK/SPACE înainte ca LOGO să termine execuția unei proceduri, forțează intreruperea acesteia și pe ecran apare mesajul: STOPPED!!!
	?
	și LOGO intră în regim de așteptare comenzi (pe nivelul cel mai de sus);

Introducerea de la tastatură a unor proceduri mai simple se poate face direct, utilizînd și tastele cu semnificație specială arătate mai sus.

Pentru editarea unor proceduri complexe definite prin texte lungi se utilizează editorul încorporat în LOGO care facilitează introducerea și ștergerea/correctarea textelor sub controlul interpretorului LOGO.

Mod editare:

Intrarea în mod editare se face tastînd EDIT sau ED urmat de numele procedurii (sau lista de proceduri) ce urmează să fie editată.

Ex: ED "PROC1 sau ED [CERC ELIPSA POLI]

Dacă există deja o procedură cu același nume ea va fi afișată pe ecran pentru a fi corectată, actualizată.

Dacă se introduce EDIT sau ED fără un nume de procedură, editorul acționează asupra textului ce a rămas în zona de memorie rezervată lui, dintr-o editare anterioară.

EDIT [] sau ED [] lansează întotdeauna editorul cu zona de lucru curată (fără nici un text de la vre-o editare prealabilă).

În mod editare caracterul? (promptul) nu mai apare. Cursorul (pătrat care clipește) se deplasează pentru a indica locul unde se va introduce următorul caracter.

Caractere speciale în mod editare:

CS 5	— Mută cursorul la stînga cu un caracter (←);
CS 6	— Mută cursorul în jos cu o linie (↓);
CS 7	— Mută cursorul în sus cu o linie (↑);
CS 8	— Mută cursorul la dreapta cu un caracter (→);
CS 0	— Șterge un caracter spre stînga;
EMODE CS 5	— Mută cursorul la începutul liniei curente;

- EMODE CS 6 — Mută cursorul la sfârșitul ecranului;
- EMODE CS 7 — Mută cursorul la începutul ecranului;
- EMODE CS 8 — Mută cursorul la sfârșitul liniei curente;
- EMODE B — Mută cursorul la începutul textului introdus;
- EMODE E — Mută cursorul la sfârșitul textului introdus;
- Defilare — Dacă textul introdus depășește o pagină de ecran, prin defilare se poate trece la pagina următoare sau la cea precedentă;
- SS S — Oprește defilarea. Orice tastă apăsată apoi pornește defilarea din nou;
- EMODE N — Mută cursorul la pagina următoare;
- EMODE P — Mută cursorul la pagina precedentă;
- EMODE Y — Șterge și salvează linia în care se găsește cursorul;
- EMODE R — Înserează linia salvată cu EMODE Y, în poziția cursorului;
- EMODE C — Se iese din mod editare și se salvează toate modificările efectuate. Editorul afișează un mesaj cu numele procedurii editate;
- CS BREAK/
SPACE — Se iese din mod editare și nu se iau în considerare modificările făcute.

În afara modului editare toate tastele cu funcții speciale de editare pot fi utilizate în cadrul aceleași linii de program LOGO.

Editarea numelor de variabile și a valorii lor se face cu comanda ENDS nume sau ENDS [LISTA de nume]. Fără argument ENDS va afișa toate numele și valorile lor definite până în acel moment.

- TO MODE — Tastind TO se trece în modul TO de definire a unei proceduri. Promptul nu mai este ? ci >. Abandonarea definirii unei proceduri se poate face tastind CAPS BREAK/SPACE. Pentru corecturi se poate utiliza comanda ERASE de ștergere a procedurii și apoi se face redefinirea ei. Ultima linie va conține doar END pentru a marca sfârșitul definirii unei proceduri. Dacă se definesc mai multe proceduri, fiecare trebuie terminată cu END. La definirea ultimei proceduri cu Editorul nu este necesar să se introducă END. La părăsirea editorului, se introduce implicit END.

10.5. Reguli gramaticale ale limbajului Logo

Plecînd de la setul standard de caractere ASCII și funcțiile speciale ale tastaturii calculatorului, se pot introduce în calculator șiruri de caractere interpretate în diferite moduri de către programul care le preia de la tastatură.

Un program LOGO este la prima vedere un șir de caractere organizat în linii mai lungi sau mai scurte, așa cum se vede pe ecranul calculatorului. Dar ca și în limbajele naturale nu orice șir de caractere formează cuvinte cu sens și nu orice înșiruire de cuvinte cu sens formează fraze cu sens. Logo înțelege un set de cuvinte introduse de la tastatură care fac parte din vocabularul de bază,

dar și neologisme al căror sens este transmis calculatorului în cadrul programului.

Deci un program Logo este format din construcții utilizând cuvinte din vocabularul predefinit și construcții care definesc și utilizează cuvinte noi îmbogățind astfel vocabularul. Caracterele și cuvintele din vocabularul de bază și regulile de combinare a acestora în propoziții și fraze formează gramatica limbajului Logo. Orice program pentru a putea fi interpretat corect de către calculator trebuie scris respectând cu strictețe regulile gramaticale care vor fi prezentate în continuare

În fine, oaca în dialogurile dintre noi uneori mai trecem cu vederea anumite greșeli gramaticale care nu afectează sensul frazelor respective, calculatorul nu iartă nimic. El va semnala imediat orice abatere de la regulile prestabilite pentru a salva interpretări greșite și generarea unor rezultate fără sens.

Gramatica limbajului Logo are un număr redus de reguli și nu are excepții!

10.6 Obiectele limbajului Logo

Datele primite de proceduri la intrare și datele furnizate la ieșire sînt obiecte sub forma de numere sau șiruri de caractere care în Logo se numesc *cuvinte*. O facilitate deosebită a limbajului Logo constă în posibilitatea de grupare și prelucrare a datelor în structuri complexe denumite *liste*.

Numerele, cuvintele și listele sînt obiectele cu care operează procedurile în limbajul Logo.

Numere

Numerele sînt reprezentate în limbajul Logo sub *formă de cuvinte* și pot fi întregi sau fracționare. De exemplu, 7 este un număr întreg iar 1.21 este un număr fracționar. Astfel, instrucțiunile următoare au exact același efect:

PR 12
12

PR "15
15

Operațiile care se pot efectua cu numere și ordinea în care se execută la evaluarea unor expresii complexe sînt:

- împărțirea, / cu prioritatea cea mai mare;
- înmulțirea, × se execută după împărțire;
- scăderea, — înaintea adunării și după înmulțire și împărțire;
- adunarea, + are prioritatea cea mai mică.

Dacă se dorește o altă ordine de efectuare a operațiilor se vor utiliza paranteze. Așa cum s-a arătat la începutul prezentării limbajului, singura excepție privind modul cum arată procedurile predefinite și cele definite de utilizator o constituie posibilitatea de utilizare sub forma infixată a operatorilor aritmetici; astfel următoarele forme sînt echivalente:

EXEMPLU:

Forma infixată

PR 4 + 5
9

Forma standard prefixată

PR SUM 4 5
9

```

PR    7- 3
4
PR   -5 + 3
-2
PR   -8 -2
-10
PR   -8-2
-6
PR    2 + 3 + 5
10
PR    6 * 3
18
PR    6 * 2 * 5
60
PR    2 + 3 * 4
14
PR   (2 + 3) * 4
20
PR   + 7 / 2
3,5
PR   - 7 / 2
-3,5
PR    7 / 0
Can't divide by zero

```

```

PR    SUM    7    -3
4
PR    SUM   -5    +3
-2
PR    SUM   -8    -2
-10
PR    SUM   -8    +2
-6
PR (SUM    2    +3 5)
10
PR  PRODUCT    6    3
18
PR (PRODUCT    6    2 5)
60
PR    SUM    2 PRODUCT 3 4
14
PR  PRODUCT SUM    2    3 4
20
PR    DIV    7    2
3,5
PR    DIV   -7    2
-3,5
PR    DIV    7    0
Can't divide by zero,

```

Trebuie observat că în Logo parantezele rotunde sînt utilizate pentru a reprezenta o grupare a unei operații cu intrările sale. Asupra regulilor sintactice de utilizare a parantezelor se va reveni în paragrafele următoare.

Cuvinte

Șirurile de caractere sînt denumite *cuvinte* în Logo.

Ca și numerele care sînt de fapt tot cuvinte, cuvintele în Logo pot fi transferte ca intrări sau ieșiri în cadrul procedurilor. De asemenea se pot efectua operații de combinare a unor cuvinte în cuvinte mai lungi sau se poate despărți un cuvînt în mai multe cuvinte mai scurte. Un cuvînt se prezintă în Logo prin prefixarea acestuia cu caracterul “.

```

PR "ELEV
ELEV

```

Numerele și variabilele logice TRUE și FALSE sînt cuvinte care se pot scrie fără prefixul “. Astfel următoarele forme sînt echivalente:

```

PR 12          PR "12
12
PR TRUE       PR "TRUE
TRUE
PR FALSE     PR "FALSE
FALSE

```

Cuvîntul care nu are nici un caracter se numește cuvînt *vid*.

```

PR "

```

Un cuvînt începe imediat după caracterul “ sau : și se termină la primul blank, [] () < > + - × / sau la terminarea liniei. Astfel în exemplul de mai jos caracterul “ de la sfîrșit face parte din cuvînt.

```

PR "EXEMPLU"
EXEMPLU"

```

Un cuvint poate conține orice caracter tipăribil în afară de blanc și caracterele delimitatoare arătate mai sus, deși există altă soluție pentru includerea acestor caractere în cuvinte, dacã este strict necesar. Astfel, un caracter din cele de mai sus dacã este precedat de / adică SYS D este luat ca atare și nu este interpretat conform funcției atribuite. Astfel cuvintul 3 [A] B poate fi definit ca " 3 /[A]/ B pentru a evita interpretarea parantezelor drepte ca delimitatori pentru o listă. Primitivele de prelucrare a cuvintelor sînt prezentate în paragraful următor. Acestea oferă posibilitatea prelucrării caracter cu caracter a șirurilor de caractere care formează cuvinte. Pentru prelucrarea șirurilor de caractere cu structuri complexe se utilizează o reprezentare a acestora sub formă de liste. Să scriem de exemplu o procedură recursivă care prelucrează un cuvint la fel cum o procedură de numărare inversă prelucrează numerele.

EXEMPLU: TO TRIUNGHI.CUV :CUVINT

```

IF CUVINT="THEN STOP
PRINT :CUVINT
TRIUNGHI.CUV BUTFIRST :CUVINT
END
TRIUNGHI "ABRACADABRA
ABRACADABRA
BRACADABRA
RACADABRA
ACADABRA
CADABRA
ADABRA
DABRA
ABRA
BRA
RA
A

```

Spre deosebire de procedura de prelucrare numere care scade 1 din număr la fiecare execuție, TRIUNGHI reduce cuvintul eliminând primul caracter pînă se obține cuvintul vid (fără nici un caracter).

Să se scrie o procedură asemănătoare care extrage ultimul caracter la fiecare execuție.

Liste

Listele sînt structuri de obiecte grupate între paranteze drepte [și]. Elementele unei liste sînt cuvinte separate prin blancuri sau alte liste delimitate de paranteze drepte.

EXEMPLE DE LISTE:

```

[BINE AȚI VENIT!] Lista formată din mai multe cuvinte.
[A1 +A2 =40] Lista formată din cuvinte, numere și semne speciale.
[BINE] Lista ce conține un cuvint.
[ ] Lista vidă, nu conține nici un element.
[ACEASTĂ LISTĂ CONȚINE [O SUB LISTĂ] CA ELEMENT]
[ACEASTĂ CONȚINE [O SUB [SUB LISTĂ]]]

```

O listă vidă notată cu [] este asemănătoare cu un cuvint vid notat cu " urmat de blanc dar nu este același lucru.

Acesta este de fapt un caz special al regulii generale că în Logo niciodată o listă nu este același lucru cu un cuvint. De exemplu [CUVINT] și "CUVINT

10. CARACTERISTICI LOGO

sînt structuri de date diferite chiar dacã

PR "CUVÎNT și PR [CUVÎNT]
CUVÎNT CUVÎNT

se executã la fel. Acesta este un alt caz particular al aceleiași legi generale referitoare la Cuvinte și Liste. Listele sînt utilizate pentru a crea structuri de date complexe în general, nu numai pentru a grupa cuvinte în propoziții și fraze.

Procedura TRIUNGHI.CUV poate fi rescrisã foarte ușor pentru a prelucra liste într-o manierã asemãnãtoare.

```
EXEMPLU: TO TRIUNGHI.LIST :LISTA
IF LISTA=[ ] THEN STOP
TRIUNGHI.LST BUTFIRST :LISTA
END
TRIUNGHI.LIST [ACEASTA ESTE O LISTĂ NU UN CUVÎNT]
ACEASTA ESTE O LISTĂ NU UN CUVÎNT
ESTE O LISTĂ NU UN CUVÎNT
O LISTĂ NU UN CUVÎNT
LISTA NU UN CUVÎNT
NU UN CUVÎNT
UN CUVÎNT
CUVÎNT
```

Un lucru foarte important care trebuie reținut este cã atît numerele, cit și cuvintele și listele pot fi transferate ca intrãri și ieșiri ale procedurilor.

Delimitatori

Cel mai utilizat delimitator este spațiul (blancul) care separã la ambele capete un cuvînt de restul liniei. Uneori însã se poate renunța la spații de delimitare dacã cuvintele sînt separate prin operatori sau alte semne speciale. Caracterele care pot fi considerate delimitatori sînt:

[] () = > < + - * /	Astfel:
1 > 2 + (3+4) / 5 - 6	este corect
1 > 2 + (3 + 4) / 5 - 6	este corect și mai clar

Trebuie remarcat modul specific în care Logo utilizeazã parantezele. În primul rînd trebuie bine fãcutã distincția între paranteze drepte [] și paranteze rotunde (). Parantezele drepte delimiteazã o listã.

EXEMPLU: [ROȘU ALB] — o listã cu douã elemente, Roșu și Alb;
[3+4 5] — tot o listã cu douã elemente, 3+4 și 5;
[CULOARE] — o listã cu un element, cuvîntul CULOARE;
[] — lista vidã, fãrã nici un element.

Parantezele rotunde aratã o grupare care nu este întotdeauna asemãnãtoare cu utilizarea în matematicã.

EXEMPLU:

```
PRINT 3+4*5      poate fi scris cu paranteze
23
PRINT (3+4)*5    ca în matematicã
35
```

Dar expresia $x^2 + y^2$ se poate scrie astfel:

```
OUTPUT            SUM            (SQRT :X) (SQRT :Y)
```

Comanda OUTPUT are nevoie de o intrare care este suma celor două intrări ale operației SUM care sînt operațiile SQRT : X și SQRT : Y. De remarcat că (SQRT : X) arată gruparea operației cu intrările sale și nu este același lucru cu SQRT (: X) care de fapt este incorect. Deci în Logo parantezele rotunde arată o grupare și nu sînt semne speciale care delimitează lista intrărilor unei funcții ca în BASIC.

Pentru operațiile aritmetice SUM, PRODUCT este necesară utilizarea parantezelor dacă operația are mai mult de două intrări:

EXEMPLU:

```
PRINT      SUM  3      4      5 nu este corect
(PRINT    SUM  3      4      5) este corect
12
```

Variabile și atribuirea numerelor în Logo

În exemplele precedente se poate constata că procedurile și variabilele de intrare au nume care facilitează referirea la aceste entități. O variabilă poate fi privită ca un container pentru un obiect. Obiectul respectiv este *valoarea* care se atribuie variabilei. Variabila la rîndul ei este referită printr-un nume. O variabilă poate fi creată prin asocierea unor intrări nedefinite numelui unei proceduri sau prin utilizarea primitivei MAKE. În Logo se utilizează comanda MAKE ca o modalitate generală de a atribui lucrurilor un nume.

EXEMPLU:

```
MAKE      "NUMĂR 5      MAKE      "NUMĂR 10
PRINT    :NUMĂR      PRINT    :NUMĂR
5        5            10       10
```

Primul parametru pentru comanda MAKE este numele care va fi atribuit obiectului care este transmis prin al doilea parametru.

De aici înainte referirile la acest obiect se pot face prin numele atribuit. Comanda MAKE poate fi utilizată în diferite moduri exemplificate mai jos.

EXEMPLE:

```
MAKE      "CULOARE      "ALBASTRU
PRINT    "CULOARE
CULOARE

PRINT    :CULOARE
ALBASTRU

MAKE      "FRAZA      [CERUL ESTE SENIN]
PRINT    :FRAZA
CERUL ESTE SENIN

PRINT SENTENCE (BUTLAST :FRAZA) :CULOARE
CERUL ESTE ALBASTRU

MAKE      "FLOARE      "COLORATA
PRINT    :FLOARE
COLORATĂ

MAKE      :FLOARE      [CU ROȘU ȘI CU ALB]
PRINT    :FLOARE
COLORATĂ
```

PRINT :COLORATĂ
CU ROȘU ȘI CU ALB

Se constată din acest exemplu că numele asociat cu [CU ROȘU ȘI CU ALB] nu este literalul "FLOARE ci lucrul asociat acestuia, care este cuvântul CULOARE.

EXEMPLU:

MAKE :FLOARE [CU ROȘU ȘI CU ALB] au același efect ca și
MAKE "COLORATĂ [CU ROȘU ȘI CU ALB].

Pentru a vedea dacă unui obiect i s-a atribuit o valoare, adică dacă unui nume i s-a atribuit un lucru se utilizează comanda NAMEP care reîntoarce valoarea TRUE în caz afirmativ și FALSE în caz negativ.

EXEMPLE:

```

PRINT NAMEP "ANOTIMP
FALSE

MAKE "ANOTIMP "VARA
PRINT :ANOTIMP
VARA

PRINT NAMEP "ANOTIMP
TRUE

```

Mai mult decât atât, funcția THING reîntoarce după apelare lucrul asociat intrării sale, sau altfel spus reîntoarce valoarea asociată unui nume. Astfel THING "X este același lucru cu : X care poate fi considerată o prescurtare.

Dar dacă THING : X este corect din punct de vedere sintactic : : X nu este .

EXEMPLE:

```

MAKE "NUME [MORARU AL. DAN]
PRINT :NUME
MORARU AL. DAN

PRINT THING "NUME
MORARU AL. DAN

PRINT THING (WORD "NU "ME)
MORARU AL. DAN

PRINT THING (FIRST [NUME ADRESA])
MORARU AL, DAN

TO INC :X
MAKE :X 1 + THING :X
END

MAKE "TOTAL 7
PRINT :TOTAL
7

INC "TOTAL
PRINT :TOTAL
8

INC "TOTAL
PRINT :TOTAL
9

```

Variabile locale și globale

Regulile de lucru cu variabile în Logo sint următoarele:

O variabilă declarată ca intrare într-o procedură, deci apare în linia cu titlul definiției, este locală acestei proceduri și subprocedurilor apelate. După terminarea execuției procedurii variabilele locale nu mai au nici o valoare. Astfel, mai multe proceduri pot utiliza variabile locale cu același nume fără să apară vreun conflict. Acest lucru este valabil și dacă în cadrul procedurii se utilizează comanda MAKE pentru a modifica valorile variabilelor de intrare, după cum se arată în exemplele următoare:

EXEMPLE:

```
TO      SCRIE      :X
PRINT  :X
INCR   :X
PRINT  :X
END

TO      INCR      :X
MAKE "X :X+1
PRINT  :X
END

SCRIE  1
1      — tipărit de SCRIE
2      — tipărit de INCR
1      — tipărit de SCRIE
```

Trebuie observat că deși INCR modifică valoarea lui X la 2, a doua afișare a lui X în SCRIE este tot 1, deoarece INCR a modificat valoarea lui X proprie procedurii SCRIE.

O variabilă creată în mod Logo, deci la nivel de comandă cu primitiva MAKE este o variabilă globală.

Astfel de variabilă nu este asociată vreunei proceduri.

Numele variabilelor globale nu interferează în nici un fel cu numele variabilelor locale. În exemplele următoare se ilustrează aceste reguli.

EXEMPLE:

```
MAKE "X 1
INCR :X
2
PRINT :X
1

MAKE "X 3
SCRIE :X
3
4
3
```

O variabilă creată cu comanda MAKE într-o procedură, dar care nu apare ca intrare în linia de definiție este o variabilă globală. Astfel de variabile se numesc în Logo *variabile libere*.

Pentru determinarea valorii asociate unui nume, în prezența variabilelor libere, se face în felul următor:

— Dacă numele apare printre variabilele de intrare ale procedurii, valoarea face parte din setul de valori proprii procedurii;

— Dacă nu, se verifică dacă numele este în setul propriu procedurii care a apelat procedura curentă;

— Dacă nu, se verifică dacă numele este în setul propriu procedurii care a apelat acea procedură, și așa mai departe pînă se ajunge la setul de valori globale definite în mod comandă.

Variabilele libere constituie un mecanism puternic de comunicație între proceduri. Utilizarea fără precauții deosebite poate duce însă la programe greu de înțeles și chiar la erori nedepanabile mai ales dacă se utilizează comanda MAKE pentru schimbarea valorilor variabilelor libere.

EXEMPLU:

```
TO      NIV1
MAKE   "Y 15
NIV2   :Y
END
```

```
TO      NIV2
PRINT  :Y
END
```

```
NIV1
15
```

Variabila Y este creată în procedura NIV1 înainte de utilizarea în procedura NIV2. Deci Y este global. (variabilă liberă).

Cu următoarele definiții se prezintă un alt exemplu:

EXEMPLU:

```
TO      NIV1
NIV1   15
PRINT  :Y
END
```

```
TO      NIV2      :Y
END
```

```
NIV1
Y has no value in NIV 1.
```

Acest mesaj arată că Y este locală subprocedurii NIV2 și nu este accesibilă superprocedurii NIV1.

10.7. Reguli privind lucrul cu proceduri

Facilitățile de definire și utilizare a procedurilor constituie o caracteristică importantă, care conferă putere de expresie deosebită limbajului Logo. Astfel, o problemă complexă poate fi rezolvată cu pași mici, mai ușor de controlat de către programator. În Logo există proceduri predefinite denumite proceduri *primitive* și proceduri definite de utilizator. În capitoul următor sînt prezentate împreună cu exemple de utilizare toate procedurile primitive grupate după

funcțiile lor. În acest paragraf se vor analiza câteva aspecte comune privind definirea și utilizarea procedurilor.

Procedurile definite de programator sînt formate din proceduri primitive.

EXEMPLU:

```
HIDETURTLE
SHOWTURTLE
```

Sînt proceduri primitive care fac ca penelul să dispară de pe ecran, respectiv să apară pe ecran.

EXEMPLU:

```
TO SCRIE
PRINT [BINE AȚI VENIT LA CERCUL DE LOGO]
END
SCRIE defned
```

Aceasta este o procedură definită de programator care scrie mesajul dintre parantezele drepte ori de cîte ori este apelată, utilizînd primitiva PRINT. Definiția unei proceduri începe totdeauna cu linia de definire a titlului care începe cu TO apoi numele procedurii și eventualii parametri de intrare. Ultima linie este totdeauna END. Deoarece o procedură întotdeauna face ceva, s-a ales TO ca șablon de început de definire a unei proceduri. TO SCRIE este ca și cum am zice A SCRIE, TO DRAW este A DESENA, etc.

Apelarea procedurii se face prin introducerea de la tastatură a numelui acesteia.

```
SCRIE și calculatorul afișează pe ecran
BINE AȚI VENIT LA CERCUL DE LOGO.
```

Să definim o procedură care o utilizează pe SCRIE.

EXEMPLU:

```
TO SALUT
SCRIE
PRINT [CRED CA O SĂ VĂ PLACĂ]
PRINT "SUCCES
END

SALUT
BINE AȚI VENIT LA CERCUL DE LOGO
CRED CĂ O SĂ VĂ PLACĂ
SUCCES
```

SCRIE este o subprocedură pentru SALUT iar SALUT este o superprocedură pentru SCRIE. Dacă se apelează o procedură care nu a fost definită și nu este o procedură primitivă se emite un mesaj de eroare.

```
JUMP
```

```
I don 't know how to JUMP.
```

Proceduri cu intrări

Procedurile SCRIE și SALUT afișează întotdeauna același lucru. Ele se execută la fel pentru fiecare apelare. PRINT este apelată de trei ori și de fiecare

dată scrie pe ecran ceea ce primește ca parametru la intrare. Acestea sînt intrări specificate explicit.

În definirea unor proceduri complexe intrările pot fi și implicite, date de ieșirile altor proceduri.

Să definim o procedură care desenează un pătrat pe ecran utilizînd comenzile de deplasare a penelului.

EXEMPLU:

```
TO      PĂTRAT
REPEAT 4 [FORWARD 20 RIGHT 90]
END
```

La fiecare apel se va desena un pătrat cu latura 20 de puncte pe ecran. (Dacă nu pare a fi pătrat aceasta se datorează distanței diferite pe orizontală și pe verticală dintre două puncte de pe ecran).

Pentru a desena un pătrat cu orice latură se definește:

EXEMPLU:

```
TO      PATRATV      :LATURA
REPEAT 4 [FORWARD :LATURA RIGHT 90]
END
```

Acum latura pătratului este specificată la apelare PĂTRATV 20 va avea același efect ca prima definiție PĂTRAT. Trebuie notat că la apelare se scrie PĂTRATV 20, PĂTRATV 100 și nu PĂTRATV : 20 sau PĂTRATV :100. La definire prin :LATURA am înțeles valoarea laturii. La apelare se specifică chiar valoarea, deci nu mai trebuie să apară : ca la definire.

Pentru a desena un dreptunghi oarecare se definește:

EXEMPLU:

```
TO      DREPTUNGHI :LUNGIME :LĂȚIME
FORWARD :LĂȚIME
RIGHT 90
FORWARD :LUNGIME
RIGHT 90
FORWARD :LĂȚIME
RIGHT 90
FORWARD :LUNGIME
RIGHT 90
END
```

Procedura are două intrări și s-a presupus că orientarea penelului este de jos în sus în momentul apelului. După execuția procedurii penelul rămîne cu aceeași orientare.

DREPTUNGHI 50 50 va desena un pătrat. Reamintim faptul că numele care apar ca intrări ale procedurilor (deci apar în linia cu titlul definiției) sînt variabile private și nu interferează cu alte proceduri. Deci mai multe proceduri pot avea același nume ca variabile de intrare fără să apară conflict.

Tipuri de proceduri

Am văzut că după modul în care sînt definite, procedurile sînt de două tipuri: primitive și definite de programator. După modul în care se execută, procedurile sînt iarăși de două tipuri: *comenzi* care nu generează o valoare de

ieşire şi efectul lor se vede imediat şi *operaţii* care generează la ieşire valori care pot fi utilizate mai departe ca intrări pentru alte comenzi sau operaţii. Aceasta înseamnă că pentru a avea sens o procedură de tipul operaţiei întotdeauna trebuie urmată de o altă procedură care face ceva cu ieşirile generate de operaţie. Mai jos se arată câteva exemple:

EXEMPLU:

```
PRINT 2 * 3 + 11
17
MAKE 'A 12
```

Acestea sînt comenzi. Efectul lor este imediat. Afişează pe ecran 17, respectiv modifică valoarea variabilei A.

EXEMPLU:

```
TO INCR :X
: X + 1
END
INCR 5
You don 't say what to do with 6 in INCR
```

Se generează acest mesaj de eroare pentru că valoarea de ieşire nu apare ca intrare într-o procedură.

EXEMPLU:

```
TO INCR :X
OUTPUT : X + 1
END

PRINT INCR 5
6
```

Aceasta este corectă. Valoarea $:X + 1$ este o intrare pentru primitiva OUTPUT. Ieşirea generată de INCR este o intrare pentru comanda PRINT. Deoarece o comandă nu generează nimic la ieşire ea nu poate fi utilizată ca intrare pentru o altă comandă.

EXEMPLU:

```
PRINT FORWARD 50
După ce penelul se deplasează înainte cu 50 paşi se emite mesajul:
FORWARD does not output to PRINT.
```

Procedurile definite de utilizator sînt fie operaţii, fie comenzi, la fel ca şi procedurile primitive.

Introducerea şi editarea procedurilor

Pentru introducerea şi editarea procedurilor trebuie să se ţină seama de modul de lucru în care Logo este la un moment dat.

Mod Logo (nivelul cel mai de sus). Caracterul ? apare la începutul fiecărei linii Logo. Acesta este modul *direct* de lucru în care fiecare instrucţiune introdusă de utilizator este interpretată şi executată imediat, de aceea se mai numeşte şi *mod comandă*.

10. CARACTERISTICI LOGO

Mod TO (de scriere a unei proceduri). În acest mod se intră din mod comandă dacă TO este primul cuvânt dintr-o linie de program Logo.

La începutul fiecărei linii de ecran apare caracterul > pe durata în care Logo este în mod TO.

Modul editare (EDITOR). În acest mod se pot crea proceduri noi sau se pot modifica proceduri definite în prealabil. Intrarea în mod editor se face cu comanda EDIT sau ED urmată de numele procedurii și a variabilelor de intrare. În colțul din stînga sus a ecranului apare și dispăre alternativ caracterul.

În paragraful Logo pe HC 85 se prezintă utilizarea tastaturii în regim de editare.

Ecranul poate fi în unul din următoarele două moduri:

— *Text* în care sînt disponibile 22 linii cu defilare implicită la umplerea ecranului.

— *Grafic* în care primele 22 de linii sînt la dispoziția utilizatorului pentru grafice și texte iar ultimele două linii sînt utilizate pentru scrierea de texte (ca un mic ecran în mod TEXT).

Ieșirea din mod comandă, deci ieșirea din Logo se poate face cu comanda BYE. Revenirea în LOGO se face cu RUN. Zona de lucru a rămas intactă și se poate continua.

10.8. Expresii condiționale

Într-o procedură comenzile și operațiile sînt interpretate și executate de Logo linie cu linie, în ordinea în care sînt scrise de programator. Dacă se întilnește numele unei proceduri se intrerupe execuția curentă pentru a se executa acea subprocedură după care se revine pentru a continua execuția comenzii sau operației următoare apelului subprocedurii. Deși utilizarea procedurilor este principalul instrument de structurare a unui program complex, există și alte modalități de modificare a secvenței în care Logo citește și execută instrucțiunile.

Instrucțiunea IF și predicate

Forma generală a instrucțiunii IF pentru implementarea pe HC 85 a limbajului Logo este:

IF predicat lista instrucțiuni 1 lista instrucțiuni 2

Prima intrare pentru IF este un predicat, o operație logică sau altfel spus condiția pe care o testează IF.

Predicatele sînt operații care generează TRUE sau FALSE după evaluarea expresiei logice de test. Dacă rezultatul este TRUE se execută lista instrucțiuni 1 iar dacă este FALSE se execută lista instrucțiuni 2, dacă este prezentă.

Dacă lista instrucțiuni 2 lipsește, execuția continuă cu instrucțiunea următoare.

EXAMPLE:

```
TO                      SEMN
IF :N < 0 [OUTPUT 'NEGATIV] [OUTPUT 'POZITIV]
END
```

```

PRINT SEMN 17
POZITIV
PRINT SEMN (5—7)
NEGATIV
TO ALEGE
IF 0-RANDOM 3 [OUTPUT "DA]
OUTPUT "NU
END
PRINT ALEGE
DA

```

În aceste exemple IF este o comandă care generează la ieșire POZITIV sau NEGATIV, DA sau NU.

În exemplul următor IF este utilizată ca o operație:

EXEMPLU:

```

TO ALEGEO
OUTPUT IF 0-RANDOM 3 ["DA] ["NU]
END
PRINT ALEGEO
DA

```

Prediccate predefinite

Pentru operațiile cu numere au fost introduse prediccatele < > și =. Există și alte prediccate predefinite ca funcții primitive în Logo. Acestea pot fi ușor reperate deoarece, de obicei, numele acestora se termină cu P. În alte implementări numele prediccatelor predefinite sau definite de programator se termină cu ?.

EXEMPLE DE PREDICATE:

SHOWNP	care generează la ieșire TRUE dacă penelul este în starea activat, afișat pe ecran, altfel generează FALSE.
EMPTYP	obiect care generează TRUE dacă obiectul este fără conținut, altfel FALSE.
EQUALP	obiect 1 obiect 2 forma prefixată a operației =. Generează TRUE dacă obiect 1 și obiect 2 sînt egale, altfel generează FALSE.
LISTP	obiect. Generează la ieșire TRUE dacă obiect este o listă, altfel generează FALSE.
MEMBERP	obiect lista. Generează TRUE dacă obiect aparține listei, altfel FALSE
NUMBERP	obiect. Generează TRUE dacă obiect este număr, altfel FALSE.
WORDP	obiect Generează TRUE dacă obiect este cuvînt, altfel FALSE.
NAMEP	obiect Generează TRUE dacă obiect are o valoare, altfel FALSE.
KEYP	Generează TRUE dacă s-a apăsăat pe o tastă sau pe o combinație de taste recunoscute de Logo, altfel FALSE.
DEFINEDP	cuvînt Generează TRUE dacă cuvînt este numele unei proceduri, altfel FALSE.
PRIMITIVEP	cuvînt Generează TRUE dacă cuvînt este numele unei primitive Logo, altfel FALSE.

Prediccate definite de utilizator

Pe lângă prediccatele predefinite se pot defini noi prediccate deoarece prediccatele nu sînt altceva decît proceduri care generează la ieșire cuvintele TRUE sau FALSE.

EXEMPLE:

```

POZITIVP :X Generează la ieșire TRUE dacă  $x \geq 0$ , altfel FALSE
TO POZITIVP :X

```

10. CARACTERISTICI LOGO

```

IF X < 0      [OUTPUT "FALSE] [OUTPUT "TRUE]
END
VOCALAP      :C generează TRUE dacă cuvântul C începe cu o vocală, astfel FALSE
TO VOCALAP :C
IF (FIRST :C) = "A      [OUTPUT TRUE]
IF (FIRST :C) = "E      [OUTPUT TRUE]
IF (FIRST :C) = "I      [OUTPUT TRUE]
IF (FIRST :C) = "O      [OUTPUT TRUE]
IF (FIRST :C) = "U      [OUTPUT TRUE]
END

```

După ce un predicat a fost definit de către programator, el poate fi utilizat în instrucțiunea IF ca și predicatele predefinite.

Pentru a facilita combinarea predicatelor în Logo există primitivele AND, NOT și OR care efectuează operații logice cu predicate și care generează rezultate tot de tipul TRUE sau FALSE.

Astfel, pentru definirea unui predicat care determină dacă un număr este cuprins între două limite se poate proceda în diferite moduri.

```

EXEMPLE:
TO INTREP :X :MIN :MAX
IF :X < :MIN [OUTPUT FALSE]
IF :X > :MAX [OUTPUT FALSE]
OUTPUT TRUE
END
TO INTREP :X :MIN :MAX
IF OR (:X < :MIN) (:X > :MAX) [OUTPUT FALSE]
OUTPUT TRUE
END
TO INTREP :X :MIN :MAX
IF AND (NOT (:X < :MIN)) (NOT (:X > :MAX)) [OUTPUT TRUE]
OUTPUT FALSE
END
TO INTREP :X :MIN :MAX
OUTPUT AND (NOT (:X < :MIN)) (NOT (:X > :MAX))
END
TO INTREP :X :MIN :MAX
OUTPUT NOT OR (:X < :MIN) (:X > :MAX)
END

```

Deoarece AND și OR sînt la rîndul lor predicate care generează TRUE sau FALSE în ultimele două exemple aceste valori sînt transmise direct procedurii care cheamă pe INTREP.

10.9. Linii complexe și interpretarea lor în Logo

În acest paragraf se vor analiza unele aspecte privind scrierea și interpretarea unor linii complexe în Logo.

O linie de program Logo poate avea pînă la 242 de caractere, deci mult mai lungă decît o linie de pe ecran. O linie se termină efectiv cînd se introduce de la tastatură ENTER. La introducerea unei linii de program complexe, la sfîrșitul liniei de ecran se trece implicit la linia de ecran următoare iar semnul ! care este scris de Logo arată că linia de ecran următoare este continuarea aceleiași linii de program Logo.

La prima vedere notația prefixată pune unele probleme privind „citirea“ unei linii complexe în Logo.

O linie de program este în general interpretată ca o secvență de cuvinte separate prin blankuri sau alți delimitatori.

Să considerăm o linie de program Logo care generează produsul $\sin(x) \cdot \cos(x)$

EXEMPLU:

FD 50 comandă mișcarea penelului cu 50 de pași înainte,
FD50 este o eroare și Logo emite mesajul:
I don't know how to FD50.

exceptând cazul în care a fost definită în prealabil procedura cu numele FD50.

OUTPUT PRODUCT (SIN :X) (COS :X)

Parantezele arată o grupare a unei operații cu intrarea sa.

Această linie se poate scrie și fără paranteze astfel:

OUTPUT PRODUCT SIN :X COS :X și se citește în felul următor:

Primul cuvânt întâlnit este OUTPUT care are nevoie de o intrare. Căutând această intrare continuă explorarea liniei de la stânga la dreapta și următorul cuvânt găsit este PRODUCT. Dar PRODUCT este o operație care necesită două intrări, deci Logo continuă cu căutarea acestora. Următorul cuvânt găsit este SIN care este o funcțiune ce necesită o intrare.

Apoi Logo găsește variabilă :X pe care o atribuie ca intrare pentru SIN și astfel s-a lămurit și prima intrare pentru PRODUCT, explorarea liniei continuă și Logo găsește funcția COS, apoi intrarea :X și cu aceasta s-a lămurit și cea de-a doua intrare pentru PRODUCT. Rezultatul operației PRODUCT constituie intrarea pentru OUTPUT.

Se poate trage concluzia că atâta timp cât se știe câte intrări are fiecare procedură și dacă toate procedurile sînt operatori prefixați pentru intrări nu este nevoie de paranteze, iar explorarea unei linii se face generalizînd exemplul de mai sus. Pe de altă parte, utilizarea parantezelor pentru grupări ușurează atât scrierea cât și interpretarea liniei complexe de program.

Pentru operatori infixati trebuie notat că operatorii aritmetici infixati, *, /, + — au prioritate mai mare decît operatorii prefixați.

EXEMPLU:

WORD 5 + 3 8	dă rezultatul 89 deoarece se execută
5 + 3 = 8	apoi funcția WORD. Aceeași expresie se poate scrie mai explicit:
WORD (5 + 3) 8	Pentru a asigura valoarea —17 variabilei cu numele X se scrie:
MAKE "X (—17)	expresie corectă, sau
MAKE "X —17	semnalată ca eroare deoarece
Logo încearcă să scindă 17 din "X.	

În sfîrșit, operatorii infixati >, <, = au prioritate mai mică decît operatorii prefixați.

Pentru procedurile cu un număr variabil de intrări, trebuie utilizate paranteze pentru grupare dacă numărul intrărilor este mai mare decît cel implicat (vezi SUM, PRODUCT, etc).

Deci regulile pentru interpretarea unei linii Logo sînt:

La întîlnirea numelui unei proceduri trebuie știut dacă:

— este o comandă sau o operație

10. CARACTERISTICI LOGO

— cite intrări trebuie să aibă.

Prima procedură dintr-o linie Logo trebuie să fie întotdeauna o comandă. O operație apare ca intrare pentru o altă procedură.

Trebuie determinată fiecare intrare a unei proceduri.

La completarea intrărilor pentru o comandă, următoarea procedură trebuie să fie tot o comandă.

10.10 Glosar de termeni Logo

- **adresă**: locația unui registru sau a unei celule de memorie prin care se specifică sursa sau destinația datelor.
- **apel**: activarea (lansarea în execuție) a unui program, a unei proceduri sau subproceduri.
- **ASCII**: American Standard Code for Information Interchange; cod standard de reprezentare în calculatoare a informațiilor alfanumerice. Fiecărei cifre sau semn special inclus în standard i se asociază în mod unic un cod format din 7 biți. Ex: A=1000001, B=1000010, 3=0110011 etc.
- **binar**: ceva cu două stări stabile posibile; de ex: sistemul binar de numerație se bazează pe cifrele 0 și 1.
- **bit**: o cifră binară; unitate elementară de reprezentare a informațiilor în calculatoare sub forma unor vectori de biți.
- **boot**: (bootstrapare): procesul de încălcare în memoria calculatorului a unui program de sistem sau aplicativ.
- **buffer** (tampon): o zonă de memorie utilizată ca zonă tampon pentru păstrarea temporară a datelor în cazul transferurilor de la un echipament la altul. O zonă de memorie unde se citesc datele sau de unde se scriu datele în operațiile de intrare/ieșire.
- **bug**: (pană): o eroare într-un program.
- **byte**: vezi octet.
- **call**: vezi apel.
- **caracter**: o literă, o cifră sau alt simbol utilizat pentru organizarea, controlul și reprezentarea datelor.
- **colectare spațiu disponibil** (garbage collection): recuperarea spațiului de memorie care a fost utilizat, dar nu mai este, în vederea reutilizării lui.
- **comandă**: o procedură Logo, fie că este primitivă sau este definită de programator, care face ceva într-un mod particular, fără să genereze ceva la ieșire spre procedura care a chemat-o. Ex. CLEAR SCREEN, FORWARD, PRINT sînt exemple de comenzi.
- **comanda calculatorului**: Dacă în maniera convențională calculatoarele erau folosite în învățămînt pentru a programa elevii — adică pentru a comanda comportarea lor — filozofia încapsulată în Logo se bazează pe ideea că elevii să programeze calculatoarele, să le controleze în mod direct. Se spune adesea că cea mai bună cale de a învăța este predîndu-le altora. Calculatorul este un elev foarte receptiv și devotat profesorului său. Controlul comportării calculatorului poate constitui și o bună experiență de conducere pentru elev.
- **condițional**: o linie Logo care are ca efect executarea unor instrucțiuni diferite, funcție de o condiție testată.
- **coordonată**: un număr care descrie poziția pe orizontală sau verticală a penelului pe ecranul grafic al calculatorului. În poziția inițială penelul are coordonatele [0, 0].
- **cursor**: un marcaj deplasabil utilizat pentru a indica o poziție pe ecran.
- **cuvînt vid**: un cuvînt care nu are nici un caracter. Un cuvînt vid se scrie astfel
- **defilare**: deplasarea imaginii pe ecran, sau a unei părți a acesteia, pe verticală sau orizontală pentru a face loc unor date să fie afișate în porțiunea utilă a ecranului iar datele de pe marginea cealaltă a ecranului dispar din câmpul vizual.
- **depanare**: găsirea și eliminarea greșelilor dintr-un program, îmbunătățind astfel comportarea unui program care nu face ceea ce se dorește de la el. În Logo se consideră că programele care nu se comportă exact cum s-a dorit sînt programe neterminate, nu terminate prost. În general în școală compozițiile scrise ale elevilor sînt notate așa cum sînt, nu după ce au fost depanate. Aceasta din motive obiective, inclusiv efortul prohibitiv care ar fi necesar dacă elevii ar fi încurajați să-și îmbunătă-

- tească lucrările scrise. În Logo elevii sînt încurajați să studieze cu atenție pe-
nele din programele lor. Există mari
șanse ca o comportare nedorită a unui
program, dacă este bine studiată să poa-
tă duce la idei noi chiar mai interesante
decît ideea inițială. Dintr-o pană, se
poate învăța întotdeauna ceva.
- **dispozitiv:** un echipament atașat cal-
culatorului, de exemplu o imprimantă,
un terminal grafic.
 - **editare:** introducerea (de la tastatură),
modificarea sau ștergerea unui text re-
prezentînd proceduri, date sau orice
șiruri de caractere.
 - **editor:** acea porțiune din Logo care asis-
tă programatorul și-l ajută să editeze
un text alfanumeric într-o zonă de me-
morie numită zonă de editare.
 - **element:** un membru dintr-o mulțime
sau un termen dintr-o serie.
 - **execută:** comandă Logo să efectueze
o instrucțiune sau o procedură. Aceasta
se poate face furnizînd numele procedu-
rii direct de la tastatură sau indi-
rect, ca instrucțiune într-o subprocedură.
 - **exersare penel:** programatorul se substi-
tue penelului și parcurge o procedură
grafică așa cum este ea văzută de penel.
Aceasta poate duce la o bună înțelegere
a unor construcții geometrice complexe,
fără o pregătire corespunzătoare în ge-
metrie. Pe un plan mai larg, acest proces
trebuie privit mai mult ca o parcurgere
mintală a unei proceduri înainte de
programarea ei, decît ca o parcurgere
a unei proceduri existente. De exemplu,
desenarea unui cerc cu penelul, știînd
că singurele comenzi sînt deplasarea
înainte și rotirea, poate fi ușor înțeleasă
prin exersarea penelului.
 - **fenomenul QWERTY:** Papert utilizează
acest fenomen pentru a se referi la si-
tuații păstrate prin tradiție care deși au
avut inițial o motivație corectă, au de-
venit învechite, demodate. Numele vine
de la cele 6 litere din partea stîngă sus
a unei tastaturi de calculator. Aranjarea
tastelor s-a făcut inițial pentru a evita
suprapunerea mai multor taste prin
„încetinirea“ vitezei de lucru a opera-
torului (taste care sînt frecvent alătura-
te în text au fost aranjate la distanță
pe tastatură). Mai multe încercări de a
optimiza aranjarea tastelor au eșuat
datorită obișnuinței de a lucra cu vechea
aranjare. Astfel multe din programele
școlare supraviețuiesc datorită obiș-
nuinței, chiar dacă nu mai sînt adec-
vate.
- **figură deschisă:** o figură, o formă formată
dintr-o linie care nu se mai întoarce în
punctul de start.
 - **figură închisă:** o figură, o formă for-
mată dintr-o linie continuă care se rein-
toarce în punctul inițial.
 - **fișier:** o colecție de informații organi-
zată și memorată pe un suport în ve-
derea unor prelucrări ulterioare.
 - **fișier AȘCII:** un fișier de tip text for-
mat din șiruri de caractere reprezentate
în cod ASCII.
 - **format:** o aranjare a datelor, conform cu
anumite șabloane, pe ecran, imprimată,
bandă sau disc magnetic.
 - **global:** care se aplică la întreg domeniul
considerat. Variabilele globale sînt acce-
sibile și pot fi modificate din orice parte
a programului. Utilizarea variabilelor
locale — a căror domeniu de valabili-
tate este o procedură — este de dorit
deoarece conduce la programe mai ordo-
nate, mai clare și mai ușor de depanat.
Sistemul de coordonate cartezian per-
mite specificarea globală a punctelor
unei figuri, față de origine. Aceasta
crează dificultăți importante în re-
prezentările grafice, fapt pentru care
Logo este orientat în general spre repre-
zentări locale, relative.
 - **grafica cu penelul:** efectuarea unor
exerciții de grafică cu penelul are un
mare succes atît pentru introducerea în
programare pentru orice vîrstă, pentru
însușirea unor concepte fundamentale
din programare cît și ca bază pentru o
programă de matematică asistată de cal-
culator.
 - **heading:** vezi orientare.
 - **imagini antropomorfe:** metafore prin
care calculatoare, proceduri sau obiecte
controlate de calculatoare sînt privite
ea și cum ar fi persoane. Imaginile antro-
pomorfe facilitează transferul de cu-
noștințe dintr-un mediu familiar spre
un nou context. Gîndindu-ne la mașini
ca la niște oameni (limitați) sau chiar
modelînd comportarea oamenilor prin
analogii cu procese ce se desfășoară în
diferite mașini nu implică tratarea oame-
nilor ca pe mașini. Prin învățarea penel-
ului să se miște după legile lui Newton
se poate spune că penelul a descoperit o
nouă lume. Aceasta nu este decît o me-
taforă pentru activitatea de programarea
calculatorului.
 - **implileit:** o valoare sau o opțiune furni-
zată de program dacă utilizatorul nu
specifică nimic atunci cînd el poate
sau trebuie să specifice aceste valori
sau opțiuni.

- **Instruire asistată de calculator (CAI):** în sens larg, orice activitate educațională care utilizează calculatoare. Logo presupune utilizarea calculatorului nu numai pentru furnizarea, exersarea și testarea cunoștințelor ci mai degrabă ca un context de utilizare și dezvoltare a gândirii în scopul rezolvării problemelor de interes specific. Instruirea programată modelează omul ca o mașină în sensul că elevul este programat de calculator. Utilizarea limbajului Logo consideră elevul ca un factor constructiv care construiește cunoștințe în mod activ. Elevul ia rolul profesorului învățând penelul o nouă lume.
- **Inteligența artificială:** o ramură a științei calculatoarelor din care a izvorât și Logo. Numele acestei ramuri a științei derivă din încercările de dezvoltare a unor metode și tehnici de utilizare a mașinilor pentru a simula comportamentul considerat inteligent al oamenilor și animalelor. Această privire îngustă nu mai este valabilă. Psihologia cunoașterii și inteligența artificială sînt deseori referite împreună ca știința cunoașterii. Printre domeniile de activitate ale IA sînt: înțelegerea limbajului natural, percepția vizuală, acumularea și reprezentarea cunoștințelor.
- **Interactiv:** un mod de lucru pe bază de dialog între calculator și utilizator.
- **Intrare/ieșire:** intrarea unei proceduri reprezintă un mesaj necesar procedurii pentru a-și executa funcția. Unele proceduri au nevoie de mai multe mesaje de intrare pentru a ști ce trebuie să facă. De exemplu o procedură de desenare a unui poligon oarecare are nevoie de două intrări, numărul de pași pe care-i face înainte (lungimea laturii) și unghiul de întoarcere la fiecare colț. Ieșirea unei proceduri reprezintă mesajul transmis de o procedură spre procedura care a apelat-o, procedură care produce o ieșire (o operație) trimite mesajul de ieșire spre o procedură care-l așteaptă ca intrare. De exemplu mesajul transmis de procedura SUM este așteptat de PRINT pentru a-l tipări. Intrările și ieșirile sînt strict corelate în Logo.
- **Iterație:** execuție repetitivă, de către calculator, a unei secvențe de instrucțiuni. În Logo se preferă recursivitatea pentru implementarea structurilor de control repetitive.
- **Intreg:** un număr pozitiv sau negativ care nu conține o parte fracționară.
- **linie de continuare:** o linie de program Logo care reprezintă extensia unei linii de ecran precedente. Punctul de întrerupere a liniei este marcat cu semnul exclamărilor.
- **LISP:** prescurtarea pentru List Processing. Un limbaj de programare foarte utilizat în cercetările din domeniul inteligenței artificiale, sursa multor idei din Logo.
- **listă:** structura de date de bază în Logo. Lista este o secvență ordonată de obiecte arbitrare. Deoarece un obiect poate fi un cuvînt, un număr sau altă listă (care la rîndul ei poate conține alte liste, și asta pe oricîte nivele), listele pot fi utilizate pentru a crea structuri de date foarte complexe.
- **lista vidă:** o listă care nu are nici un element. O listă vidă se scrie astfel [].
- **local:** ceva care are sens (se aplică) într-un context (domeniu) specific în numai în cadrul procedurii în care a fost definit. Într-o altă procedură poate fi definită o altă variabilă locală cu același nume fără ca ele să interfere. Modul de deplasare a penelului este de asemenea local, fără să fie nevoie să se țină seama de poziția absolută în raport cu o origine fixă. Pentru a desena un cerc, penelul ia în considerare doar porțiunea din plan din jurul poziției curente. Spre deosebire de aceasta, generarea cercului după execuția $x^2 + y^2 = r^2$ are un caracter global, într-un sistem de coordonate global.
- **Logo:** este o prescurtare ce derivă din cuvîntul grecesc pentru „cuvînt” sau „gînd”.
- **notație infixată:** un mod de a scrie expresiile aritmetice astfel că operatorul este plasat între operanzi.
- **notație prefixată:** un mod de scriere a expresiilor aritmetice astfel că operatorul este plasat înaintea operanzilor.
- **număr real:** un număr fracționar pozitiv sau negativ.
- **obiect:** entitate specifică de reprezentare a datelor în Logo. Logo operează cu obiecte reprezentînd cuvînte și liste. Deși numerele sînt tot cuvînte, uneori sînt privite ca obiecte separate. Unele implementări includ obiecte de tip tablou. Obiectele sînt prelucrate, prin intermediul lor se transmit mesaje de intrare și ieșire în cadrul procedurilor. Deși procedurile sînt privite ca elemente active, ele pot fi manipulate (create, distruse, modificate, transmise înainte sau înapoi)? de către alte proceduri. Deci și procedurile pot fi uneori privite ca obiecte.

- **octet**: un vector binar format din 8 biți. Lungimea cuvintelor cu care operează majoritatea calculatoarelor este de 1 sau mai mulți octeți.
- **operație**: o procedură în Logo care calculează o valoare și o transmite procedurii chemătoare. Valoarea respectivă poate fi un număr, un cuvânt sau o listă.
- **operație logică**: un predicat care are ca intrări valorile TRUE sau FALSE.
- **orientare**: direcția, în grade, spre care este orientat penelul cu vârful colorat al triunghiului. Este un număr cu valoarea între 0 și 359 și este furnizat la ieșirea operației **HEADING**; 0 corespunde direcției Nord, 90 — Est, 180 — Sud și 270 — Vest.
- **pană**: vezi bug.
- **penel**: peniță electronică ale cărei mișcări pe ecranul de afișare sînt controlate de către calculator prin comenzi de mișcare (**FORWARD**, **BACK**) sau comenzi de rotație (**LEFT**, **RIGHT**). Pe ecran penelul apare ca un triunghi mic cu un vîrf colorat care denotă și orientarea, direcția pe care se va deplasa la o comandă. Prin deplasare, penelul lasă urma deplasării sale pe ecran și poate fi folosit în acest mod pentru a desena pe ecran. În limba engleză se utilizează termenul de „turtle”.
- **peneldin**: penel dinamic. Așa cum penelul (static) are poziția spațială și orientarea fixate prin comenzi Logo, penelul dinamic poate avea viteza sau accelerația fixate. O comandă de schimbare a stării modifică coordonatele spațiale sau orientarea penelului static sau modifică viteza penelului dinamic, avînd un efect analog aplicării unei forțe. Peneldin se comportă ca o rachetă în spațiul extraterestru. Pentru a deplasa nava este necesară aplicarea unui impuls prin aprinderea unei rachete. Aceasta se mișcă apoi în aceeași direcție pînă primește un nou impuls. Noua mișcare este o combinație între mișcarea veche și mișcarea cauzată de impulsul aplicat. Penelul dinamic este un instrument deosebit în studierea unor probleme de fizică și matematică.
- **Piaget**: Jean Piaget, considerat unul dintre marii gînditori în domeniul educației. Observînd comportamentul copiilor în fiecare, atunci cînd învață, are un rol activ în propria-i dezvoltare. A arătat valoarea examinării și înțelegerii unui răspuns greșit. Un răspuns greșit nu trebuie privit ca o mișcare aleatoare în lipsa unui răspuns corect, ci ca o pană într-un program care în cea mai mare parte oferă răspunsul corect. Trebuie examinată „logica gîndirii” care a dus la acel răspuns sau explicație. Aceasta este modul în care în Logo sînt privite penele dintr-un program și depanarea acestora.
- **pixel**: un punct grafic pe ecran.
- **predicat**: o procedură care generează la ieșire TRUE sau FALSE.
- **primitive**: procedurile intrinseci ale limbajului Logo. Acestea pot fi utilizate de către programatori pentru construirea propriilor proceduri.
- **procedură**: element conceptual de bază în construirea programelor în Logo. Programatorul are inițial un vocabular format din proceduri primitive pe care le poate utiliza pentru definirea de noi proceduri care sînt utilizate exact în aceeași manieră ca și cele primitive.
- **prompt**: un semn afișat de către calculator pe ecran pentru a atenționa utilizatorul să furnizeze informații.
- **program**: datorită utilizării extensive a procedurilor, putem spune că procedura de pe cel mai înalt nivel — o superprocedura execută un program de obicei foarte simplu sau care constă numai din apelul mai multor subproceduri.
- **propoziție**: o listă de cuvinte. În Logo sînt furnizate unele puternice de prelucrare a cuvintelor unei propoziții.
- **recursivitatea**: o procedură sau o funcție recursivă este definită în termenii ei însăși. Prin recursivitate o problemă foarte complicată poate fi exprimată în termenii unor versiuni mai simple ale aceleiași probleme. În general o definiție recursivă a unei proceduri este mai eficientă decît o definiție repetitivă, iterativă. De multe ori recursivitatea este singura cale de rezolvare a unei probleme.
- **rezolvare de probleme**: deprindere care constituie punctul central al oricărei activități în Logo. Cea mai mare parte a instruirii tradiționale în matematică (și alte discipline) se referă la probleme rezolvate, la însușirea tehnicilor altora de rezolvare a problemelor. Logo încearcă implementarea ideii de rezolvare a unor probleme noi. Ca tehnică se utilizează divizarea unei probleme complexe în părți mai mici prin utilizarea extensivă a procedurilor.
- **scroll**: vezi defilare.
- **sintaxă**: regulile care stau la baza structurii unui limbaj.

- **spațiu de lucru**: o parte a spațiului de memorie a calculatorului utilizată pentru a memora variabile, proceduri și proprietăți într-o sesiune de lucru în Logo.
- **stare**: o parte din istoria evoluției unui lucru care permite ca pe baza unor acțiuni asupra lui să putem determina evoluția ulterioară. Prin stare se exprimă proprietățile relevante la un moment dat. De exemplu, starea penelului la un moment dat este data de poziția și direcția în care este orientat. Cunoșcând starea curentă, aceasta este suficient de relevantă deoarece prin aplicarea unor comenzi prin program se poate determina comportarea penelului. Posibilitatea de izolare, de decupare a aspectelor importante (relevante) ale unei situații este un instrument de depanare foarte eficient.
- **stivă**: o structură de date astfel că ultimul element memorat în stivă va fi primul element extras în vederea prelucrării.
- **subprocedură**: o procedură care este apelată de o altă procedură. O procedură poate fi subprocedura pentru ea însăși, vezi recursivitate.
- **șir**: o secvență de caractere.
- **știința de carte în calculatoare**: în general prin știința de carte se înțelege posibilitatea de a scrie și a citi. În mod similar știința de carte în calculatoare a fost văzută ca o experiență generală și adeseori superficială în calculatoare. Cunoașterea principiilor de funcționare și de organizare a unui calculator și posibilitatea de exprimare fluentă într-un limbaj de programare constituie sensul dat în această carte termenului respectiv. Oferind elevilor posibilitatea de a fi programatori sînt ajutați în exprimarea fluentă a ideilor matematice și logice. Aceasta poate apărea mai degrabă ca învățarea unei limbi străine — o opțiune foarte bună pentru cei ce s-au hotărît să o facă, dar nu neapărat necesară pentru a trăi și a munci. Răspîndirea foarte largă a calculatoarelor în toate domeniile de activitate oferă un statut nou, obiectiv, științei de carte în calculatoare.
- **tampon**: vezi *buffer*.
- **valoare**: conținutul unei variabile. Valoarea este asociată unei variabile cu un anumit nume.
- **variabilă**: un container (locație de memorie) care are un nume și care conține o valoare.
- **workspace**: vezi spațiu de lucru.

11.1. Caracteristici generale ale primitivelor LOGO

LOGO este un limbaj flexibil, de tip procedural. Limbajul dispune de proceduri implicite denumite *primitive*. Aceste primitive stau la baza definirii procedurilor utilizator, cu ajutorul cărora se descriu algoritmi care implementează diverse programe. Din punctul de vedere al programatorului nu există deosebire între primitivele limbajului și procedurile construite de el în ceea ce privește folosirea acestora.

În acest capitol se vor prezenta primitivele limbajului LOGO cu ajutorul cărora utilizatorii își vor descrie procedurile și programele lor.

Principalele grupuri de primitive sînt:

- primitive pentru controlul penelului și ecranului în mod grafic și alfanumeric;
- primitive pentru specificarea operațiilor matematice și logice;
- primitive pentru manipularea obiectelor LOGO (variabile, numere, cuvinte, liste);
- primitive pentru comunicația cu echipamentele de intrare/ieșire;
- primitive care asigură ramificațiile în program;
- primitive pentru lucrul cu fișiere și gestiunea spațiului de lucru;
- primitive pentru editarea procedurilor;
- primitive pentru diferite funcții.

În limbajul LOGO există două tipuri de primitive:

- comenzi și
- operații.

Comenzile sînt acele primitive care au o acțiune directă în momentul apelării, de exemplu: diferite operații asupra penelului, încărcarea/salvarea procedurilor, listarea/ștergerea procedurilor, etc. fără a produce o valoare care să fie utilizată de o altă primitivă sau procedură.

Operațiile sînt acele primitive care generează o valoare care va fi utilizată de o altă primitivă sau procedură.

În continuare se vor descrie diversele tipuri de primitive ale limbajului LOGO sub forma:

- denumirea primitivei;
- mnemonica primitivei și intrările asociate acesteia;

- mnemonica prescurtată și intrările asociate;
- descrierea funcției efectuată de primitivă;
- exemplu de utilizare a primitivei.

11.2. Primitive LOGO pentru controlul penelului și al ecranului în regim grafic

În limbajul LOGO ecranul calculatorului poate fi utilizat în două regimuri de lucru:

- a. mod grafic = ecranul este considerat o suprafață cu 256 puncte pe orizontal și 175 puncte pe vertical;
- b. mod alfanumeric = ecranul este considerat o suprafață cu 22 de linii a câte 32 caractere fiecare.

În regim grafic există un cursor grafic numit **penel** (pensula cu care se pictează sau prescurtare de la **penița** electronică), ce poate fi controlat de utilizator prin intermediul primitivelor LOGO. În limba engleză, pentru acest cursor grafic, se folosește noțiunea „turtle“ (trădus: „broșcuța“).

În limbajul LOGO există o serie de primitive care oferă utilizatorului posibilitatea de a controla ceea ce se vede pe ecran în regim grafic și de a afla (specifica) starea ecranului și a penelului.

Primitivele vor fi grupate în patru categorii:

- primitive pentru schimbarea stării penelului;
- primitive care specifică starea penelului;
- primitive pentru utilizarea penelului și a ecranului;
- primitive care specifică starea penelului și a ecranului.

Pentru mișcarea penelului pe ecran se pot folosi comenzi de:

- mișcare absolută (față de originea axelor ecranului);
- mișcare relativă (față de ultima poziție a penelului pe ecran)— prin specificarea distanței de deplasare și a unghiului de rotire a direcției.

Având în vedere că mișcarea penelului se face prin specificarea distanței de deplasare și a direcției este necesar să precizăm axele ecranului și sensul direcției.

În limbajul LOGO ecranul în regim grafic, arată ca în figura 11.1

La inițierea unei sesiuni de lucru în limbajul LOGO, penelul este considerat în originea axelor, punctul de coordonate 0, 0 și orientat spre direcția Nord.

Orice schimbare a direcției penelului în sensul mișcării acelor de ceas se consideră spre dreapta, iar în sensul opus spre stînga.

Primitivele din cadrul fiecărei categorii se vor descrie, în general, în ordine alfabetică pentru a ușura accesul utilizatorului.

11.2.1. Primitive pentru schimbarea stării penelului

În acest paragraf se vor descrie o serie de comenzi care vor controla mișcarea penelului, pe ecranul aflat în regim grafic.



Fig. 11.1. Orientarea și limitele poziționale ale ecranului în regim grafic

Comenzile care vor fi tratate în acest paragraf sînt:

BACK	RIGHT
CLEARSCREEN	SETHEADING
FORWARD	SETPOS
HIDETURTLE	SETX
HOME	SETY
LEFT	SHOWTURTLE

Comanda BACK

BACK d

BK d

Comanda BACK, mută penelul înapoi (în sensul invers virfului penelului) cu distanța „d” pași (puncte), specificată în cadrul comenzii. Direcția penelului nu se schimbă. De notat faptul că dacă distanța „d” este negativă, mișcarea se face înainte:

(BK -d ≡ FD d)

Dacă penița este jos (penelul este activ) atunci se trasează o linie în timpul mișcării penelului. Mișcarea este relativă față de poziția curentă a penelului.

EXEMPLU: (Fig. 11.2).
BK 50



Fig. 11.2. Exemplu de utilizare a comenzii BACK

Comanda CLEARSCREEN

CLEARSCREEN
CS

Comanda CLEARSCREEN, șterge ecranul considerat în regim grafic și, mută penelul în centrul acestuia, coordonate 0, 0, orientat spre Nord.

EXEMPLU: (Fig. 11.3).
CS



Fig. 11.3. Exemplu de utilizare a comenzii CLEAR SCREEN

Comanda FORWARD

FORWARD d
FD d

Comanda FORWARD, mută penelul înainte (în sensul indicat de vârful penelului) pe distanța „d” pași. Direcția penelului nu se schimbă. De notat faptul că dacă distanța „d” este negativă mișcarea se face înapoi. (FD -d ≡ BK d). Dacă penița este jos (penelul este activ), atunci se trasează o linie în timpul mișcării penelului. Mișcarea este relativă față de poziția curentă a penelului.

EXEMPLU: (Fig. 11.4).
FD 50



Fig. 11.4. Exemplu de utilizare a comenzii FORWARD

Comanda HIDE/TURTLE

HIDE/TURTLE

HT

Comanda HIDE/TURTLE, trece penelul într-o stare în care nu se mai vede pe ecran (este invizibil). Operațiile asupra sa au același efect ca pînă acum numai că el nu se vede. Deoarece penelul nu se mai desenează pe ecran, orice comandă de mișcare a penelului (cu trasare sau nu) se execută mai repede. Efectuarea unui desen pe ecran cu penelul invizibil se realizează mai repede decît cu penelul vizibil. Se recomandă însă, ca în faza de punere la punct a unui desen, să se lase penelul în stare vizibilă și numai cînd acesta este finalizat să se comande trecerea penelului în starea invizibilă.

EXEMPLU: (Fig. 11.5)

HT



HT

Fig. 11.5. Exemplu de utilizare a comenzii HIDE/TURTLE

Comanda HOME

HOME

Comanda HOME, mută penelul în poziția din centrul ecranului, coordonate 0, 0, cu fața spre Nord (0°). Dacă penița este jos (penelul este activ) se trasează o dreaptă de la vechea poziție a penelului pînă la punctul de coordonate 0,0.

EXEMPLU: (Fig. 11.6)

HOME



HOME

Fig. 11.6. Exemplu de utilizare a comenzii HOME

Comanda LEFT

LEFT gr

LT gr

Comanda LEFT rotește direcția penelului cu „gr” grade spre stînga (invers mișcării acelor de ceas) fără a schimba poziția x, y, a acestuia. Dacă parametru „gr” are valoare negativă direcția penelului se rotește spre dreapta ((LT -gr ≡ RT gr). Rotirea este relativă față de direcția curentă a penelului.

11. PRIMITIVE LOGO

EXEMPLU: (Fig. 11.7).

LT 45



Fig. 11.7. Exemplu de utilizare a comenzii LEFT

LT 45

Comanda **RIGHT**

RIGHT gr
RT gr

Comanda **RIGHT** rotește direcția penelului cu „gr“ grade spre dreapta (în sensul mișcării acelor de ceas), fără a schimba poziția x, y, a acestuia. Dacă parametrul „gr“ are valoarea negativă direcția penelului se rotește spre stînga (**RT -gr** \equiv **LT gr**). Rotirea este relativă față de direcția curentă a penelului.

EXEMPLU: (Fig. 11.8).

RT 90



Fig. 11.8. Exemplu de utilizare a comenzii RIGHT

RT 90

Comanda **SETHEADING**

SETHEADING gr
SETH gr

Comanda **SETHEADING**, stabilește direcția penelului în funcție de valoarea parametrului „gr“, exprimat în grade, față de Nord (0°), orientarea inițială a ecranului. Parametrul „gr“ specifică direcția absolută a penelului.

Direcția penelului se stabilește considerînd valoarea parametrului modulo 360. Dacă „gr“ are valoarea negativă atunci direcția se stabilește complementar față de 360° .

EXEMPLU: (Fig. 11.9).

SETH 45 \equiv SETH -315 \equiv SETH 765



Fig. 11.9. Exemplu de utilizare a comenzii SETHEADING

SETH 45

SETH -315

SETH 765

Comanda SETPOS

SETPOS [x y]

Comanda SETPOS mută penelul în poziția determinată de coordonatele absolute x y, specificate sub formă de listă [x y]. Dacă penelul este activ se trasează o dreaptă între poziția curentă a cursorului și poziția specificată de coordonatele x y. Direcția penelului nu se schimbă. Coordonatele x y pot depăși valorile maxime admise pentru ecran, dacă acesta este configurat cu comanda WINDOW, dar în acest caz el nu se va vedea pe ecran. Dacă coordonatele depășesc limitele prezentate în figura 11.1, se indică eroare în cazul în care ecranul a fost configurat cu comanda FENCE. În cazul în care ecranul a fost configurat cu comanda WRAP, valorile se iau modulo limite.

Trebuie menționat faptul că dacă dorim să poziționăm penelul într-un punct de coordonate :X :Y — reprezentate de valorile unor variabile, va trebui să creăm o listă cu aceste variabile, utilizând operația SENTENCE:

SETPOS SENTENCE :X :Y

— NU este permisă construcția SETPOS [:X :Y]

EXEMPLU: (Fig. 11.10)
SETPOS [20 30]



SETPOS [20 30] Fig. 11.10. Exemplu de utilizare a comenzii STPOS

Comanda SETX

SETX x

Comanda SETX, muta penelul orizontal în punctul de abscisă x, ordonata y rămânând nemodificată. Dacă penelul este activ (penița jos), se va trasa o linie orizontală între poziția curentă și poziția specificată.

EXEMPLU: (Fig. 11.11)
SETX 70



SETX 70

Fig. 11.11. Exemplu de utilizare a comenzii SETX

Comanda SETY

SETY y

Comanda SETY, mută penelul în punctul de ordonată y, abscisa x rămânând nemodificată. Dacă penelul este activ (penița jos), se va trasa o linie ver-

ticală între poziția curentă și poziția specificată.

EXEMPLU: (Fig. 11.12).

SETY 75



SETY 75

Fig. 11.12. Exemplu de utilizare a comenzii SETY

Comanda **SHOWTURTLE**

SHOWTURTLE

ST

Comanda **SHOWTURTLE**, trece penelul în starea în care se vede pe ecran (penelul vizibil). Are un efect invers față de comanda **HIDETURTLE**.

EXEMPLU: (Fig. 11.13)

ST



ST

Fig. 11.13. Exemplu de utilizare a comenzii SHOWTURTLE

11.2.2. Primitive care specifică starea penelului

În acest paragraf se descriu operațiile prin care utilizatorul poate afla informațiile privind starea penelului.

Primitivele ce vor fi tratate în acest paragraf sint:

HEADING	TOWARDS
POSITION	XCOR
SHOWNP	YCOR

Operația **HEADING**

HEADING

Operația **HEADING**, generează un număr natural între 0 și 359, care specifică direcția absolută a penelului. Numărul generat de operația **HEADING** poate fi utilizat ca parametru pentru diverse comenzi **LOGO**.

La începutul unei sesiuni de lucru **LOGO** direcția penelului este spre Nord (0°).

EXEMPLU:

CS
LT 90
PRINT HEADING
270

Operația POSITION

POSITION
POS

Operația POSITION, generează coordonatele absolute ale poziției curente a penelului, sub forma unei liste [x y]. La începutul unei sesiuni de lucru apelul operației POSITION va genera lista [0 0].

EXEMPLU:

FD 30
RT 90
FD 70
PRINT POSITION
70 30

Operația SHOWNP

SHOWNP

Operația SHOWNP generează valoarea logică TRUE, dacă penelul este în starea vizibilă și valoarea logică FALSE, în caz contrar.

EXEMPLU:

ST
PRINT SHOWNP
TRUE
HT
PRINT SWOWNP
FALSE

Operația TOWARDS

TOWARDS [x y]

Operația TOWARDS, generează un număr natural între 0 și 359, care specifică ce direcție trebuie să aibă penelul pentru a fi îndreptat spre punctul de coordonate x, y. Coordonatele x, y, se specifică sub forma unei liste [x y].

Trebuie menționat faptul că dacă dorim să îndreptăm penelul spre un punct ale cărui coordonate rezultă din valorile unor variabile :X :Y, va trebui să creăm o listă cu aceste variabile, utilizând operația SENTENCE, astfel:

TOWARDS SENTENCE :X :Y

EXEMPLU: (Fig. 11.14)

SETHEADING TOWARDS [20 50]

**SETHEADING TOWARDS [20 50] Fig. 11.14. Exemplu de utilizare a operației
FD 30 TOWARDS**

Operația **XCOR**

XCOR

Operația **XCOR** generează un număr care specifică valoarea abscisei x corespunzătoare poziției curente a penelului.

EXEMPLU:

```
CS
RT 90
FD 40
PRINT XCOR
40
```

Operația **YCOR**

YCOR

Operația **YCOR** generează un număr care specifică valoarea ordonatei y corespunzătoare poziției curente a penelului.

EXEMPLU:

```
CS
FD 20
PRINT YCOR
20
```

11.2.3. Primitive pentru utilizarea peniței electronice și a ecranului

În acest paragraf se descriu comenzile care permit utilizatorului să controleze direct penelul și ecranul.

Primitivele care vor fi tratate în acest paragraf sînt:

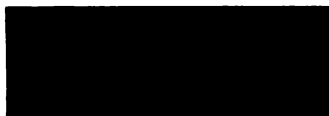
```
CLEAN          SETBG
DOT            SETBORDER
FENCE         SETPC
PENDOWN       SETSCRUNCH
PENERASE      WINDOW
PENREVERSE    WRAP
PENUP
```

Comanda **CLEAN**

CLEAN

Comanda **CLEAN** șterge ecranul, în regim grafic, fără să afecteze poziția și direcția penelului.

EXEMPLU: (Fig. 11.15)
CLEAN



CLEAN

Fig. 11.15. Exemplu de utilizare a comenzii CLEAN

Comanda DOT

DOT [x y]

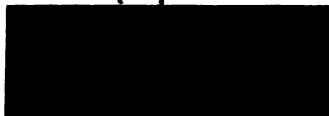
Comanda DOT, desenează un punct în poziția determinată de coordonatele x y, coordonate specificate sub forma de listă [x y], fără ca penelul să-și schimbe poziția și direcția.

Trebuie menționat faptul că dacă dorim să desenăm un punct într-o poziție determinată de valorile unor variabile :X :Y, trebuie să creăm o listă cu aceste variabile utilizând funcția SENTENCE:

DOT SENTENCE :X :Y

Dacă coordonatele depășesc limitele ecranului atunci va apare un mesaj corespunzător.

EXEMPLU: (Fig. 11.16)
DOT [0 0]



DOT [0 0]

Fig. 11.16. Exemplu de utilizare a comenzii DOT

Comanda FENCE

FENCE

Comanda FENCE limitează mișcarea penelului în limitele ecranului (Fig. 11.1). După executarea acestei comenzi, orice încercare de mutare a penelului în afara limitelor ecranului produce un mesaj de eroare și penelul nu își schimbă poziția. Altfel spus, comanda FENCE nu permite trecerea penelului din zona vizibilă a ecranului în zona invizibilă (Fig. 11.21)

Dacă penelul se află deja în afara limitelor vizibile ale ecranului, comanda FENCE nu se validează.

EXEMPLU:
FENCE
CS
FD 200

— se va genera mesajul de eroare:

Comanda PENDOWN

PENDOWN

PD

Comanda PENDOWN activează penelul, în sensul că pune jos penița. Orice mișcare a penelului va trasa urma pe ecran. La inițierea unei sesiuni de lucru LOGO penelul este implicit activ (penița jos).

EXEMPLU: (Fig. 11.17)

PENDOWN

FD 120



**PENDOWN
FD 120**

Fig. 11.17. Exemplu de utilizare a comenzii PENDOWN

Comanda PENERASE

PENERASE

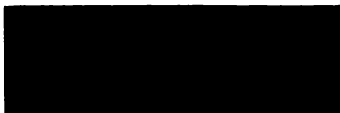
PE

Comanda PENERASE va șterge punctele peste care trece penelul dacă anterior acestea au fost desenate. Efectul comenzii PENERASE este anulat prin utilizarea oricăreia dintre comenzile PENDOWN, PENUP, sau PENREVERSE.

EXEMPLU: (Fig. 11.18)

PENERASE

FD 120



**PENERASE
FD 120**

Fig. 11.18. Exemplu de utilizare a comenzii PENERASE

Comanda PENREVERSE

PENREVERSE

PX

Comanda PENREVERSE va complementa punctele peste care trece penelul. Dacă acestea sînt desenate, atunci le șterge, iar dacă nu, atunci le desenează.

Efectul comenzii PENREVERSE este anulat prin utilizarea oricăreia dintre comenzile PENDOWN, PENUP sau PENERASE.

EXEMPLU: (Fig. 11.19)

PENREVERSE



PENREVERSE
FD 120

Fig. 11.19. Exemplu de utilizare a comenzii PENREVERSE

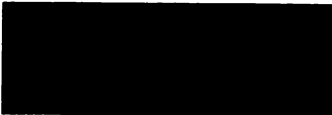
Comanda PENUP

PENUP
PU

Comanda PENUP face inactiv penelul, adică penița este ridicată și orice mișcare a sa nu va produce urmă pe ecran. Penelul nu va putea desena pînă cînd nu se activează acesta cu comanda PENDOWN.

EXEMPLU: (Fig. 11.20)

PENUP
FD 120



PENUP
FD120

Fig. 11.20. Exemplu de utilizare a comenzii PENUP

Comanda SETBG

SETBG nrc

Comanda SETBG stabilește culoarea fondului ecranului în regim grafic, în funcție de numărul culorii „nrc” specificat. Corespondența între numărul culorii „nrc” și culoarea fondului este următoarea:

0 — negru	4 — verde
1 — albastru	5 — turcoaz
2 — roșu	6 — galben
3 — purpuriu (magenta)	7 — alb

EXEMPLU:

SETBG 6

Culoarea galben, devine culoarea fondului în regim grafic.

Comanda SETBORDER

SETBORDER nrc
SETBR nrc

Comanda SETBORDER stabilește culoarea chenarului în funcție de numărul culorii „nrc” specificat. Corespondența între numărul culorii „nrc” și culoarea chenarului este aceeași cu cea de la comanda SETBG.

EXEMPLU:
SETBORDER 1

Culoarea albastru devine culoarea chenarului.

Comanda SETPC

SETPC nrc

Comanda SETPC, stabilește culoarea cu care penelul scrie pe ecran, în regim grafic, în funcție de numărul culorii „nrc” specificat.

Corespondența între numărul culorii „nrc” și culoarea cu care penelul scrie pe ecran este aceeași cu cea de la comanda SETBG.

Trebuie subliniat faptul că, la calculatorul HC-85, culoarea cu care se scrie (INK) poate fi stabilită numai la nivel de zonă caracter, nu la nivel de punct. De aceea o linie trasată cu o culoare nouă va schimba culoarea tuturor punctelor peste care trece și a punctelor din zona caracter asociată. Din acest motiv trebuie avut grijă de modul în care se schimbă culorile pentru ca desenul să nu piardă din acuratețe.

EXEMPLU:
SETPC 4

Culoarea cu care scrie penelul va fi culoarea verde.

Comanda SETSCRUNCH

SETSCRUNCH [x y]

Comanda SETSCRUNCH stabilește scala de reprezentare pe axele de coordonate, față de care se realizează un desen. La inițierea unei sesiuni de lucru LOGO, implicit scara de reprezentare este [100 100]. În urma unei comenzi SETSCRUNCH această scală se poate modifica. Astfel:

SETSCRUNCH [100 25]

modifică o figură ce se va desena, pe ecranul calculatorului în sensul că lățimea rămâne aceeași (scala pe x a rămas nemodificată), iar înălțimea este de 4 ori mai mică.

SETSCRUNCH [25 25]

va modifica figura, micșorînd de 4 ori atât înălțimea cât și lățimea.

Important de menționat faptul că dacă dorim să stabilim scala pe axele de coordonate în funcție de valorile unor variabile, va trebui să creăm o listă cu aceste variabile utilizînd funcția SENTENCE, astfel:

SETSCRUNCH SENTENCE :X :Y

EXEMPLU:

```
REPEAT 4[FD 80 RT 90]
SETSCRUNCH [100 25]
REPEAT 4 [FD 80 RT 90]
SETSCRUNCH [25 25]
REPEAT 4[FD 80 RT 90]
```

Pătratul desenat în urma comenzii SETSCRUNCH [100 25] va deveni un dreptunghi la care înălțimea este un sfert din lățime. Ultima comandă SET-

SCRUNCH [25 25], va face ca pătratul ce se va desena să aibă latura de 4 ori mai mică decât cel inițial.

După încercarea acestui exemplu este necesar să revenim la scala inițială de [100 100], printr-o comandă corespunzătoare.

Comanda **WINDOW**

WINDOW

Comanda **WINDOW** va permite ca penelul să poată fi poziționat în afara limitelor ecranului prezentate în Fig. 11.1. Putem face afirmația că ecranul are o dimensiune mare și numai o porțiune delimitată de limitele din Fig. 11.1 este vizibilă, Fig. 11.21.

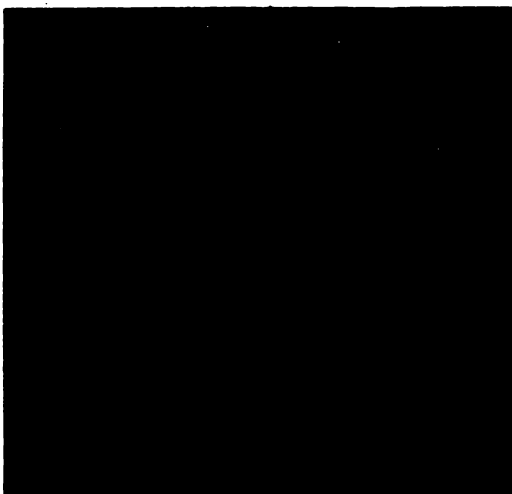


Fig. 11.21. Limitele zonei vizibile și zonei invizibile ale ecranului în cadrul comenzii **WINDOW**

EXEMPLU: (Fig. 11.22)

```
WINDOW  
CS  
REPEAT 4 [FD 120 RT 90]  
120
```



```
CS  
WINDOW  
REPEAT 4[FD120 RT90]
```

Fig. 11.22. Exemplu de utilizare a comenzii **WINDOW**

Practic, comanda **WINDOW** permite penelului să treacă din zona vizibilă în zona invizibilă, fără ca limbajul **LOGO** să emită mesaj de eroare. Orice mișcare a penelului în afara zonei vizibile se va executa, numai că utilizatorul nu o va vedea pe ecran.

La inițierea unei sesiuni de lucru **LOGO**, implicit se consideră atât zona vizibilă cât și cea invizibilă (ca și cum s-ar fi specificat inițial o comandă **WINDOW**).

De notat faptul că, dacă penelul se află în zona invizibilă, limitarea mișcării penelului numai la zona vizibilă printr-o comandă **FENCE** va produce un mesaj de eroare.

Din pătratul cu latura de 120 se va vedea pe ecran numai zona continuă.

Utilizarea unei comenzi **SETSCRUNCH [50 50]**, va modifica scala de reprezentare și deci figura se va micșora (latura pătratului se va înjumătăți) și se va putea vedea tot pătratul.

Comanda WRAP WRAP

Comanda WRAP, face ca zona vizibilă a ecranului să fie continuă (marginea de sus a ecranului să fie adiacentă cu marginea de jos, iar marginile laterale să fie de asemenea adiacente (Fig 11.23). În acest caz, cînd penelul traversează o latură a ecranului va reapare imediat în latura opusă considerată adiacentă.

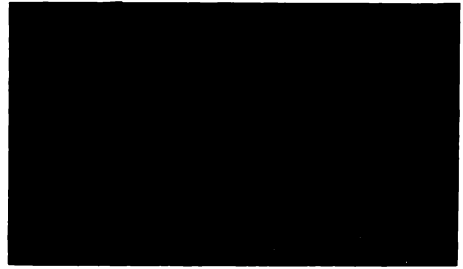


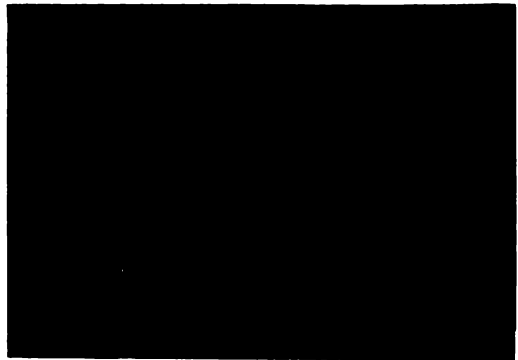
Fig. 11.23. Adiacenta laturilor ecranului în urma execuției comenzii WRAP

Punctul de coordonate (0, 88) este identic cu cel de coordonate (0, -88), iar punctul (0, 89) identic cu (0, -87).

EXEMPLU: (Fig. 11.24)

```
WRAP  
REPEAT 4 [FD 120 RT 90]
```

Zona din pătrat, care în cadrul exemplului din figura 11.22. era invizibilă, a fost desenată jos, considerîndu-se laturile de sus și jos ale ecranului ca fiind adiacente.



11.2.4. Primitive care specifică starea peniței și a ecranului.

În acest paragraf se descriu operațiile prin care utilizatorul poate afla informații privind starea peniței și a ecranului.

Fig. 11.24. Exemplu de utilizare a comenzii WRAP

Primitivele care vor fi tratate în acest paragraf sînt:

```
BACKGROUND  
PENCOLOUR  
SCRUNCH
```

Operația BACKGROUND

```
BACKGROUND  
BG
```

Operația BACKGROUND, generează un număr între 0 și 7 ce reprezintă culoarea fondului ecranului. Corespondența între numărul generat și culoarea fondului este aceeași cu cea descrisă la comanda SETBG.

La inițierea unei sesiuni de lucru LOGO, culoarea fondului este alb. Este operația inversă a comenzii SETBG.

Ex:
PRINT BACKGROUND
7

Operația PENCOLOUR

PENCOLOUR
PC

Operația PENCOLOUR, generează un număr între 0 și 7 ce reprezintă culoarea cu care scrie penelul. Corespondența între numărul generat și culoarea cu care scrie penelul este aceeași cu cea descrisă la comanda SETBG.

La inițierea unei sesiuni de lucru LOGO, culoarea cu care scrie penelul este negru.

Ex:
PRINT PENCOLOUR
0

Operația SCRUNCH

SCRUNCH

Operația SCRUNCH generează o listă formată din două numere care reprezintă scala pe axele de coordonate. La inițierea unei sesiuni de lucru LOGO, scala de reprezentare este [100 100].

11.3. Primitive LOGO pentru controlul ecranului în regim alfanumeric

În regim alfanumeric, ecranul este considerat o suprafață organizată în 22 de linii a câte 32 de caractere fiecare, (Fig. 11.25).



Fig. 11.25. Organizarea ecranului în mod alfanumeric

În regim de comandă în prima coloană este afișat prompterul ?, iar în regim de editare este afișat >.

Utilizatorul, cu ajutorul comenzilor LOGO poate stabili culoarea fondului ecranului și culoarea caracterului.

Afișarea se poate realiza:

— în video normal (video direct) constă în scrierea cu culoarea caracterului pe un fond care are culoarea fondului;

— în video invers constă în scrierea cu culoarea fondului pe un fond care are culoarea caracterului. Altfel spus se interschimbă (inversează) culoarea caracterului cu cea a fondului.

Primitivele care vor fi tratate în acest paragraf sînt:

BRIGHT	NORMAL
CLEARTEXT	OVER
COPYSCREEN	SETCURSOR
CURSOR	SETTC
FLASH	TEXTSCREEN
INVERSE	

Comanda BRIGHT

BRIGHT b

Comanda BRIGHT, stabilește nivelul de luminozitate a fondului ecranului în funcție de valoarea parametrului „b”, ce poate lua valori binare 1 sau 0. Dacă: b = 1 — comanda stabilește fond luminos;

· b = 0 — comanda stabilește fond normal.

EXEMPLU:

BRIGHT 1

Comanda CLEARTEXT

CLEARTEXT
CT

Comanda CLEARTEXT șterge ecranul. Dacă ecranul este în regim grafic, efectul comenzii CLEARTEXT constă în ștergerea a două linii din partea de jos a ecranului.

EXEMPLU: (Fig. 11.26)

CLEARTEXT



Fig. 11.26. Exemplu de utilizare a comenzii CLEARTEXT

Operația COPYSCREEN

COPYSCREEN

Operația COPYSCREEN copiază ecranul la imprimantă. Utilizatorul poate folosi această comandă indiferent de modul de organizare a ecranului alfanumeric sau grafic. Ea permite obținerea unui „hard copy”, la imprimantă.

Imprimanta trebuie să fie conectată în prealabil pe interfața paralelă co-respunzătoare.

Observație: Această interfață constituie o dezvoltare ulterioară a calculatorului.

EXEMPLU:

COPYSCREEN

Operația CURSOR

CURSOR

Operația CURSOR generează o listă formată din două numere care specifică coordonatele *coloană*, *linie* a poziției curente a cursorului în regim alfanumeric (poziția unde va apare primul caracter care va fi afișat pe ecran). Coordonatele coloană, linie vor fi în concordanță cu cele prezentate în figura 11.25.

La inițierea unei sesiuni de lucru LOGO, poziția inițială a cursorului este în partea stângă sus a ecranului.

EXEMPLU:

PRINT CURSOR

0 1

Comanda FLASH

FLASH

Comanda FLASH stabilește ca orice caracter ce se afișează pe ecran, după ce s-a executat această comandă, să apară în regim de clipire (se alternează, cu o anumită perioadă de timp, afișarea caracterului în video invers cu cea în video normal).

Ecranul rămâne în acest regim de afișare pînă cînd se dă o comandă NORMAL sau se produce o eroare.

EXEMPLU:

```
TO AFIȘ.FLASH
PRINT [afișare în mod normal]
FLASH
PRINT [afișare în mod clipire]
NORMAL
END
```

AFIȘ.FLASH

Observație: Funcția FLASH nu este activă în regim de comandă.

Comanda INVERSE

INVERSE

Comanda INVERSE are ca efect interschimbarea culorii caracterului cu

cea a fondului. Se rămîne în acest regim de afişare pînă la o comandă NORMAL care trece în regim normal sau se produce o eroare.

EXEMPLU:

```
TO AFIŞ.INV
PRINT [afişare în mod normal]
INVERSE
PRINT [afişare în mod invers]
NORMAL
END
AFIŞ.INV
```

Comanda NORMAL
NORMAL

Comanda NORMAL are ca efect readucerea la modul normal de scriere (scriere cu culoarea caracterului pe culoarea fondului), indiferent de modul de afişare în care se află ecranul, INVERSE sau FLASH. Comanda NORMAL anulează efectul comenzilor INVERSE şi FLASH.

EXEMPLU:

```
TO AFIŞARE
PRINT [afişare în mod normal]
FLASH
PRINT [afişare în mod clipire]
NORMAL
INVERSE
PRINT [afişare în mod invers]
NORMAL
PRINT [afişare în mod normal]
END
AFIŞARE
```

Comanda OVER
OVER b

Comanda OVER stabileşte modul de trasare sau scriere în funcţie de parametrul „b” ce poate lua valori binare 1 sau 0.

Dacă:

b = 1 Se va scrie peste desenul sau textul de pe ecran în mod SAU EXCLUSIV (punctele care sînt stinse vor fi aprinse, iar cele care sînt aprinse vor fi stinse de trecerea penelului în regim grafic sau a cursorului în regim alfanumeric). Se va rămîne în această stare de scriere pînă cînd o comandă OVER 0 va fi executată sau se va

produce o eroare;

b = 0 Se va scrie în regim normal.

EXEMPLU:

```
CLEARTEXT
TO AFIS.OVER
TS
SETCURSOR [10 1]
PRINT [textul se şterge în regim OVER]
```



```

OVER 1
WAIT 25
SETCURSOR [10 1]
PRINT [textul se șterge în regim OVER]
OVER 0
END
AFIȘ.OVER

```

Comanda SETCURSOR

SETCURSOR [c l]

SETCUR [c l]

Comanda SETCURSOR, mută cursorul în regim alfanumeric, în poziția determinată de coordonatele absolute c (coloana), l (linie) specificate sub formă de listă [c l]. Valorile parametrilor pot fi în următoarele limite:

— coloana „c” între 0 și 31;

— linia „l” între 0 și 21.

Depășirea valorilor extreme ale parametrilor va produce un mesaj de eroare.

Pentru poziționarea cursorului într-un punct de coordonate :X :Y reprezentate de valorile unor variabile va trebui să creăm o listă cu aceste variabile, utilizând operația SENTENCE sau operația LIST.

EXEMPLU:

```

TEXTSCREEN
CLEARSCREEN
SETCURSOR [3 8]
PRINT [text din coloana 3 linia 8]

```

Comanda SETTC

SETTC [f c]

Comanda SETTC stabilește culoarea fondului pe care se scrie și culoarea cu care se scrie caracterul în funcție de parametrii f (fond) și c (caracter), specificați.

Parametrii f și c sînt specificați sub formă de listă [f c] și pot lua valori numerice între 0 și 7.

Correspondența între numărul culorii fondului f, respectiv a caracterului c și culorile utilizate este aceeași cu cea prezentată la comanda SETBG.

Pentru stabilirea culorilor fondului și caracterului în funcție de valorile unor variabile :F :C, va trebui să creăm o listă cu ele utilizând operațiile SENTENCE sau LIST.

EXEMPLU:

```

SETBORDER 1
SETTC [6 2]

```

Se stabilește chenarul albastru și se scrie cu culoarea roșie pe fond galben.

Comanda TEXTSCREEN

**TEXTSCREEN
TS**

Comanda TEXTSCREEN trece ecranul în regim alfanumeric.

În acest mod de lucru al ecranului nu se vede penelul și nici desenele realizate cu el, ci numai textul procedurilor sau textul realizat prin execuția programelor.

EXEMPLU:

TEXTSCREEN

11.4. Primitive LOGO pentru operații matematice și logice

În acest capitol se prezintă operatorii și primitivele care manipulează numere în vederea realizării unor operații matematice.

În limbaj LOGO se lucrează cu operanzi ce fac parte din:

- mulțimea numerelor întregi Z ;
- mulțimea numerelor reale R .

Reprezentarea operanzilor din mulțimea R poate fi realizată în două moduri:

a) prin specificarea părții întregi și zecimale separate prin punct (.)

EXEMPLU: 2.793; -796.69189

Punctul (.) înlocuiește virgula (,) pe care o utilizăm în scrierea numerelor zecimale la matematică.

b) prin specificarea mantisei și a exponentului:

<mantisa> {E} <exponent>

EXEMPLU: 2.793162E4 \equiv 27931.62

8.7134E-3 \equiv 0.0087134

Valorile: 2.793162, respectiv 8.7134 reprezintă mantisa numărului iar E 4 reprezintă 10^4 , respectiv E-3 reprezintă 10^{-3} ;

EXEMPLU: 3.791E2 \equiv $3.791 \times 10^2 = 379.1$

Utilizarea reprezentării cu mantisă și exponent este necesară în cazul în care avem de reprezentat valori foarte mari sau foarte mici.

Domeniul de reprezentare a valorilor numerelor este:

- în mulțimea numerelor întregi: $-99999999 < N < 99999999$
- în mulțimea numerelor reale: $-10^{38} < N < 10^{38}$, iar cea mai mică valoare de reprezentare este: 0.000001×10^{-38} .

În cadrul reprezentării cu mantisă și exponent, mantisa se poate exprima prin maxim 8 cifre semnificative, iar exponentul este cuprins între -38 și $+38$.

Regulile de evaluare a expresiilor sînt aceleași cu cele din matematică.

O precizare suplimentară care trebuie făcută este aceea că, pentru primitivele matematice delimitarea unei expresii de parametri asupra căreia se aplică operatorul se realizează cu sau fără paranteze.

EXEMPLU:

COS 25+10 \equiv COS (25+10)

COS 25+10 \equiv (COS 25+10)

11.4.1. Operatorii aritmetici și logici

În acest paragraf se descriu operatorii aritmetici și logici cu care se pot defini expresii. Deoarece simbolurile operatorilor constituie separatori de cuvinte, spațiul între termeni și operatori este opțional.

Excepție face operatorul de împărțire (/) care trebuie precedat și urmat de spațiu.

Operatorul +

$nr1 + nr2$

Operatorul „+”, calculează suma celor doi termeni nr1, nr2 specificați. Termenii pot fi numere întregi sau reale.

EXEMPLU:

```
PRINT 7 + 9.652
16.652
```

Operatorul -

$nr1 - nr2$

Operatorul „-”, calculează diferența între cei doi termeni nr1 și nr2. Dacă termenul nr1 este omis și nu se lasă spațiu între „-” și nr2, se generează numărul nr2 cu semn schimbat.

Termenii pot fi numere întregi sau reale.

EXEMPLU:

```
PRINT 8 - 2
6
PRINT 8-2
6
PRINT -3 - -2
-1
```

Trebuie avut grijă în Interpretarea simbolului „-”, ca operator sau ca semn minus.

EXEMPLU:

```
7- 1 - generează valoarea 6
7 - 1 - generează valoarea 6
7- 1 - generează valoarea 6
7 -1 - reprezintă o pereche de numere formată din 7 și -1 (și nu
operația 7-1)
```

Operatorul *

$nr1 * nr2$

Operatorul „*”, calculează produsul între cei doi factori. Factorii pot fi numere întregi sau reale.

EXEMPLU:

```
PRINT 1 * 2 * 3
```

6
PRINT 2 * -1.46
 -2.92

Operatorul /

nr1 / nr2

Operatorul „/“, calculează cîtu între cei doi factori. Factorii pot fi numere întregi sau reale, rezultatul fiind întotdeauna reprezentat în real. Împărțitorul nr2 trebuie să fie diferit de zero. Între operator și factori, trebuie lăsat cîte un spațiu.

EXEMPLU:

```
PRINT 6 / 2
3.0
PRINT 5 / 7.6
0.657895
PRINT 0 / 3
0
PRINT 7 / 0
,, , can't divide by zero
(emite mesajul că nu poate împărți la zero)
```

Operatorul <

nr1 < nr2

Operatorul „<“ (mai mic), generează valoarea logică TRUE, dacă numărul nr1 este strict mai mic decît nr2. În caz contrar generează valoarea logică FALSE.

EXEMPLU:

```
PRINT 2 < 3          — generează valoarea logică TRUE
TRUE
PRINT -1.25 < -4.65 — generează valoarea logică FALSE
FALSE
```

Operatorul =

obiect 1 = obiect 2

Operatorul „=“ (egal), generează valoarea logică TRUE, dacă obiectele obiect 1 și obiect 2 sînt două numere egale, două cuvinte identice sau două liste identice. În caz contrar generează valoarea logică FALSE.

EXEMPLU:

```
PRINT 50 = 2 * 25    — generează valoarea logică TRUE
TRUE
PRINT 7.0 = 7        — generează valoarea logică TRUE
TRUE                 (numărul real este echivalent cu corespunzătorul întreg)

PRINT 60 = 70 - 10
TRUE
'PIONIER = WORD 'PION 'IER — generează valoarea logică TRUE
TRUE
PRINT 60 = 70 - 10
FALSE
you don 't say what to do with -10
```

În expresia de mai sus $70 > -10$ nu reprezintă diferența între 70 și -10, ci reprezintă o pereche de numere 70 și -10. Se evaluează, dacă $60 = 70$ și se generează valoarea logică FALSE, după care se pune întrebarea ce trebuie făcut cu -10.

Operatorul >

nr1 > nr2

Operatorul „>” (mai mare), generează valoarea logică TRUE, dacă numărul nr1 este strict mai mare decât nr2.

În caz contrar, generează valoarea logică FALSE.

EXEMPLU:

```
PRINT 4>3
TRUE
PRINT -15>-8
FALSE
```

11.4.2.. Primitive pentru operații matematice

În acest paragraf se vor descrie primitivele cu care utilizatorul poate specifica operații matematice.

Primitivele care vor fi tratate în acest paragraf sînt:

ARCCOS	PRODUCT
ARCCOT	RANDOM
ARCSIN	REMAINDER
ARCTAN	ROUND
COSINE	SINE
COTANGENT	SQRT
DIV	SUM
INT	TANGENT

Operația ARCCOS

ARCCOS nr

Operația ARCCOS calculează valoarea, în grade, a funcției arccosinus de argument „nr”.

EXEMPLU:

```
PRINT ARCCOS 0.45
63.256316
PRINT ARCCOS 1
90
```

Operația ARCCOT

ARCCOT nr

Operația ARCCOT calculează valoarea, în grade, a funcției arccotangent de argument „nr”.

EXEMPLU:
PRINT ARCCOT 1
45

Operația ARCSIN

ARCSIN nr

Operația ARCSIN calculează valoarea, în grade, a funcției arcsinus de argument „nr“.

EXEMPLU:
PRINT ARCSIN 0.45
26.743684

Operația ARCTAN

ARCTAN nr

Operația ARCTAN calculează valoarea, în grade, a funcției arctangent de argument „nr“.

EXEMPLU:
PRINT ARCTAN 1
45

Operația COSINE

COSINE gr
COS gr

Operația COSINE calculează valoarea funcției cosinus de argument „gr“. Argumentul „gr“ se specifică în grade.

EXEMPLU:
PRINT COS 60
0.5

Operația COTANGENT

COTANGENT gr
COT gr

Operația COTANGENT calculează valoarea, funcției cotangent de argument „gr“. Argumentul „gr“ se specifică în grade.

EXEMPLU:
PRINT COT 45
1

Operația DIV

DIV nr1 nr2

Operația DIV calculează cîtul împărțirii, în care nr1 este deîmpărțitul, iar nr2 este împărțitorul. Împărțitorul nr2 trebuie să fie diferit de zero ($nr2 \neq 0$).

Operația este asemănătoare cu operatorul „/“.

EXEMPLU:

```
PRINT DIV 78 2
39
PRINT DIV -18.64 2
-9.32
PRINT DIV 4 0
,,can't divide by zero (nu se poate împărți la zero)
```

Operația INT

INT nr

Operația INT generează partea întreagă, prin trunchiere, a argumentului „nr“. Dacă argumentul este număr întreg, operația INT nu are nici un efect.

EXEMPLU:

```
PRINT INT 6.1279
6
PRINT INT 6.9321
6
PRINT INT 6
6
PRINT INT -12.3
-12
```

Operația PRODUCT

PRODUCT nr1 nr2

(PRODUCT nr1 nr2 nr3 ... nrm)

Operația PRODUCT calculează produsul între factorii nr1, nr2 ... nrm. Când sînt mai mult de doi factori, este necesară utilizarea parantezelor ().

EXEMPLU:

```
PRINT PRODUCT 5 8
40
(PRINT PRODUCT 1 2 3 4 5)
120
```

Operația RANDOM

RANDOM nri

Operația RANDOM generează un număr întreg pozitiv, aleator, cuprins între 0 și nri -1. Argumentul „nri“ trebuie să fie un număr întreg pozitiv (sau real pozitiv)

EXEMPLU:

```
RANDOM 5
- generează oricare din valorile 0, 1, 2, 3, 4.
```

Trebuie avut în vedere că evaluarea expresiilor:

1+RANDOM 7

RANDOM 7 + 1

— se face în mod diferit și anume:

a) — în primul caz la numărul aleator generat între 0 și 6, se adună valoarea 1;

b) — în al doilea caz se adună 7 cu 1 și apoi se generează un număr aleator între 0 și 7.

$1 + \text{RANDOM } 7 \equiv (\text{RANDOM } 7) + 1$

Operația REMAINDER

REMAINDER nr1 nr2

Operația REMAINDER calculează restul împărțirii argumentelor nr1 la nr2. Dacă argumentele nu sînt întregi atunci ele se trunchiază și apoi se calculează restul împărțirii. Argumentul nr2 trebuie să fie diferit de zero.

EXEMPLU:

```
PRINT REMAINDER 17 4
1
PRINT REMAINDER 17 8
1
PRINT REMAINDER -12 5
-2
```

Operația ROUND

Round nr

Operația ROUND generează cel mai apropiat întreg față de argumentul „nr”. Dacă argumentul este egal depărtat de valoarea întregă superioară lui și valoarea întregă inferioară lui, atunci se generează întregul a cărui valoare absolută este mai mare.

EXEMPLU:

```
PRINT ROUND 7.4896
7
PRINT ROUND 7.5087
8
PRINT ROUND 7.5
8
PRINT ROUND -7.5
-8
```

Operația SINE

SINE gr

SIN gr

Operația SINE calculează valoarea funcției sinus de argument „gr”. Argumentul „gr” se specifică în grade.

EXEMPLU:

```
PRINT SINE 30
0.5
```

Operația SQRT

SQRT nr

Operația SQRT (rădăcina pătrată), calculează valoarea funcției radical de ordinul 2 din argumentul „nr”. Argumentul „nr” trebuie să fie pozitiv.

EXEMPLU:

```
PRINT SQRT 36
6
PRINT SQRT 259
16.0935
PRINT SQRT 3 * 3 + 4 * 4
5
```

Operația SUM

SUM nr1 nr2
(SUM nr1 nr2 ... nrm)

Operația SUM calculează suma termenilor nr1 nr2 ... nrm. Cind sînt mai mult de doi termeni este necesară utilizarea parantezelor ().

EXEMPLU:

```
PRINT SUM 2 7
9
PRINT (SUM 4 7 -1 9 -2)
17
```

Operația TANGENT

TANGENT gr
TAN gr

Operația TANGENT, calculează valoarea funcției tangent de argument „gr“. Argumentul „gr“ se exprimă în grade.

EXEMPLU:

```
PRINT TANGENT 50
1.1917536
```

11.4.3. Primitive pentru operații logice

În acest paragraf se descriu primitivele care permit descrierea funcțiilor logice.

Există trei primitive, care formează sistem complet de funcții (cu ajutorul lor se poate descrie orice funcție logică) și anume: AND, NOT, OR.

Tabelele de adevăr a acestor operatori logici sînt prezentate în Tabelul 11.1.

Tabelul 11.1 — Tabelele de adevăr a funcțiilor AND, NOT, OR

<u>arg1</u>	<u>arg2</u>	<u>AND</u>	<u>arg</u>	<u>NOT</u>	<u>arg1</u>	<u>arg2</u>	<u>OR</u>
FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE
FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	TRUE
TRUE	FALSE	FALSE			TRUE	FALSE	TRUE
TRUE	TRUE	TRUE			TRUE	TRUE	TRUE

Se definește noțiunea de predicat ca fiind o funcție care generează ca rezultat una din valorile logice TRUE sau FALSE.

11. PRIMITIVE LOGO

Putem considera că operațiile AND, OR, NOT necesită ca intrări predicate, rezultatele produse de ele fiind tot predicate.

Operația AND

AND pred1 pred2
(AND pred1 pred2... predn)

Operația AND generează valoarea logică TRUE numai dacă toate argumentele (predicatele) de intrare au valoarea logică TRUE. În caz contrar generează valoarea logică FALSE. (Calculează funcția SI între argumentele de intrare) (Tabelul 11.1). Dacă sînt mai mult de două argumente, atunci este necesară utilizarea parantezelor ().

EXEMPLU:

```
PRINT AND TRUE TRUE
TRUE
SETPC 4
PRINT AND PENCOLOR-2 TRUE
FALSE
```

Operația NOT

NOT pred

Operația NOT generează valoarea logică TRUE, dacă argumentul (predicatul) de intrare este FALSE, iar dacă argumentul este TRUE generează valoarea logică FALSE.

EXEMPLU:

```
PRINT NOT 2>3
TRUE
```

Operația OR

OR pred1 pred2
(OR pred1 pred2 ... predn)

Operația OR generează valoarea logică FALSE, dacă toate argumentele (predicatele) de intrare au valoarea logică FALSE. În caz contrar generează valoarea logică TRUE (Tabelul 11.1). Dacă sînt mai mult de două argumente, atunci este necesar să se utilizeze parantezele ().

EXEMPLU:

```
PRINT OR FALSE TRUE
TRUE
```

11.5. Primitive LOGO pentru lucrul cu cuvinte și liste

În acest capitol se descriu primitive care permit lucrul, cu două tipuri de obiecte și anume:

- lucrul cu cuvinte;
- lucrul cu liste.

Cuvîntul este un obiect LOGO, constituit dintr-un șir de elemente de tip caracter. Un cuvînt LOGO se specifică printr-un șir de caractere prefixat de ghilimele (").

EXEMPLU:

```
PRINT "CUVÎNT
CUVÎNT
```

Delimitarea cuvîntelor se face cu ajutorul caracterelor speciale: (spațiu), (,), [,], < , > + , - , * , / .

Caracterul special ghilimelele (")

— este utilizat pentru a specifica faptul că șirul de caractere ce urmează este un cuvînt.

Caracterul special două puncte (:)

— este utilizat înaintea unui șir de caractere pentru a preciza că se lucrează cu conținutul variabilei specificată prin cuvîntul respectiv.

Există un cuvînt vid, specificat de ghilimele (") urmate de spațiu, care este un cuvînt fără nici un caracter (element).

Dacă dorim ca în cadrul unui cuvînt să apară unul din semnele speciale precizate anterior și acesta să nu fie interpretat ca delimitator trebuie ca să fie precedat de semnul (\) (slash invers SS D).

Lista este un obiect LOGO, constituit dintr-un șir de cuvinte sau de subliste delimitat de paranteze drepte [] .

EXEMPLU:

```
PRINT [ACEASTA ESTE O LISTĂ]
ACEASTA ESTE O LISTĂ
```

Lista vidă se specifică cu [] , fără nici un caracter sau cuvînt în interior. Cuvintele sau sublistele constituie elemente ale listei.

Astfel, lista: [ACEASTA ESTE O LISTĂ], are patru elemente:

ACEASTA

ESTE

O

LISTĂ

Lista [1 [2 3] [4 [5 6]]] — conține numai trei elemente, nu șase cum pare la prima vedere, și anume:

1 — element de tip cuvînt;

[2 3] — element de tip listă;

[4 [5 6]] — element de tip listă compusă.

Lista [[ACEASTA ESTE O LISTĂ] [CARE ARE PROPRIETĂȚILE]], conține două elemente:

[ACEASTA ESTE O LISTĂ] — element de tip listă;

[CARE ARE PROPRIETĂȚILE] — element de tip listă.

Caracterele cu care lucrează limbajul LOGO se împart în:

— caractere alfanumerice: — litere: (A, B, C, ..., Z);

— cifre: (0, 1, 2, ..., 9);

— caractere speciale: (spațiu) ! " # \$ % & ' () * + - . , / : ;

< = > ? [;] — { } ~

Pentru a reprezenta un caracter în memoria calculatorului acesta se pune în corespondență cu un cod binar pe 7 biți, conform unui standard internațional numit ASCII (American Standard Code, for Information Interchange).

Fiecare caracter din memoria calculatorului este reprezentat prin codul ASCII, corespunzător.

11.5.1, Primitive care specifică preluarea unor elemente din cuvinte sau liste

În acest paragraf se descriu operațiile care permit descompunerea și specificarea unor elemente din cadrul unor cuvinte sau liste.

Se vor descrie primitivele:

BUTFIRST **LAST**
BUTLAST **ITEM**
FIRST

Operația **BUTFIRST**

BUTFIRST obiect
BF obiect

Operația **BUTFIRST** generează obiectul specificat, fără primul element al său. Dacă operația se aplică unui obiect vid, se generează un mesaj de eroare.

EXEMPLU:

```
PRINT BUTFIRST "MARE  
ARE  
PRINT BUTFIRST [ACEASTA ESTE O LISTĂ]  
ESTE O LISTĂ  
PRINT BUTFIRST [[ACEASTA ESTE O LISTĂ] [CARE ARE  
PROPRIETĂȚILE]]  
CARE ARE PROPRIETĂȚILE  
PRINT BUTFIRST [POPESCU CRISTIAN]  
CRISTIAN
```

Operația **BUTLAST**

BUTLAST obiect
BL obiect

Operația **BUTLAST** generează obiectul specificat, fără ultimul element al său. Dacă operația se aplică unui obiect vid, se generează un mesaj de eroare.

EXEMPLU:

```
PRINT BUTLAST "PIONIERI  
PIONIER  
PRINT BUTLAST [ACEASTA ESTE O LISTĂ]  
ACEASTA ESTE O  
PRINT BUTLAST [[ACEASTA ESTE O LISTĂ] [CARE ARE  
PROPRIETĂȚILE]]  
ACEASTA ESTE O LISTĂ  
PRINT BUTLAST [POPESCU CRISTIAN]  
POPESCU
```

Operația FIRST

FIRST obiect

Operația FIRST generează primul element al obiectului specificat. Dacă operația se aplică unui obiect vid, se generează un mesaj de eroare. Operația FIRST aplicată unui cuvânt generează un singur caracter, iar aplicată unei liste generează un cuvânt sau o listă în funcție de structura listei specificate ca argument.

EXEMPLU:

```
PRINT *PIONIER
P
PRINT FIRST [PIONIER]
PIONIER
PRINT FIRST [ACEASTA ESTE O LISTĂ]
ACEASTA
PRINT FIRST [[ACEASTA ESTE O LISTĂ] [CARE ARE
PRIORITĂȚILE]]
ACEASTA ESTE O LISTĂ
PRINT FIRST [POPESCU CRISTIAN]
POPESCU
```

Operația LAST

LAST obiect

Operația LAST generează ultimul element al obiectului specificat. Dacă operația se aplică unui obiect vid, se generează un mesaj de eroare. Operația LAST aplicată unui cuvânt generează un singur caracter, iar aplicată unei liste generează un cuvânt sau o listă în funcție de structura listei specificate ca argument.

EXEMPLU:

```
PRINT LAST *PIONIER
R
PRINT LAST [PIONIER]
PIONIER
PRINT LAST [ACEASTA ESTE O LISTĂ]
LISTA
PRINT LAST [[ACEASTA ESTE O LISTĂ] [CARE ARE
PROPRIETĂȚILE]]
CARE ARE PROPRIETĂȚILE
```

Operația ITEM

ITEM nr obiect

Operația ITEM generează elementul a cărui poziție în cadrul obiectului specificat corespunde numărului „nr”.

De exemplu: — dacă nr = 4, operația ITEM extrage al patrulea element din cadrul obiectului specificat.

Operația ITEM se poate aplica unor obiecte de tip cuvânt sau listă.

11. PRIMITIVE LOGO

Dacă operația se aplică unui obiect vid, sau numărul „nr“ este mai mare decât numărul de elemente ale obiectului se generează un mesaj de eroare.

EXEMPLU:

```
PRINT ITEM 4 "PIONIER
N
PRINT ITEM 2 [PIONIER]
not enough items in [PIONIER] — mesaj de eroare deoarece:
lista [PIONIER] are un singur element
PRINT ITEM 4 [IONESCU POPESCU GEORGESCU PETRESCU
MATEESCU]
PETRESCU
PRINT ITEM 2 [[ACEASTA ESTE O LISTĂ] [CARE ARE
PROPRIETĂȚILE]]
CARE ARE PROPRIETĂȚILE
```

11.5.2. Primitive care concatenează cuvinte și liste

În acest paragraf se descriu operațiile care concatenează (pune împreună), cuvinte sau liste.

Se vor descrie primitivele:

```
FPUT SENTENCE
LIST WORD
LPUT
```

Operația FPUT

FPUT obiect listă

Operația FPUT generează o nouă listă care se formează prin introducerea obiectului specificat în cadrul operației, ca prim element în cadrul listei specificate ca argument.

EXEMPLU:

```
FPUT "IONESCU [POPESCU GEORGESCU]
— generează lista [IONESCU POPESCU GEORGESCU]
FPUT [ACEASTA ESTE O LISTĂ] [CARE ARE ELEMENTELE]
— generează lista [ [ACEASTA ESTE O LISTĂ]. CARE ARE ELEMEN-
TELE]
FPUT "PIONIER [ ]
— generează lista [PIONIER]
FPUT [ACEASTA ESTE O LISTĂ] [ ]
— generează lista: [ [ACEASTA ESTE O LISTĂ] ]
```

Operația LIST

LIST obiect1 obiect2

(LIST obiect1 obiect2 ... obiectn)

Operația LIST, generează o listă care are ca elemente obiectele specificate ca argumente. Trebuie notat faptul că dacă sînt mai mult de două argumente este necesară utilizarea parantezelor.

EXEMPLU:

- LIST "IONESCU [POPESCU GEORGESCU]
- generează lista: [IONESCU [POPESCU GEORGESCU]]
- (LIST "IONESCU "POPESCU "GEORGESCU)
- generează lista: [IONESCU POPESCU GEORGESCU]
- LIST "PIONIER []
- generează lista: [PIONIER []]
- LIST [ACEASTA ESTE O LISTĂ] [CARE ARE ELEMENTELE]
- generează lista: [[ACEASTA ESTE O LISTĂ] [CARE ARE ELEMENTELE]]

Operația LPUT

LPUT obiect lista

Operația LPUT generează o nouă listă, care se formează prin introducerea obiectului specificat în cadrul operației, ca ultim element în cadrul listei specificate ca argument.

EXEMPLU:

- LPUT "IONESCU [POPESCU GEORGESCU]
- generează lista: [POPESCU GEORGESCU IONESCU]
- LPUT [URMĂTOARE:] [CARE ARE ELEMENTELE]
- generează lista: [CARE ARE ELEMENTELE [URMĂTOARE:]]
- LPUT "PIONIER []
- generează lista: [PIONIER]
- LPUT [ACEASTA ESTE O LISTĂ] []
- generează lista: [[ACEASTA ESTE O LISTĂ]]

Operația SENTENCE

SENTENCE obiect1 obiect2**SE obiect1 obiect2**

(SENTENCE obiect1 obiect2 ... obiectn)

(SE obiect1 obiect2 ... obiectn)

Operația SENTENCE generează o listă care are ca elemente conținutul elementelor specificate ca argumente.

Dacă sint mai mult de două obiecte specificate este necesară utilizarea parantezelor.

EXEMPLU:

- SENTENCE "POPESCU "CRISTIAN
- generează lista: [POPESCU CRISTIAN]
- SENTENCE "IONESCU [POPESCU GEORGESCU]
- generează lista: [IONESCU POPESCU GEORGESCU]
- SENTENCE [ACEASTA ESTE O LISTĂ] [CARE ARE ELEMENTELE:]
- generează lista: [ACEASTA ESTE O LISTĂ CARE ARE ELEMENTELE]
- SENTENCE "PIONIER []
- generează lista [PIONIER]
- (SENTENCE "IONESCU "POPESCU "GEORGESCU)
- generează lista [IONESCU POPESCU GEORGESCU]

Reamintim că sint primitive LOGO, cum ar fi SETPOS [x y], DOT [x y], care necesită ca parametru, o listă. Când coordonatele x y sint valorile unor

variabile :X :Y, nu se poate scrie SETPOS [: X :Y], deoarece se generează un mesaj de eroare.

În acest caz, se folosește operația SENTENCE pentru a crea o listă cu parametrii respectivi.

EXEMPLU:

SETPOS SENTENCE :X :Y

Deosebirea între SENTENCE și LIST constă în faptul că:

— operația SENTENCE generează o listă alcătuită cu conținutul obiectelor specificate;

— operația LIST generează o listă alcătuită cu obiecte specificate.

EXEMPLU:

SENTENCE [ELEVUL POPESCU] [ARE [NOTE MARI]]
 — generează lista: [ELEVUL POPESCU ARE [NOTE MARI]]
LIST [ELEVUL POPESCU] [ARE [NOTE MARI]]
 — generează lista: [[ELEVUL POPESCU] [ARE [NOTE MARI]]]

Operația WORD

WORD cuvint1 cuvint2

(WORD cuvint1 cuvint2 ... cuvintn)

Operația WORD, generează un cuvânt obținut prin contatenarea (alipirea) cuvintelor specificate ca argument de intrare.

EXEMPLU:

PRINT WORD "ION " ESCU
 IONESCU
PRINT (WORD "P "O "P "E "S "C "U)
 POPESCU
PRINT "WORD "ANA "\ — "MARIA)
 ANA-MARIA
PRINT (WORD "ADRI "ANA "\ TAPUS)
 ADRIANA TAPUS

Observație — În versiunea implementată pe HC-85 primitiva WORD interpretează cuvintele formate din cifre ca numere. În acest caz, cuvintele ce au mai mult de 8 cifre sînt transformate în reprezentarea cu mantisă și exponent.

Pentru a înțelege mai bine cum lucrează primitivele FPUT, LIST, LPUT, SENTENCE, WORD, vom da câteva exemple care compară efectele acestora:

Operație	Arg. 1	Arg. 2	Ieșire
FPUT	"POPESCU	"CRISTIAN	eroare — arg 2 trebuie să fie listă
LIST	"POPESCU	"CRISTIAN	[POPESCU CRISTIAN]
LPUT	"POPESCU	"CRISTIAN	eroare — arg 2 trebuie să fie listă
SENTENCE	"POPESCU	"CRISTIAN	[POPESCU CRISTIAN]
WORD	"POPESCU	"CRISTIAN	POPESCU CRISTIAN

Operația	Arg. 1	Arg. 2	Ieșire
FPUT	"CRISTIAN	[ESTE ELEV]	[CRISTIAN ESTE ELEV]
LIST	"CRISTIAN	[ESTE ELEV]	[CRISTIAN [ESTE ELEV]]
LPUT	"CRISTIAN	[ESTE ELEV]	[ESTE ELEV CRISTIAN]
SENTENCE	"CRISTIAN	[ESTE ELEV]	[CRISTIAN ESTE ELEV]
WORD	"CRISTIAN	[ESTE ELEV]	eroare — arg 2 trebuie să fie cuvint
<hr/>			
FPUT	[POPESCU CRISTIAN]	[ESTE ELEV]	[[POPESCU CRISTIAN] ESTE ELEV]
LIST	[POPESCU CRISTIAN]	[ESTE ELEV]	[[POPESCU CRISTIAN] [ESTE ELEV]]
LPUT	[POPESCU CRISTIAN]	[ESTE ELEV]	[ESTE ELEV [POPESCU CRISTIAN]]
SENTENCE	[POPESCU CRISTIAN	[ESTE ELEV]	[POPESCU CRISTIAN ESTE ELEV]
WORD	[POPESCU CRISTIAN]	[ESTE ELEV]	eroare — argumentele trebuie să fie cuvinte
<hr/>			
FPUT	"CRISTIAN	[]	[CRISTIAN]
LIST	"CRISTIAN	[]	[CRISTIAN [.]]
LPUT	"CRISTIAN	[]	[CRISTIAN]
SENTENCE	"CRISTIAN	[]	[CRISTIAN]
WORD	"CRISTIAN	[]	eroare

11.5.3. Primitive care examinează cuvinte și liste

În acest paragraf se descriu primitivele care permit utilizatorului să examineze cuvinte și liste în vederea obținerii unor informații despre acestea.

Se vor descrie primitivele:

```

ASCII      LISTP
CHAR       MEMBERP
COUNT    NUMBERP
EMPTYP    WORDP
EQUALP

```

Operația ASCII

ASCII caracter

Operația ASCII generează codul ASCII (American Standard Code for Information Interchange), al caracterului specificat.

EXEMPLU

```

PRINT ASCII "A
65
(65 = valoarea în zecimal al codului ASCII (010000012) pus în corespun-
dență cu caracterul A).
PRINT ASCII "B
66

```

11. PRIMITIVE LOGO

Operația CHAR

CHAR nr

Operația CHAR generează caracterul al cărui cod ASCII este numărul „nr“. Numărul „nr“, trebuie să fie un întreg cuprins între 32 și 165. Dacă argumentul specificat în cadrul operației CHAR nu este un cod ASCII valid se va genera un mesaj de eroare.

EXEMPLU:

```
PRINT CHAR 65
A
PRINT CHAR 66
B
PRINT WORD CHAR 65 CHAR 66
AB
```

Operația COUNT

COUNT obiect

Operația COUNT generează un număr care reprezintă cite elemente are obiectul (cuvînt sau listă) specificat.

```
EXEMPLU: PRINT COUNT "POPESCU
7
PRINT COUNT [POPESCU CRISTIAN ESTE ELEV]
4
PRINT COUNT [[POPESCU CRISTIAN] [ESTE ELEV]]
2
```

Operația EMPTY

EMPTY obiect

Operația EMPTY, generează valoarea logică TRUE, dacă obiectul specificat este vid (nu are nici un element) sau valoarea logică FALSE în caz contrar.

EXEMPLU:

```
PRINT EMPTY "CRISTIAN
FALSE
PRINT EMPTY BUTFIRST "A
TRUE
PRINT EMPTY BUTLAST [ELEV]
TRUE
PRINT EMPTY ITEM 2 [POPESCU [ ] [ESTE ELEV]]
TRUE
```

Operația EQUALP

EQUALP obiect1 obiect2

Operația EQUALP generează valoarea logică TRUE, dacă obiectele obiect1 și obiect2, sînt două numere egale; două cuvinte identice sau două

liste identice. În caz contrar generează valoarea logică FALSE. Această operație este echivalentă cu operatorul (=).

EXEMPLU:

```
PRINT EQUALP "CRISTIAN LAST [POPESCU CRISTIAN]
TRUE
PRINT EQUALP 40 2 * 20
TRUE
PRINT EQUALP [EL ESTE [ELEV]] [EL ESTE ELEV]
FALSE
PRINT EQUALP " [ ]
FALSE      (cuvântul vid nu este identic cu lista vidă)
```

Operația LISTP

LISTP obiect

Operația LISTP generează valoarea logică TRUE dacă obiectul specificat este o listă, sau valoarea logică FALSE în caz contrar.

EXEMPLU:

```
PRINT LISTP 243
FALSE
PRINT LISTP [243]
TRUE
PRINT LISTP [ ]
TRUE      (lista vidă)
PRINT LISTP "
FALSE     (cuvântul vid nu este identic cu lista vidă)
PRINT LISTP BUTFIRST [ELEV]
TRUE
```

Operația MEMBERP

MEMBERP obiect lista

Operația MEMBERP generează valoarea logică TRUE, dacă obiectul specificat este un element al listei. În caz contrar, generează valoarea logică FALSE.

EXEMPLU:

```
PRINT MEMBERP "ESTE [ACEASTA ESTE O LISTĂ]
TRUE
PRINT MEMBERP "IONESCU [POPESCU IONESCU GEORGESCU]
TRUE
PRINT MEMBERP "IONESCU [POPESCU [IONESCU GEORGESCU]]
FALSE
PRINT MEMBERP "ESCU [POPESCU IONESCU]
FALSE

TO VOCALA :L
PRINT MEMBERP :L [A E I O U]
END

VOCALA A
TRUE
VOCALA B
FALSE
```

Operația NUMBERP

NUMBERP obiect

Operația NUMBERP generează valoarea logică TRUE dacă obiectul specificat este număr și valoarea logică FALSE în caz contrar.

EXEMPLU:

```
PRINT NUMBERP 3.4
TRUE
PRINT NUMBERP [3.4]
FALSE
PRINT NUMBERP 3.1E2
TRUE
PRINT NUMBERP BUTFIRST 8.693
TRUE
PRINT NUMBERP "
FALSE
```

Operația WORDP

WORDP obiect

Operația WORDP generează valoarea logică TRUE, dacă obiectul specificat este cuvânt și valoarea logică FALSE, în caz contrar. Trebuie avut în vedere, că în LOGO numerele sint considerate cuvinte.

EXEMPLU:

```
PRINT WORDP "ELEV
TRUE
PRINT WORDP 473
TRUE
PRINT WORDP [473]
FALSE
PRINT WORDP "
TRUE
PRINT BUTFIRST WORDP [ELEV]
FALSE
```

11.6. Primitive LOGO care lucrează asupra variabilelor

În acest capitol se descrie cum se definesc și cum se utilizează variabilele în limbajul LOGO.

Prin variabilă înțelegem o zonă de memorie unde se păstrează un obiect LOGO (număr, cuvânt sau listă). Zona de memorie are un nume și o valoare.

Numele variabilei este simbolul cu care ne adresăm variabilei respective, iar valoarea variabilei este conținutul specificat de acel simbol (conținutul obiectului respectiv).

O variabilă se creează cu ajutorul comenzii MAKE sau ca argument de intrare într-o procedură.

Referirea numelui variabilei trebuie precedată de ghilimele, iar referirea conținutului variabilei trebuie precedată de două puncte.

EXEMPLE:

"ALFA — specifică variabila cu numele ALFA;
:ALFA — specifică conținutul variabilei cu numele ALFA.

În limbajul LOGO sînt două tipuri de variabile:

- a) — variabile locale;
- b) — variabile globale.

a) Variabilele locale sînt utilizate ca argumente de intrare în procedură. Ele există numai în cadrul procedurii respective, nu sînt accesibile altor proceduri și dispar din spațiul de lucru după ce procedura respectivă și-a terminat execuția.

b) Variabilele globale sînt create cu comanda MAKE, la nivel de comandă (nu în cadrul procedurilor) și sînt accesibile tuturor procedurilor din cadrul spațiului de lucru.

Comanda MAKE

MAKE nume obiect

Comanda MAKE, atribuie conținutul obiectului specificat variabilei nume.

EXEMPLE:

```
MAKE "DIMENSIUNE 50
PRINT :DIMENSIUNE
50
MAKE "ANIMAL "ELEFANT
PRINT :ANIMAL
ELEFANT
MAKE "ELEV [POPESCU CRISTIAN]
PRINT :ELEV
POPESCU CRISTIAN
```

În exemplele următoare vom scoate în evidență relația dintre numele variabilei și valoarea variabilei:

EXEMPLE:

```
MAKE "CULOARE "ALBASTRU
PRINT "CULOARE
CULOARE
PRINT :CULOARE
ALBASTRU
MAKE "FLOARE "CRIN
PRINT :FLOARE
CRIN
MAKE :FLOARE "ALB
PRINT :CRIN
ALB
```

Conținutul variabilei FLOARE este cuvîntul CRIN. Dacă conținutului variabilei FLOARE îi atribuim cuvîntul ALB, este echivalent cu atribuirea cuvîntului ALB variabilei CRIN.

EXEMPLU:**MAKE :FLOARE obiect****— are același efect cu: MAKE "CRIN obiect**

Operația NAMEP

NAMEP cuvint

Operația NAMEP generează valoarea logică TRUE, dacă cuvântul specificat are atribuită o valoare, adică reprezintă numele variabilei. În caz contrar generează valoarea logică FALSE.

EXEMPLU:

```
PRINT NAMEP "PIONIER
FALSE
MAKE "ELEVI [CRISTIAN ȘI ADRIANA]
PRINT :ELEVI
CRISTIAN ȘI ADRIANA
PRINT NAMEP "ELEVI
TRUE
```

Operația THING

THING nume

Operația THING generează valoarea asociată numelui specificat. Este utilizată pentru a afla ce valoare este asociată valorii variabilei (un fel de adresare indirectă).

THING "CULOARE — este echivalentă cu :CULOARE

Pentru a afla ce valoare este asociată conținutului

:CULOARE — ar trebui scris ::CULOARE — sintaxă care nu este permisă în LOGO. Pentru a rezolva acest impediment se folosește THING :CULOARE

EXEMPLU:

```
MAKE "CULOARE "ALBASTRU
MAKE "ALBASTRU [CULOAREA PREFERATĂ]
PRINT :CULOARE
ALBASTRU
PRINT THING "CULOARE
ALBASTRU
PRINT THING :CULOARE
CULOAREA PREFERATĂ
PRINT THING THING "CULOARE
CULOAREA PREFERATĂ
```

11.7. Primitive LOGO pentru comunicația cu mediul extern

În acest capitol se descriu primitivile prin care utilizatorul poate comunica cu mediul extern. Se prezintă modul în care se pot citi caractere de la tastatură și se poate scrie pe ecranul calculatorului.

11.7.1. Primitive care citeșc informații din exterior

În acest paragraf se prezintă operațiile prin care se verifică dacă s-a tastat un caracter la tastatură, se citește un caracter sau un șir de caractere de la tastatură.

Se descriu operațiile:

KEYP
READCHAR
READLIST

Operația **KEYP**

KEYP

Operația **KEYP** generează valoarea logică **TRUE**, dacă o tastă (sau o combinație de taste) este apăsată și caracterul tastat nu a fost încă preluat printr-o operație **READCHAR** sau **READLIST**. Este utilizată pentru sesizarea apăsării unei taste.

Operația **READCHAR**

READCHAR
RC

Operația **READCHAR**, așteaptă tastarea unui caracter, al cărui cod îl furnizează în momentul tastării acestuia. Caracterul tastat nu este tipărit pe ecran.

EXEMPLU — programul citește de la tastatură și în funcție de caracterul tastat efectuează una din operațiile:

F = deplasează penelul 5 pași înainte;
B = deplasează penelul 5 pași înapoi;
R = rotește penelul cu 30° dreapta;
L = rotește penelul cu 30° stînga.

```
TO DESEN
MAKE "CAR READCHAR
IF "CAR = "F [FD 5]
IF "CAR = "B [BK 5]
IF "CAR = "R [RT 30]
IF "CAR = "L [LT 30]
DESEN
END
DESEN
```

Operația **READLIST**

READLIST
RL

Operația **READLIST** citește o linie de la tastatură (un șir de caractere terminat cu **CR**) și furnizează o listă cu conținutul tastat. Fiecare caracter care se tastează, pe parcursul introducerii liniei, va fi tipărit pe ecran.

EXEMPLU:

```
PRINT COUNT READLIST
ACEASTA ESTE O LISTĂ INTRODUSĂ DE LA TASTATURĂ
8
TO DISCUȚIE
PRINT [CARE ESTE NUMELE TĂU?]
MAKE "NUME READLIST
PRINT [IMI PARE BINE CĂ TE CUNOSC]
PRINT [DOMNULE / ] :NUME
END
DISCUȚIE
CARE ESTE NUMELE TĂU?
IONESCU ȘTEFAN
IMI PARE BINE CĂ TE CUNOSC
DOMNULE IONESCU ȘTEFAN
```

11.7.2. Primitive care afișează pe ecranul calculatorului

Descrierea comenzilor prin care se pot afișa mesaje pe ecranul calculatorului.

Se descriu operațiile:

PRINT
SHOW
TYPE

Comanda PRINT

PRINT obiect
PR obiect
(**PRINT** obiect1 obiect2 ... obiectn)
(**PR** obiect1 obiect2 ... obiectn)

Comanda PRINT afișează pe ecran obiectele specificate după care se trece implicit la rînd nou.

Obiectul specificat poate fi număr, cuvînt sau listă. În cazul în care este listă se afișează numai conținutul respectiv, fără parantezele [] care o delimitează.

EXEMPLU:

```
PRINT 273
273
PRINT "CUVÎNT
CUVÎNT
PRINT [ACEASTA ESTE O LISTĂ]
ACEASTA ESTE O LISTĂ
```

Comanda SHOW

SHOW obiect

Comanda **SHOW** afișează pe ecran obiectul specificat după care se trece implicit la rând nou. Are același efect cu comanda **PRINT** numai că în cazul în care obiectul este listă se afișează și parantezele [] care o delimitează.

EXEMPLU:

```
SHOW 273  
273  
SHOW "CUVINT  
CUVINT  
SHOW [ACEASTA ESTE O LISTĂ]  
[ACEASTA ESTE O LISTĂ]  
SHOW [A SHOW [B C D]  
A  
[B C D]
```

Comanda **TYPE**

TYPE obiect

(TYPE obiect1 obiect2 ... obiectn)

Comanda **TYPE** afișează pe ecran obiectele specificate fără a se trece implicit la rând nou. Dacă obiectul specificat este lista, nu se afișează și parantezele [] care o delimitează. Are același efect cu comanda **PRINT** numai că după afișarea obiectului nu se mai trece implicit la rând nou.

EXEMPLU

```
TYPE 273  
273  
TYPE "CUVINT  
CUVINT  
TYPE [ACEASTA ESTE O LISTĂ]  
ACEASTA ESTE O LISTĂ  
TYPE "A TYPE [B C D]  
AB C D
```

11.7.3. Primitive pentru generare de sunete

În acest paragraf se descrie o primitivă care permite utilizatorului să genereze sunete de durată și înălțime specificate ca intrări.

Comanda **SOUND**

SOUND [durată înălțime]

Comanda **SOUND** generează un sunet de **durată** și **înălțime** specificate ca intrări sub formă de listă. Durata este specificată în secunde, iar înălțimea în semitonuri față de nota C. Durata este un număr natural cuprins între 0 și 255, iar înălțimea este un număr întreg cuprins între -62 și 75.

Pentru specificarea duratei și înălțimii, în funcție de valorile unor variabile :D și :I, trebuie să creăm o listă cu aceste variabile utilizând operația **SENTENCE**.

EXEMPLU

```
SOUND [1 0]
SOUND SENTENCE :X :Y
```

În continuare prezentăm un exemplu care pune în corespondență tastele calculatorului cu sunete.

EXEMPLU:

```
TO SUNETAST
SOUND SENTENCE 0.5 (ASCII READCHAR) — 65
SUNETAST
END
SUNETAST
```

11.8. Primitive LOGO care asigură ramificația în program și care controlează fluxul execuției comenzilor:

Fluxul de control al instrucțiunilor se referă la ordinea în care se execută acestea.

În mod normal instrucțiunile se execută, una după alta, în ordinea în care au fost așezate în program.

Uneori este necesar să se schimbe această ordine implicită.

Pot apare următoarele cazuri:

- se execută o instrucțiune sau alta în funcție de îndeplinirea sau nu a unei condiții;
- repetarea unei liste de instrucțiuni de un număr de ori;
- întreruperea (terminarea) execuției unei proceduri înainte de a ajunge la sfârșitul ei;
- întreruperea temporară (pe o anumită perioadă de timp) a execuției unei proceduri.

În acest paragraf se prezintă primitivile prin care utilizatorul poate schimba fluxul secvențial de execuție a instrucțiunilor. Aceste primitive se împart în trei categorii în funcție de efectul pe care îl au în cadrul programelor, și anume:

- primitive care asigură ramificația în cadrul programului în funcție de anumite condiții;
- primitive care produc întreruperea execuției procedurilor;
- primitive care asigură execuția și repetarea unei liste de instrucțiuni.

11.8.1. Primitive pentru ramificarea condiționată

În acest paragraf se prezintă primitiva care asigură ramificația în program în funcție de o condiție specificată. Noțiunea de predicat se referă la o operație care generează una din valorile logice TRUE sau FALSE. El reprezintă, îndeplinirea sau neîndeplinirea unor condiții.

Primitiva IF

IF predicat lista de instrucțiuni1

IF predicat lista de instrucțiuni1 lista de instrucțiuni2

Primitiva IF, testează valoarea predicatului și:

— dacă valoarea logică, generată de predicat este TRUE se execută lista de instrucțiuni1 și nu se mai execută lista de instrucțiuni2;

— dacă valoarea logică, generată de predicat este FALSE se execută lista de instrucțiuni2 (dacă aceasta există) și nu se mai execută lista de instrucțiuni1.

Dacă în urma execuției uneia din listele de instrucțiuni, se generează un obiect atunci primitiva IF are rol de operație, iar în caz contrar are rol de comandă.

Ex:

```
IF "A=READCHAR [PRINT [ATI TASTAT A]]  
[PRINT [NU ATI TASTAT A]]
```

În acest exemplu primitiva IF este interpretată ca fiind comandă. În funcție de ceea ce se tastează, se recunoaște dacă s-a apăsător pe A sau altă literă.

Ex:

```
PRINT IF "A=READCHAR [ATI TASTAT A] [NU ATI TASTAT  
A]
```

În acest exemplu primitiva IF este interpretată ca fiind operație. În urma execuției, se generează lista: [ATI TASTAT A] sau lista [NU ATI TASTAT A].

Comanda PRINT afișează rezultatul operației IF.

11.8.2. Primitive pentru intreruperea procedurilor

În acest paragraf se descriu primitivele care permit terminarea execuției unei proceduri fără ca aceasta să fi ajuns la sfârșit. De asemenea se prezintă primitivele prin care se realizează intreruperea temporară a execuției unei proceduri.

Se vor descrie următoarele comenzi:

```
BYE WAIT  
OUTPUT TOPLEVEL  
STOP
```

Comanda BYE

BYE

Comanda BYE închide o sesiune de lucru LOGO și transferă controlul în limbajul BASIC.

Dacă se dorește reîntrarea imediată în limbajul LOGO (înainte de a se fi executat ceva în BASIC), pentru a salva diverse proceduri sau programe LOGO, care sînt utile și nu au fost în prealabil salvate, se poate realiza acest lucru prin introducerea comenzii RUN în BASIC.

Reîntoarcerea imediată din BASIC în LOGO, se poate realiza și prin comenzile BASIC:

RANDOMIZE USR 24835 — reîntoarcere care nu modifică spațiul de lucru LOGO.

sau

RANDOMIZE USR 24832 — reîntoarcere care inițializează spațiul de lucru LOGO.

Comanda OUTPUT

OUTPUT obiect

OP obiect

Comanda OUTPUT, generează ca ieșire din procedură, obiectul specificat. Totodată întrerupe execuția procedurii respective (fără a mai continua pînă la sfîrșitul ei, indicat de END) și reîntoarce controlul în procedura sau programul care a apelat-o.

Trebuie subliniat faptul că primitiva OUTPUT este o comandă, iar procedura care o conține devine o operație, în sensul că generează un obiect de ieșire care este utilizat în procedura sau programul care a apelat-o.

O procedură care se termină cu END este interpretată ca fiind comandă, iar cea care se termină prin OUTPUT, este interpretată ca operație.

Comanda OUTPUT are sens numai în cadrul procedurilor. Nu poate fi specificată în cadrul programelor principale.

EXEMPLU:

```
TO POZIȚIE :CAR :LISTA
IF NOT MEMBERP :CAR :LISTA [OUTPUT 0]
IF :CAR = FIRST :LISTA [OUTPUT 1]
OUTPUT 1 +POZIȚIE :CAR BUTFIRST :LISTA
END

MAKE "VOCALE [A E O U]
PRINT POZIȚIE "E :VOCALE
2
PRINT POZIȚIE "B :VOCALE
0
```

Acest exemplu afișează poziția caracterului specificat în lista de intrare, precizată. Dacă caracterul nu exista în listă, se afișează valoarea 0.

EXEMPLU:

```
TO MEDIE :X :Y
(:X + :Y)/2
END

PRINT MEDIE 4 6
— generează mesaj de eroare:
you don't say what to do with 5 in MEDIE
(Nu se spune ce se face cu rezultatul 5 în MEDIE),
```

Comanda STOP

STOP

Comanda STOP are ca efect terminarea (întreruperea execuției) procedurii curente (în care se află primitiva STOP) și reîntoarcerea controlului în procedura sau programul care a apelat-o, (Fig. 11.27).



Fig. 11.27. Efectul comenzii STOP

Această comandă are sens numai în cadrul procedurilor.

Trebuie precizat faptul că o procedură care se termină cu STOP este interpretată ca fiind comandă. Ea nu generează nici o valoare care ar putea fi utilizată în programul chemător.

EXEMPLU:

```
TO NUMĂRĂ :NR
  PRINT :NR
  IF :NR = 0 [PRINT "GATA STOP]
  NUMĂRĂ :NR -1
END
```

NUMĂRĂ 5

4

3

2

1

0

GATA

```
TO EXTRAGE :OBIECT
  IF EMPTY? :OBIECT [STOP]
  PRINT :OBIECT
  EXTRAGE BUTLAST :OBIECT
END
```

```

EXTRAGE -CUVÎNT
CUVÎNT
CUVÎN
CUVI
CUV
CU
C
EXTRAGE [ACEASTA ESTE O LISTĂ]
ACEASTA ESTE O LISTĂ
ACEASTA ESTE O
ACEASTA ESTE
ACEASTA

```

Comanda **WAIT**

WAIT nr

Comanda **WAIT** suspendă temporar execuția primitivelor **LOGO**, pentru o durată de timp egală cu $nr/50$ secunde. Practic introduce o așteptare, proporțională cu numărul „nr“ specificat.

```

EXEMPLU: TO TRASEU
RIGHT 10 * RANDOM 30
FORWARD 10 * RANDOM 10
PRINT POSITION
WAIT 100
TRASEU
END
TRASEU

```

Acest exemplu mută penelul pe ecran, deplasându-l înainte și rotindu-l la dreapta în mod aleator.

După fiecare mișcare, se afișează coordonatele poziției sale pe o durată de timp de $100/50=2$ secunde, după care se reia operația de mutare, respectiv afișare.

Comanda **TOPLEVEL**

TOPLEVEL

Comanda **TOPLEVEL** are ca efect întreruperea execuției procedurii sau programului curent și reîntoarcerea controlului în regim de comandă, (Fig. 11.28).

Este bine să stabilim care este deosebirea între primitivile **TOPLEVEL** și **STOP**.

Primitiva **STOP** întrerupe execuția procedurii curente și reîntoarce controlul în procedura sau programul care a apelat-o, în timp ce **TOPLEVEL** întrerupe execuția întregului program.

În general o comandă **TOPLEVEL** este utilizată în momentul în care apare o eroare în programul în curs de execuție (neavînd sens continuarea sa) și trebuie trecut controlul utilizatorului.

Vom descrie două proceduri care scot în evidență deosebirea între **TOPLEVEL** și **STOP**.

EXEMPLU:



Fig. 11.28. Efectul comenzii TOPLEVEL

```
TO TOPCAUTĂ :ELEM :LISTA
IF EMPTY? :LISTA [STOP]
IF :ELEM = FIRST :LISTA [MAKE "OBIECT :LISTA TOPLEVEL]
TOPCAUTĂ :ELEM BUTFIRST :LISTA
END
```

```
TO STOPCAUTĂ :ELEM :LISTA
IF EMPTY? :LISTA [STOP]
IF :ELEM = FIRST :LISTA [MAKE "OBIECT :LISTA]
STOPCAUTĂ :ELEM BUTFIRST :LISTA
END
```

Ambele exemple atribuie variabilei OBIECT, sublista care începe cu elementul specificat de ELEM.

În primul exemplu, procedura atribuie prima sublistă care începe cu elementul specificat, iar în al doilea exemplu, procedura atribuie ultima sublistă.

EXEMPLU:

```
TOPCAUTA 'I 'CRISTIAN
PRINT :OBIECT
  ISTIAN
STOPCAUTA 'I 'CRISTIAN
PRINT :OBIECT
  IAN
TOPCAUTA 'ELEVII [ELEVII CLASEI A IV-A SÎNT ELEVII CEI
  MAI BUNI]
PRINT :OBIECT
  ELEVII CLASEI A IV-A SÎNT ELEVII CEI MAI BUNI
STOPCAUTA 'ELEVII [ELEVII CLASEI A IV-A SÎNT ELEVII CEI
  MAI BUNI]
PRINT :OBIECT
  ELEVII CEI MAI BUNI
```

11.8.3. Primitive pentru execuția și repetarea unei liste de instrucțiuni

În acest paragraf se descriu primitivele LOGO, care permit execuția și repetarea unei liste de instrucțiuni specificate.

11. PRIMITIVE LOGO

Se vor descrie:

REPEAT
RUN

Comanda REPEAT

REPEAT nr lista__de__instrucțiuni

Comanda REPEAT, repetă lista de instrucțiuni specificată de un număr de ori stabilit de „nr”. Numărul „nr” trebuie să fie pozitiv. În cazul în care „nr” nu este întreg este trunchiat și transformat în întreg.

EXEMPLU:

```
REPEAT 4 [FD 60 RT 90]
REPEAT 3 [FD 40 RT 120]
REPEAT 360 [FD 1 RT 1]
REPEAT 4 [PRINT [ACEASTA ESTE O LISTĂ]]
```

Comanda RUN

RUN lista__de__instrucțiuni

Comanda RUN execută lista de instrucțiuni specificată ca și cum ea ar fi fost introdusă în mod direct.

Vom încerca ca în cele două exemple care urmează să evidențiem când o listă este interpretată ca atare și când elementele sale sînt interpretate ca primitive.

EXEMPLU:

```
TO TEXTE
PRINT READLIST
PRINT
TEXTE
END

TO EXECTEXT
PRINT RUN READLIST
PRINT
EXECTEXT
END
```

În primul exemplu lista citită este afișată pe ecran, în timp ce în al doilea exemplu este executată interpretînd elementele ca primitive LOGO și rezultatul execuției este afișat pe ecran.

EXEMPLU:

```
TEXTE
2+3
2+3
6/2
6/2
42=6*7
42=6*7
REMAINDER 16 5
REMAINDER 16 5
```



```

EXECTEXT
2 +3
5
6/2
3
42 = 6 * 7
TRUE
REMAINDER 16 5
1

```

În exemplul următor, se arată cum o procedură se poate aplica fiecărui element al unei liste.

EXEMPLU:

```

TO REPCOM :COM :LISTA
IF EMPTY :LIST [STOP]
RUN LIST :COM FIRST :LIST
REPCOM :COM BUTFIRST :LIST
END

TO PATRAT :L
REPEAT 4 [FD :L RT 90]
END

```

REPCOM "PĂTRAT [10 20 30 40 50 60 70], va genera un set de șapte pătrate ale căror laturi sînt specificate în lista specificată, (Fig. 11.29).

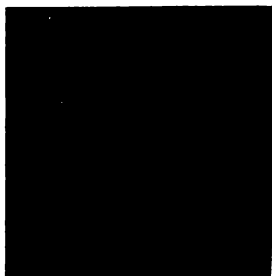


Fig. 11.29. Exemplu de utilizare a comenzii RUN

11.9. Primitive pentru lucrul cu fișiere

În timpul unei sesiuni de lucru LOGO, programul elaborat, procedurile și variabilele utilizate sînt create într-o zonă de memorie numită spațiu de lucru. Informația în cadrul spațiului de lucru este organizată în fișiere.

Fișierele pot fi împărțite în următoarele tipuri:

- fișier LOGO, care conține programe, proceduri și variabile elaborate cu ajutorul limbajului LOGO;
- fișier de editare LOGO, care conține fișierul de lucru al editorului LOGO;
- fișier binar, care conține o zonă de memorie, a cărei informație este interpretată ca date binare sau ca program obiect în limbaj de asamblare.

Fișierele de lucru pot fi salvate pe casetă sau încărcate de pe casetă.

Specificarea numelui fișierului se face utilizînd ghilimelele: "numefișier" — unde numele fișierului poate avea maxim 7 caractere. Pentru a le deosebi este bine ca fiecare nume de fișier să aibă o extensie în funcție de tip:

- .TXT — fișiere text
- .SCR — fișiere ecran
- .BIN — fișiere binare
- .LOG — fișiere LOGO

11. PRIMITIVE LOGO

11.9.1. Primitive pentru încărcare de pe casetă

În acest paragraf se prezintă primitivele LOGO, care permit utilizatorului încărcarea fișierelor rezidente pe caseta magnetică.

Se descriu primitivele:

LOAD
LOADD
LOADSCR

Comanda LOAD

LOAD "numefișier

Comanda LOAD, încarcă de pe casetă fișierul specificat, care în prealabil a fost salvat cu comanda SAVE, în spațiul de lucru LOGO. În timpul încărcării se afișează ce proceduri sînt definite în cadrul fișierului. La terminarea încărcării, se trece în regim de comandă.

EXEMPLU:

**LOAD "FIGURI
PATRAT DEFINED
TRIUNGHI DEFINED**

Comanda LOADD

LOADD "numefișier

Comanda LOADD încarcă de pe casetă fișierul specificat, care în prealabil a fost salvat cu comanda SAVED, în spațiul fișierului de editare al editorului LOGO.

EXEMPLU:

LOAD "FIGURI

Comanda LOADSCR

LOADSCR "numefișier

Comanda LOADSCR încarcă de pe casetă fișierul specificat, care în prealabil a fost salvat cu comanda SAVESCR, în zona de memorie asociată ecranului. Informația încărcată de pe casetă, se afișează pe ecran.

11.9.2. Primitive pentru salvarea pe casetă

În acest paragraf se prezintă primitivele LOGO, care permit utilizatorului să salveze fișiere pe casetă.

Se descriu primitivele:

SAVE

**SAVED
SAVESCR
SAVEALL**

Comanda **SAVE**

SAVE "numefișier [lista de proceduri]

Comanda **SAVE** salvează pe casetă într-un fișier cu numele **numefișier**, procedurile specificate în lista de proceduri. Procedurile specificate trebuie să fie definite în cadrul spațiului de lucru **LOGO**.

EXEMPLU:

SAVE "FIGURI [PĂTRAT TRIUNGHI]

Comanda **SAVEALL**

SAVEALL "numefișier

Comanda **SAVEALL**, salvează pe casetă, într-un fișier cu numele **numefișier**, tot spațiul de lucru **LOGO**. În fișierul respectiv se salvează toate procedurile și toate variabilele globale.

EXEMPLU:

SAVEALL "FIGURI

Comanda **SAVED**

SAVED "numefișier

Comanda **SAVED**, salvează pe casetă, într-un fișier cu numele **numefișier**, fișierul de lucru al editorului **LOGO**.

EXEMPLU:

SAVED "EDITSP

Comanda **SAVESCR**

SAVESCR "numefișier

Comanda **SAVESCR**, salvează pe casetă, într-un fișier cu numele **numefișier**, zona de memorie asociată ecranului.

EXEMPLU:

SAVESCR "ECRAN

11.9.3. Primitive pentru controlul imprimantei și tipărirea informației de pe ecran

În acest paragraf se prezintă primitivele **LOGO**, care permit utilizatorului să tipărească la imprimantă textul programului elaborat, rezultatele obținute sau conținutul ecranului.

11. PRIMITIVE LOGO

Imprimanta trebuie să fie cuplată pe interfața paralelă (care deocamdată nu se livrează în configurația standard). Imprimanta poate fi conectată pe interfața 1 și în acest caz trebuie instalată înainte de încărcarea și lansarea în execuție a limbajului LOGO, cu comenzile din BASIC:

```
FORMAT "t"; 9600  
OPEN #3; "t"
```

S-a considerat că imprimanta serială funcționează cu rata de transfer de 9600 bauds.

Se descriu primitivele:

```
PRINTON  
PRINTOFF  
COPYSCREEN
```

Comanda PRINTON

```
PRINTON
```

Comanda PRINTON, activează imprimanta. După execuția acestei comenzi, tot ce se afișează în regim alfanumeric, se tipărește la imprimantă (cu excepția comenzilor în regim comandă).

```
EXEMPLU:  
PRINTON
```

Comanda PRINTOFF

```
PRINTOFF
```

Comanda PRINTOFF, dezactivează imprimanta. După execuția acestei comenzi, nu se mai tipărește la imprimantă textul.

```
EXEMPLU:  
PRINTOFF
```

Comanda COPYSCREEN

```
COPYSCREEN
```

Comanda COPYSCREEN, copiază ecranul la imprimantă. Această primitivă este utilizată în ambele regimuri de lucru ale ecranului, mod alfanumeric sau mod grafic.

Imprimanta conectată la calculator trebuie să fie imprimanta grafică, altfel comanda nu are nici un sens. Această comandă este executată numai când imprimanta este conectată prin intermediul interfeței paralele.

```
EXEMPLU:  
COPYSCREEN
```

11.9.4. Primitive pentru lucrul cu microdrive-ul

În acest paragraf se prezintă primitivele, care permit utilizatorului să lucreze cu microdriver-ul. Primitivele referitoare la microdrive au aceeași formă și pentru discul flexibil.

Se descriu primitivele:

SETDRIVE
CATALOG
ERASEFILE

Comanda SETDRIVE

SETDRIVE nr

Comanda SETDRIVE, stabilește ce dispozitiv de memorare extern se utilizează. Dacă „nr“ este 0, atunci se consideră caseta (caseta se consideră și în mod implicit), iar dacă „nr“ este un număr între 1 și 8 stabilește ca dispozitiv microdriver-ul (unitatea de disc flexibil) cu numărul specificat.

EXEMPLU:

**SETDRIVE 1
SAVEALL "FIGURI**

Comanda CATALOG

CATALOG

Comanda CATALOG, afișează numele microdriver-ului (discului) și numele tuturor fișierelor rezidente pe el. De asemenea, afișează spațiul liber în kiloocteți.

EXEMPLU:

**SETDRIVE 1
CATALOG**

Comanda ERASEFILE

ERASEFILE "numefișier.extensie

Comanda ERASEFILE, șterge fișierul, al cărui nume este specificat în cadrul comenzii, de pe microdriver-ul (discul) respectiv. Stabilirea microdriver-ului se face printr-o comandă SETDRIVE. Dacă nu se specifică extensia numelui fișierului, atunci ea se consideră implicit .LOG

Alte extensii care trebuie explicitate sînt:

BIN – specifică fișier obiect (binar)

SCR – specifică fișier asociat zonei ecran

TXT – specifică fișier de lucru al editorului LOGO

EXEMPLU:

ERASEFILE "DESEN. SCR

11.10. Primitive pentru gestiunea spațiului de lucru

În acest capitol se prezintă primitivele, care gestionează spațiul de lucru din memoria calculatorului. Aceste primitive permit afișarea procedurilor din spațiul de lucru, ștergerea lor precum și examinarea spațiului de lucru.

11. PRIMITIVE LOGO

11.10.1. Primitive pentru examinarea spațiului de lucru

În acest paragraf se prezintă primitivele, care permit utilizatorului să actualizeze spațiul de lucru, în sensul că poate să obțină informații despre spațiul disponibil și poate compacta spațiul utilizat.

Se descriu primitivele:

NODES
RECYCLE

Operația **NODES**

NODES

Operația **NODES**, generează un număr care specifică câte noduri mai pot fi introduse în spațiul de lucru continuu. Un nod ocupă 5 octeți în memorie. Operația **NODES** furnizează utilizatorului câtă memorie mai este disponibilă spațiului de lucru pentru proceduri, variabile și program.

Informația furnizată de operația **NODES** este mai exactă, dacă este specificată după o comandă **RECYCLE**.

Comanda **RECYCLE**

RECYCLE

Comanda **RECYCLE**, efectuează o compactare a spațiului de lucru, prin rearanjarea procedurilor în memoria calculatorului, astfel încât toate spațiile disponibile, individuale (rămase din ștergerea unor proceduri) să fie adunate într-un spațiu disponibil continuu și unic.

Această compactare, se realizează în mod implicit la umplerea spațiului de lucru. Compactarea durează cca. 1 secundă.

Utilizarea comenzii **RECYCLE**, previne compactarea implicită care ar putea avea loc într-un moment nedorit din punct de vedere al execuției programului.

EXEMPLU:

RECYCLE
NODES

11.10.2 Primitive pentru afișarea conținutului spațiului de lucru

În acest paragraf se prezintă primitivele, care permit utilizatorului, să afișeze informații din spațiul de lucru.

Se descriu primitivele:

PO **POPS**
POALL **POTS**
PONS

În timpul afișării, se poate controla defilarea („scroll”) informației pe ecran, cu ajutorul funcției de control **SS S**.

Comanda PO

PO lista de nume proceduri

Comanda PO, afișează definițiile (conținutul) procedurilor specificate în lista de nume proceduri.

EXEMPLU:

```
PO [PĂTRAT]
TO PĂTRAT :L
REPEAT 4 [FD :L RT 90]
END
```

Comanda POALL

POALL

Comanda POALL, afișează conținutul tuturor procedurilor și valorile variabilelor din spațiul de lucru.

EXEMPLU:

```
POALL
```

Comanda PONS

PONS

Comanda PONS, afișează numele și valorile tuturor variabilelor din spațiul de lucru.

EXEMPLU:

```
PONS
MAKE "F 3
MAKE "LIST [A B C]
```

Comanda POPS

POPS

Comanda POPS, afișează conținutul tuturor procedurilor din spațiul de lucru.

EXEMPLU:

```
POPS
```

Comanda POTS

POTS

Comanda POTS, afișează numele și parametrii de intrare ai tuturor procedurilor din spațiul de lucru.

EXEMPLU:

```
POTS
```

11.10.3. Primitive pentru ștergerea informațiilor din spațiul de lucru

În acest paragraf se prezintă primitivele care șterg informațiile din spațiul de lucru.

Se descriu primitivele:

ERALL ERNS
ERASE ERPS
ERN

Comanda ERALL

ERALL

Comanda ERALL, șterge toate procedurile și variabilele din spațiul de lucru. Are ca efect ștergerea întregului spațiu de lucru. Este echivalent cu a începe o nouă sesiune de lucru LOGO. Înainte de a specifica o astfel de comandă trebuie avut grijă ca în prealabil să se salveze procedurile utile.

Conținutul spațiului de editare nu este afectat de comanda ERALL. Pentru a șterge și spațiul de editare se folosește combinația.

ERALL
EDIT

Comanda ERASE

ERASE lista_de_nume_proceduri
ER lista_de_nume_proceduri

Comanda ERASE, șterge din spațiul de lucru, procedurile specificate în lista_de_nume_proceduri.

Dacă se specifică o singură procedură, atunci ea se poate preciza sub forma de listă sau cuvânt.

EXEMPLU:

ERASE [PĂTRAT TRIUNGHI]
ERASE "PĂTRAT
ERASE [PĂTRAT]

Comanda ERN

ERN lista_de_nume_variabile

Comanda ERN, șterge din spațiul de lucru, variabilele specificate în lista_de_nume_variabile. Dacă se specifică o singură variabilă, atunci ea se poate preciza ca listă sau cuvânt.

EXEMPLU:

ERN "L
ERN [L]
ERN [L A]

Comanda ERNS

ERNS

Comanda ERNS, șterge din spațiul de lucru, toate variabilele care au fost definite pînă în momentul respectiv.

EXEMPLU:

ERNS

Comanda ERPS

ERPS

Comanda ERPS, șterge din spațiul de lucru, toate procedurile.

EXEMPLU:

ERPS

11.11. Primitive pentru definirea și editarea procedurilor

În acest capitol se prezintă modul în care utilizatorul își pregătește programul în limbajul LOGO.

Există două moduri în care se pot defini procedurile:

- a) mod editare în care se utilizează editorul LOGO
- b) mod interactiv în care procedura se definește în regim comandă

În mod editare, orice eroare de introducere poate fi înlăturată imediat cu ajutorul facilităților oferite de editorul LOGO, în timp ce în mod interactiv nu se poate reveni asupra textului introdus.

Cel de-al doilea mod este utilizabil numai în cazul procedurilor simple, de complexitate redusă.

11.11.1. Primitive pentru definire și editare proceduri

În acest paragraf se prezintă primitivele LOGO, care permit utilizatorului să-și definească și editeze proceduri.

Se descriu primitivele:

```
EDIT TO  
ENDS END
```

Comanda EDIT

```
EDIT "numeprocedura  
ED "numeprocedura
```

11. PRIMITIVE LOGO

Comanda EDIT, lansează în execuție editorul LOGO în vederea definirii sau editării procedurii cu numele specificat de **numeprocedură**.

Editarea mai multor proceduri, se realizează prin specificarea în loc de nume procedură o listă cu numele procedurilor respective.

Pentru a defini o procedură nouă, se introduce:

EDIT [] sau
EDIT "numeprocedură"

În acest caz, se lansează editorul LOGO, care va avea spațiul său de lucru vid.

Editarea unei proceduri definite anterior, se realizează prin introducerea EDIT "numeprocedură".

Revenirea la ultima procedură cu care a lucrat editorul, care se află deja în spațiul său de lucru, se face prin introducerea comenzii EDIT

În regim de editare, caracterul de început de rând (?) din regim comandă nu mai apare.

În cadrul editorului, sînt o serie de caractere de control al editării, care se referă la:

- mutarea cursorului pe ecran;
- ștergerea unei zone de text;
- terminarea unei sesiuni de lucru cu editorul.

Prezentarea caracterelor de control, utilizate în procesul de editare:

EMODE — reprezintă modul extins de utilizare a tastaturii, obținut prin apăsarea simultană a tastelor:

CS — litere mari (caps shift)
SS -- caractere simbolice (symbol shift)

Caractere de control al editării:

— caractere de control pentru mutarea cursorului:

CS 5 — mută cursorul un caracter la stînga (←)
CS 6 — mută cursorul o linie mai jos (↓)
CS 7 — mută cursorul o linie mai sus (↑)
CS 8 — mută cursorul un caracter la dreapta (→)

EMODE CS 5 — mută cursorul la începutul liniei curente
EMODE CS 6 — mută cursorul la sfîrșitul ecranului
EMODE CS 7 — mută cursorul la începutul ecranului
EMODE CS 8 — mută cursorul la sfîrșitul liniei curente

EMODE B — mută cursorul la începutul textului de editare
EMODE E — mută cursorul la sfîrșitul textului de editare
EMODE P — mută cursorul la pagina anterioară
EMODE N — mută cursorul la pagina următoare

Observație — În timpul trecerii la pagina anterioară sau următoare de text procesul de defilare („scroll”) poate fi oprit cu SS S pînă la tastarea oricărei taste.

— caractere de control pentru ștergere text:

- CS 0 — șterge un caracter aflat la stînga cursorului;
- EMODE Y — șterge linia curentă (cea pe care se găsește cursorul) începînd cu poziția în care se află cursorul pînă la sfîrșit și o memorează într-o zonă de memorie;
- EMODE R — readuce linia ștearsă și salvată cu EMODE Y, începînd cu poziția curentă a cursorului.

— caractere de control pentru terminarea unei sesiuni de lucru cu editorul.

- EMODE C — se trece din regim editare în regim comandă salvînd toate modificările efectuate.
- CS BREAK — ignoră sesiunea curentă de editare (nu ia în considerare modificările făcute) și trece în regim de comandă.

Comanda EDNS

EDNS lista de nume

Comanda EDNS, permite editarea numelor variabilelor și a valorilor asig-nate acestora. Sînt listate variabilele specificate în lista de nume și valorile asociate acestora, permițînd utilizatorului să editeze numele sau valorile.

Dacă EDNS nu are specificat o listă de nume atunci se listează toate varia-bilele din spațiul de lucru cu valorile asignate și se pot edita în mod corespun-zător.

EXEMPLU:

```
EDNS
MAKE "FLOARE "CRIN
MAKE "DIMENSIUNE 50
MAKE "ELEV [POPESCU CRISTIAN]
```

Cu caracterele de control al editării se pot modifica numele sau valorile variabilelor. După modificare se revine în regim comandă cu EMODE C.

Comanda TO

TO nume_procedură intr1 intr2... intrn

Comanda TO, permite definirea unei proceduri în regim comandă. După introducerea comenzii de definire TO urmată de numele procedurii și de para-metrii de intrare în procedură (intr1 intr2 ... intrn), pe ecran va apare un nou caracter de început rînd (>), care ne invită să introducem definirea pro-cedurii.

Comenzile introduse nu sînt executate, ci vor constitui corpul procedurii respective.

Terminarea definirii procedurii, se realizează cu comanda END.

EXEMPLU:

```
? TO PĂTRAT :L
> REPEAT 4 [FORWARD :L RIGHT 90]
> END
PĂTRAT Defined
?
```

Abandonarea definirii unei proceduri, în timpul introducerii acesteia, se realizează prin tastarea CS BREAK simultan. Pentru a schimba conținutul unei proceduri cu ajutorul comenzii TO trebuie ca aceasta să fie ștersă în prealabil cu comanda ERASE.

Editarea conținutului unei proceduri este de preferat să se facă cu EDIT.

Comanda END

END

Comanda END are un rol special și anume acela de a specifica sfârșitul unei proceduri. Ea trebuie pusă în ultima linie de definire a procedurii.

11.11.2. Primitive pentru modificarea procedurilor sub controlul programului

În acest paragraf se prezintă primitivele care permit definirea procedurilor sub controlul programului.

Se descriu primitivele:

**COPYDEF PRIMITIVEP
DEFINE TEXT
DEFINEP**

Comanda COPYDEF

COPYDEF numenou nume

Comanda COPYDEF copiază corpul de definire (conținutul) procedurii cu numele **nume** (care a fost definită anterior), într-o nouă procedură, cu numele **numenou**. Procedura existentă inițial nu își pierde conținutul.

Ex:

COPYDEF "PT "PĂTRAT

Copiază definiția procedurii PĂTRAT, sub numele PT. În urma execuției comenzii, vom avea două proceduri, una cu numele PT și alta cu numele PĂTRAT, care au același conținut.

Comanda DEFINE

DEFINE nume lista

Comanda DEFINE, permite generarea prin program a corpului unei proceduri. **Nume** reprezintă numele procedurii astfel create, iar **lista** specifică parametrii de intrare în procedură și liniile de definire a acesteia. Dacă procedura nu are parametri de intrare, trebuie ca lista să înceapă cu o sublistă vidă.

**EXEMPLU: DEFINE "DREPTUNGHI [[:L :I] [FD :I RT 90] [FD :L RT 90]
[FD :I RT 90] [FD :L RT 90]]**

PO "DREPTUNGHI

```

TO DREPTUNGHI :L :I
FD :I RT 90
FD :L RT 90
FD I RT 90
FD :L RT 90
END

```

```

DEFINE "SALUT [] PRINT [BUNĂ ZIUA ]
PO "SALUT
TO SALUT
PRINT [BUNĂ ZIUA]
END

```

Operația DEFINEP

DEFINEP cuvînt

Operația DEFINEP, generează valoarea logică TRUE, dacă parametrul **cuvînt** este numele unei proceduri definite.

În caz contrar, generează valoarea logică FALSE.

EXEMPLU:

```

PRINT DEFINEP "PĂTRAT
TRUE

```

Operația PRIMITIVEP

PRIMITIVEP nume

Operația PRIMITIVEP generează valoarea logică TRUE, dacă parametrul **nume** reprezintă o primitivă LOGO.

În caz contrar, generează valoarea logică FALSE.

EXEMPLU:

```

PRINT PRIMITIVEP "FORWARD
TRUE
PRINT PRIMITIVEP "PĂTRAT
FALSE

```

Operația TEXT

TEXT nume

Operația TEXT furnizează definiția procedurii, cu numele **nume** ca o listă de subliste, care poate fi utilizată ca intrare într-o comandă DEFINE.

Primul element din listă reprezintă intrările în procedură. Celelalte subliste reprezintă, fiecare, câte o linie din definiția procedurii.

EXEMPLU:

```

SHOW TEXT "DREPTUNGHI
[[:L :I] [FD :I RT 90 ] [FD :L RT 90] [FD :I RT 90]
[FD :L RT 90]]

```

11. PRIMITIVE LOGO

11.12. Primitive pentru funcții diverse

În acest capitol se prezintă primitivele care permit accesul utilizatorului la memoria calculatorului în mod direct. Utilizatorul trebuie să cunoască bine alocarea spațiului de memorie, deoarece aceste primitive adresează memoria privită ca resursă a calculatorului și orice modificare a conținutului său poate distruge conținutul spațiului de lucru sau conținutul variabilelor de sistem.

În general aceste primitive încep cu punct (.).

11.12.1. Primitive pentru accesul la memoria calculatorului

Aceste primitive permit utilizatorului să aibă acces în mod direct la memoria calculatorului.

Se descriu primitivele:

.BLOAD	.EXAMINE
.BSAVE	.RESERVE
.CALL	.RESERVED
.DEPOSIT	

Comanda **.BLOAD**

.BLOAD "numefișier adr

Comanda **.BLOAD** încarcă de pe casetă sau microdrive (disc) în memoria calculatorului conținutul fișierului specificat începînd cu adresa „adr”. Fișierul de pe casetă poate conține un program obiect direct executabil (de către unitatea de comandă a calculatorului) sau o zonă de date binare.

Trebuie avut grijă ca adresa „adr” să nu se suprapună peste spațiul de lucru LOGO, sau peste zona variabilelor de sistem.

EXEMPLU:

.BLOAD "FOBIECT 64824

Conținutul fișierului obiect FOBIECT se citește de pe casetă și se depune în memorie începînd cu adresa 64824.

Comanda **.BSAVE**

.BSAVE *numefișier [startadr lng]

Comanda **.BSAVE** salvează pe casetă sub numele **numefișier**, conținutul unei zone de memorie delimitată de adresa de început **startadr** și lungimea **lng**.

EXEMPLU:

.BSAVE "FOBIECT [64824 200]

În exemplul de mai sus se salvează pe casetă zona de memorie 64824 ... 65024 într-un fișier care are numele FOBIECT.

Comanda **.CALL**

.CALL adr

Comanda **.CALL** lansează în execuție un program obiect, care se află în memoria calculatorului, începînd cu adresa „adr”. Programul obiect trebuie încărcat în prealabil cu o comandă **.BLOAD**

EXEMPLU:

.CALL 64824

Comanda **.DEPOSIT**

.DEPOSIT adr val

Comanda **.DEPOSIT** depune în celula de memorie, a cărei adresă este specificată de „adr”, valoarea „val”.

EXEMPLU:

.DEPOSIT 65010 55

Procedura care definește un caracter special (1/4), care poate fi ulterior utilizat în cadrul programelor LOGO, se realizează astfel:

EXEMPLU:

```
TO USR.A
.DEPOSIT 65368 64
.DEPOSIT 65369 68
.DEPOSIT 65370 72
.DEPOSIT 65371 80
.DEPOSIT 65372 42
.DEPOSIT 65373 74
.DEPOSIT 65374 15
.DEPOSIT 65375 2
PRINT CHAR 144
END
USR.A
1/4
```

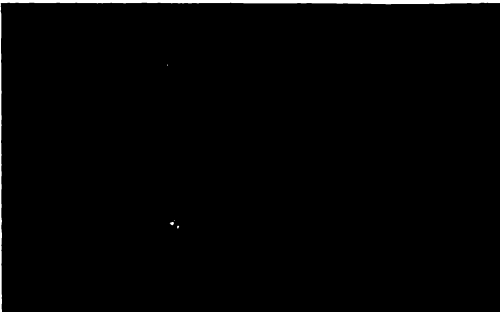


Fig. 11.30. Exemplu de utilizare a comenzii **.DEPOSIT**

Un caracter este definit printr-o matrice de puncte. Avînd adresa de început a matricei respective, se poate modifica în mod corespunzător forma caracterului în funcție de ceea ce se depune în acea zonă. Valorile depuse trebuie interpretate ca valori binare. Biții egali cu 1 se consideră puncte aprinse, iar biții egali cu 0 se consideră puncte stinse.

În exemplul precedent s-a obținut desenul din Fig. 11.30.

Adresă	Conținut		
	Zecim.	Hexazecim.	Binar
65368	64	= 40	= 01000000
65369	68	= 44	= 01000100
65370	72	= 48	= 01001000
65371	80	= 50	= 01010000
65372	42	= 2A	= 00101010
66373	74	= 4A	= 01001010
65374	15	= 0F	= 00001111
65375	2	= 02	= 00000010

Tabelul 11.2 Conținutul zonei de memorie 65368—65375 ca urmare a comenzii .DEPOSIT

Operația .EXAMINE

EXAMINE adr

Operația .EXAMINE generează o valoare ce reprezintă conținutul celei de memorie specificată de adresa **adr**.

EXEMPLU:

```
PRINT .EXAMINE 65370
72
```

Comanda .RESERVE

.RESERVE nr

Comanda .RESERVE rezervă o zonă de memorie de lungime „nr“, la sfârșitul spațiului de lucru LOGO. Această zonă devine neutilizabilă sub controlul limbajului LOGO (cu excepția comenzilor .BLOAD, .DEPOSIT), fiind rezervată pentru o eventuală încărcare a unor programe obiect, sau a unor date binare.

Rezervarea unei zone de memorie cu comanda .RESERVE se poate face numai la începutul unei sesiuni de lucru LOGO.

Să presupunem că dorim să rezervăm o zonă de 200 de octeți pentru a încerca un program obiect.

La începutul unei sesiuni de lucru LOGO se dă comanda .RESERVE 200.

Structura memoriei calculatorului este prezentată în Fig. 11.31.

Fig. 11.31. Exemplu de utilizare a comenzii .REVERSE

Operația .RESERVED

.RESERVED

Operația **.RESERVED** generează adresa de început și adresa de sfârșit a zonei de memorie care în prealabil a fost rezervată cu o comandă **.RESERVE**.

EXEMPLU:

PRINT .RESERVED
64824 65024

11.12.2. Primitive pentru lucrul cu interfața serială

În acest paragraf se prezintă primitivile care permit utilizatorului să lucreze cu interfața serială RS232.

La interfața serială se pot conecta echipamente periferice care lucrează serial asincron conform standardului CCITT V. 24. Conexiunea între calculator și echipament se face pe linie serială. Informația paralelă (8 biți) se trimite/recepționează pe câte o singură linie de date (una pentru recepție alta pentru transmisie).

Structura logică a informației de date este prezentată în figura 11.32.



Fig. 11.32. Structura logică a informației seriale

Durata „t”, cit un bit este ținut pe linia de transmisie, este funcție de rata de transfer a datelor. Rata de transfer este standardizată și poate fi: 50, 110, 150, 300, 600, 1200, 2400, 4800, 9600, 19200 bauds.

(Baud = unitate de măsură a ratei de transfer, este echivalent într-o oarecare măsură, cu numărul de biți pe secundă care se transmit).

Se descriu primitivile:

.SETSERIAL
.SERIALIN
.SERIALOUT

Comanda .SETSERIAL

.SETSERIAL ratatransf

Comanda **.SETSERIAL** stabilește rata de transfer (viteza de transfer) a interfeței seriale la valoarea **ratatransf**.

Rata de transfer poate fi una din valorile: 50, 110, 150, 300, 600, 1200, 2400, 4800, 9600, 19200.

La inițierea unei sesiuni de lucru LOGO, rata de transfer se stabilește în mod implicit la valoarea 9600.

Ex:

.SETSERIAL 1200

Stabilește rata de transfer la 1200 bauds.

Operația **.SERIALIN**

.SERIALIN

Operația **.SERIALIN** furnizează o valoare pe 8 biți (între 0 și 255) ce reprezintă informația recepționată de interfața serială. Operația **.SERIALIN** așteaptă pînă cînd echipamentul serial cu care este conectat calculatorul transmite un caracter (un cuvînt pe 8 biți).

EXEMPLU:

PRINT .SERIALIN
65

Comanda **.SERIALOUT**

.SERIALOUT data

Comanda **.SERIALOUT** transmite, prin intermediul interfeței seriale, valoarea dată (care trebuie să fie între 0 și 255) către echipamentul serial care este conectat cu calculatorul.

EXEMPLU:

.SERIALOUT 65

11.12.3. Primitive pentru specificarea obiectelor și primitivelor LOGO

În acest paragraf se prezintă comenzile LOGO care permit utilizatorului să afișeze toate primitivile recunoscute de limbajul LOGO sau să afișeze ce primitive și proceduri sînt implementate la un moment dat în spațiul de lucru.

Se descriu primitivile:

.CONTENTS
.PRIMITIVES

Operația **.CONTENTS**

.CONTENTS

Operația **.CONTENTS** generează o listă cu tot ce este implementat în spațiul de lucru LOGO. Lista cuprinde procedurile utilizatorului, variabilele etc. Trebuie avut în vedere că operația **.CONTENTS** poate utiliza un spațiu de memorie considerabil de mare.

Comanda **.PRIMITIVES**

.PRIMITIVES

Comanda **.PRIMITIVES** afișează toate primitivile implicite, recunoscute de limbajul LOGO.

EXEMPLU:

.PRIMITIVES

12.1. Proiectarea programelor

Pentru scrierea programelor în limbajul LOGO se utilizează primitivele prezentate în capitolul 11. Operațiile care conduc la rezolvarea unei probleme sînt descrise în general de un algoritm. Algoritmul constă dintr-o mulțime de operații executate într-o ordine bine stabilită, asupra unui set de valori denumite date de intrare și care produc, în timp finit, un set de valori denumite date de ieșire.

Elaborarea unui program presupune atît realizarea algoritmului cît și implementarea acestuia într-un limbaj de programare. Aceste operații pot fi structurate pe două nivele:

- nivelul logic;
- nivelul fizic.

Nivelul logic constă din trei etape:

- etapa de definire;
- etapa de descriere;
- etapa de evaluare.

Nivelul fizic constă din două etape:

- etapa de implementare;
- etapa de testare.

Etapa de definire constă dintr-un set de operații prin care se specifică:

- lista variabilelor de intrare și ieșire ale programului;
- tipul și organizarea datelor de intrare și ieșire;
- reprezentarea internă și externă a datelor.

Etapa de descriere constă în specificarea algoritmului care prelucrează mulțimea valorilor de intrare. Algoritmul poate fi descris în cuvinte din limbajul natural, în pseudocod sau prin organigrame. Se definesc astfel o serie de proceduri de bază, specificate prin:

- setul de valori de intrare;
- setul de valori de ieșire;
- operațiile asupra datelor.

Etapa de evaluare constă în verificarea faptului că algoritmul elaborat în etapa de descriere realizează toate cerințele enunțului problemei și că repre-

zentarea datelor este corect aleasă. Dacă în urma evaluării rezultatele nu sînt satisfăcătoare se reiau etapele de definire și descriere pînă cînd se realizează toate cerințele problemei.

Aceste etape spunem că reprezintă nivelul logic deoarece nu depind decît în mică măsură de limbajul concret în care se va face implementarea.

Nivelul fizic constă în implementarea efectivă a algoritmului într-un limbaj de programare adecvat.

Etapa de implementare constă în transpunerea în limbajul de programare ales, în cazul nostru LOGO, a algoritmului prezentat în etapa de descriere.

În *etapa de testare* se validează buna funcționare a programului. În această etapă se elimină erorile sintactice (nerespectarea regulilor de scriere specifice limbajului ales) și a celor semantice generate de un algoritm incorect sau o transcriere incorectă a algoritmului în limbajul respectiv.

Se reia faza de implementare pînă se obțin rezultatele dorite.

Respectarea acestor etape creează premisele unei proiectări sistematice a programelor. Limbajul LOGO oferă facilități de elaborare sistematică a programelor pornind de la general la particular (top-down) prin rafinări succesive pînă la obținerea unei versiuni corecte a programului.

EXEMPLU:

În continuare se va exemplifica această tehnică de abordare de la general la particular prin descompunerea unei probleme în subprobleme în cazul unui program de grafică. Se cere să se deseneze un robot cu specificațiile din fig. 12.1.

În primul rînd descompunem robotul în elementele componente: CAP, GIT, CORP, MIINI, PICIOARE.

La rîndul lor unele din aceste elemente pot fi descompuse în elemente mai simple. Astfel CAP-ul poate fi descompus în:

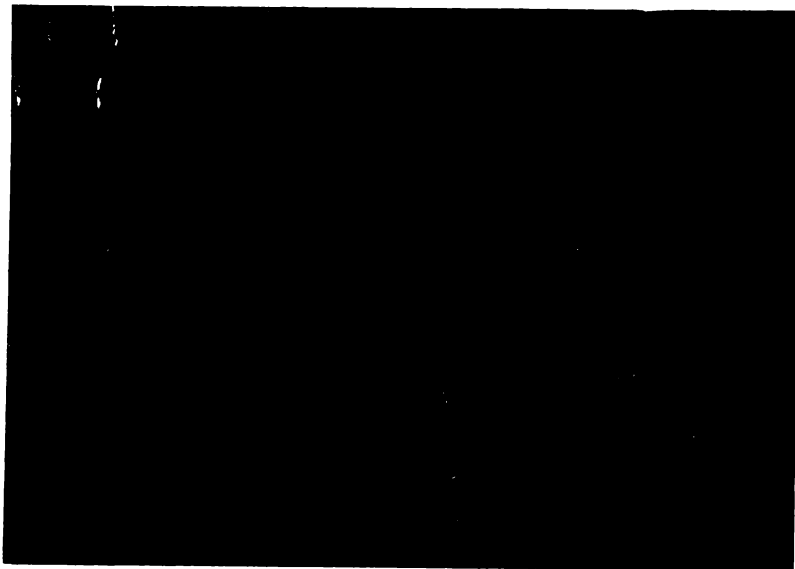


Fig. 12.1. Robot

— formă cap FORMACAP;
 — ochi stînga OCHIST;
 — ochi dreapta OCHIDR;
 — nas NAS;
 — gura GURA.

De fapt se observă că robotul are la bază două elemente simple: dreptunghiul și pătratul.

Astfel pentru a construi robotul este necesar să se definească dreptunghiul și pătratul cu laturi variabile și coordonatele unui vîrf. Plecînd de la general la particular, desenarea robotului se va realiza cu următoarea succesiune de proceduri:

TO ROBOT
 CAP
 GÎT
 CORP
 MÎINI
 PICIOARE
 END

TO CAP
 FORMACAP
 OCHIST
 OCHIDR
 NAS
 GURĂ
 END

TO MÎINI
 MINAST
 MINADR
 END

TO PICIOARE
 PICIORST
 PICIODR
 END

TO PĂTRAT :X :Y :L
 PU
 SETPOS SENTENCE :X :Y
 PD
 REPEAT 4 [FD :L RT 90]
 END

TO DREPTUNGHI :X :Y :I :L
 PU
 SETPOS SENTENCE :X :Y
 PD
 REPEAT 2 [FD :I RT 90 FD :L
 RT 90]
 END

TO FORMACAP
 DREPTUNGHI — 30 20 50 60
 END

TO OCHIST
 PĂTRAT —20 50 10
 END

TO OCHIDR
 PĂTRAT 10 50 10
 END

TO NAS
 PĂTRAT —5 40 10
 END

TO GURA
 DREPTUNGHI —15 30 5 30
 END

TO GÎT
 DREPTUNGHI —10 10 —10 20
 END

TO CORP
 DREPTUNGHI —30 —40 50 60
 END

TO MINAST
 DREPTUNGHI —60 —10 20 30
 END

TO MINADR
 DREPTUNGHI 30 —10 20 30
 END

TO PICIORST
 DREPTUNGHI —30 —80 40 20
 END

TO PICIODR
 DREPTUNGHI 10 —80 40 20
 END

EXEMPLU:

Un alt exemplu care ilustrează această tehnică constă în desenarea unui peisaj similar cu cel din fig. 12.2

TO PEISAJ
 CASA
 BRAZI
 END

TO CASA
 PEREȚI
 GEAMURI
 UȘA
 ACOPERIȘ
 END

TO PEREȚI
 DREPTUNGHI —20 —60 70 120
 END

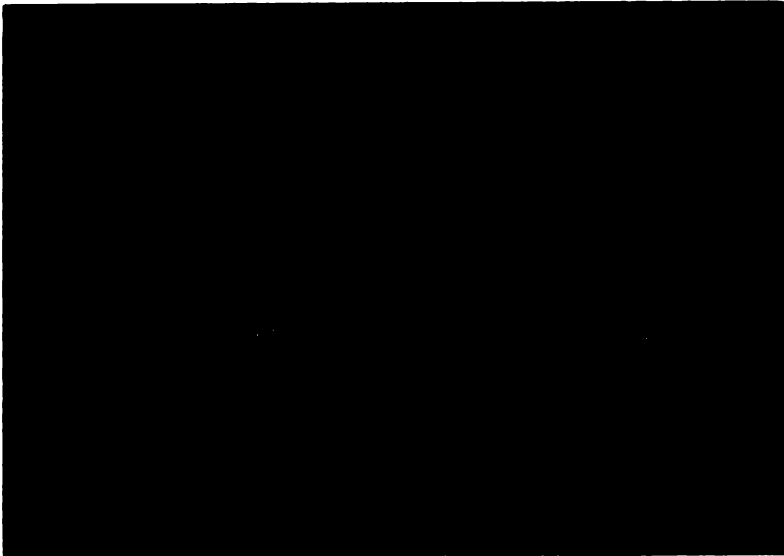


Fig. 12.2. Pelsaj

```
TO GEAMURI
  DREPTUNGHI 0 -20 20 30
  DREPTUNGHI 50 -20 20 30
END
```

```
TO UȘA
  DREPTUNGHI 30 -60 30 20
END
```

```
TO ACOPERIȘ
  PU
  SETPOS [-20 10]
  PD
  RT 45
  FD 30 * 1.41
  RT 45
  FD 60
  RT 45
  FD 30 * 1.41
END
```

```
TO BRAD :X :Y :L
  PU
  SETPOS SENTENCE :X :Y
  PD
  IF :L<2 [STOP]
  FD :L
  RT 120
  FD :L
  BK :L
  LT 60
  BK :L
  FD :L
  LT 60
  BRAD :X :Y +:L :L * 0.6
END
```

```
TO BRAZI
  BRAD -100 -60 20
  BRAD -60 -60 15
  BRAD -30 -50 10
END
```

EXEMPLU:

În exemplele prezentate la începutul acestui capitol s-a arătat tehnica de dezvoltare a unui program prin descompunerea în subprograme. Deoarece algoritmul de rezolvare era foarte simplu în ambele cazuri s-au obținut rezultatele finale dorite cumulând rezultatele subprogramelor componente. În cazul unor probleme care necesită algoritmi mai complicați, în special algoritmi euristici, se poate aborda o altă cale de la general la particular sau de la complex la simplu prin simplificarea inițială a programelor și rafinarea succesivă a acestora. Această tehnică va fi exemplificată prin scrierea unui program pentru jocul Nim. Acesta constă din a extrage una, două sau trei bețe dintr-o mulțime dată. Doi jucători execută mutări (extrageri) succesive. Jucătorul care extrage ultimul băț, adică face ultima mutare acela câștigă jocul. Inițial se va scrie o procedură simplă care va fi analizată și perfecționată pentru a obține o serie de programe din ce în ce mai „inteligente“.

Astfel primul program nu implementează de fapt o strategie de joc ci pur și simplu ține scorul unui joc între două persoane, fără a analiza nici măcar corectitudinea mutărilor acestora. În etapa următoare programul va analiza corectitudinea mutărilor și va anunța câștigătorul. În continuare programul va fi dezvoltat astfel încât să poată el însuși juca cu programatorul efectuând mutări aleatoare dar corecte (nu neapărat pe cele mai bune). După ce acest program se execută corect el poate fi rafinat în continuare îmbunătățindu-i strategia, sau se poate scrie un program complet nou pe baza experienței acumulate, până se obține un program „intelligent“ cu o strategie infailibilă.

Pentru a trece la scrierea programului se vor face următoarele notații:

— JOCNIM numele procedurii care implementează jocul Nim;
 — :BEȚE numărul de bețe rămase neextrase;
 — :JUCĂTOR numele unui jucător;
 — :OPONENT numele partenerului de joc;
 — :MUTARE numărul de bețe ce se vor extrage în această rundă.

JOCNIM trebuie să țină minte :BEȚE, :JUCĂTOR și :OPONENT de la runda precedentă și să asigure preluarea prin :MUTARE a numărului de bețe ce vor fi extrase. Primele trei variabile trebuie transmise de la o rundă la alta deci trebuie să fie intrări. Pe de altă parte :MUTARE este preluată de la tastatură deci nu trebuie să fie intrare. Când algoritmul va fi suficient de elaborat ca să execute el însuși mutări, :MUTARE o să fie generată de o procedură. Preluarea unei mutări de la tastatură va fi realizată cu procedura PRELMUTARE. Iată prima versiune a jocului logice JOCNIM:

```
TO JOCNIM :BEȚE :JUCĂTOR :OPONENT
  PRINT SENTENCE [NUMĂRUL DE BEȚE ESTE] :BEȚE
  PRINT SENTENCE :JUCĂTOR [CARE ESTE MUTAREA TA?]
  MAKE "BETEN :BEȚE — PRELMUTARE
  JOCNIM :BETEN :OPONENT :JUCĂTOR
END
```

```
TO CITNUMĂR
  MAKE "IN FIRST READLIST
  IF NUMBERP :IN [OUTPUT :IN] [PRINT [VA ROG INTRODUCEȚI UN
  NUMĂR] OUTPUT CITNUMĂR]
END
```

```
TO PRELMUTARE
  MAKE "MUTARE CITNUMĂR
  OUTPUT :MUTARE
END
```

Procedura PRELMUTARE putea fi mai simplă în această etapă, de exemplu fără să testeze dacă răspunsul jucătorului este număr sau altceva. În continuare programul va fi dezvoltat cu următoarele funcții:

- detectarea sfârșitului jocului;
- declanșarea câștigătorului la sfârșitul jocului;
- verificarea mutărilor jucătorilor dacă sînt 1,2 sau 3 de fiecare dată.

Primele două funcții sînt implementate astfel:

```
IF :BETEN = 0 [PRINT SENTENCE :JUCĂTOR [ESTE CÎȘTIGĂTOR!] STOP]
```

A treia funcție se poate implementa modificînd pe PRELMUTARE astfel ca să efectueze o reinierare dacă este cazul. Se va utiliza predicatul MEMBERP care primește la intrare un element și o listă și testează dacă elementul aparține listei.

```
TO PRELMUTARE
  PRINT [PUTEȚI EXTRAGE 1, 2 SAU 3 BEȚE]
  MAKE "MUTARE CITNUMĂR
  IF MEMBER :MUTARE [1 2 3] [ ] [OUTPUT PRELMUTARE]
  OUTPUT :MUTARE
END
```

Deși programul este mult mai perfecționat acum mai are însă multe aspecte nerezolvate. Astfel atunci când :BEȚE este 2, PRELMUTARE permite să se extragă 1, 2 sau 3 bețe. Dacă JUCĂTOR extrage 3 bețe, :BEȚE devine negativ și jocul nu se mai termină niciodată. Rămâne ca exercițiu eliminarea acestei pene. În această formă programul poate fi completat cu o serie de funcții noi cum ar fi mutări contra cronometru, declararea câștigătorului cu una sau două mutări în avans, revenirea la mutarea precedentă, afișarea permanentă a punctajului, îndrumarea jucătorilor privind regulile de joc, schimbarea regulilor de joc, etc.

În continuare se va scrie programul JOCNIM astfel ca să poată efectua propriile sale mutări. Cea mai simplă soluție pentru efectuarea unei mutări este o alegere aleatoare cu procedura GENMUTARE. Vom obține astfel:

```
TO PRELMUTARE :JUCĂTOR
  IF :JUCĂTOR = [CALCULATOR] [MAKE "MUTARE GENMUTARE] [PRINT
    ]PUTEȚI EXTRAGE 1, 2 SAU 3 BEȚE] MAKE "MUTARE CITNUMĂR]
    .....
```

```
TO GENMUTARE
  OUTPUT ITEM (1 +RANDOM 3) [1 2 3]
END
```

În această etapă pana dinainte poate avea efecte foarte serioase. O soluție de eliminare a acestei pene constă în furnizarea numărului de bețe la intrarea procedurii PRELMUTARE:

```
TO PRELMUTARE :JUCĂTOR :BEȚE
; iar după liniile MAKE se introduce textul
  IF :MUTARE > :BEȚE [OUTPUT PRELMUTARE :JUCĂTOR :BEȚE ]
```

Exersînd programul în această etapă se pot constata următoarele greșeli mari pe care le poate face în timpul jocului:

- mai sînt 5 bețe și calculatorul extrage două;
 - mai sînt șase sau șapte bețe și calculatorul nu lasă patru;
 - mai sînt două sau trei bețe și calculatorul nu le extrage pe toate!
- Pentru a elimina aceste greșeli posibile se va proceda astfel:
- se va testa dacă mai există una, două sau trei bețe:


```
IF MEMBERP :BEȚE [1 2 3]...
```
 - dacă testul este adevărat extrage toate bețele:


```
OUTPUT :BEȚE
```
 - dacă testul nu este satisfăcător procedează ca mai înainte


```
MAKE "MUTARE GENMUTARE
```

Se obține astfel procedura:

```
TO EXECUTARE :BEȚE
  IF MEMBERP :BEȚE [1 2 3] [OUTPUT :BEȚE] [OUTPUT GENMUTARE]
END
```

Această procedură se va utiliza în locul procedurii GENMUTARE din PRELMUTARE se mai poate adăuga:

```
IF :BEȚE = 5 [OUTPUT 1]
```

Pentru a crea un program inteligent, sub forma sa finală, se va proceda astfel:

- jocul se termină cînd un jucător lasă zero bețe după ce a efectuat mutarea sa, deci numărul de bețe rămase după mutare are un rol determinant în stabilirea strategiei;
- pentru a ne asigura că la următoarea mutare vor rămîne zero bețe la mutarea curentă trebuie să lăsăm patru;
- pentru ca la mutarea curentă să lăsăm patru la mutarea precedentă trebuia să lăsăm opt;
- deci pentru a putea să câștigăm trebuie să lăsăm în urma unei mutări 0, 4, 8, 12, 16... bețe, adică un număr divizibil cu 4;

REMAINDER :NUMĂR 4 să fie 0;
 — mutarea va fi dată astfel de expresia:
 :NUMĂR — REMAINDER :NUMĂR 4

Astfel modificând EXECUTARE se obține MUTAREINT adică mutare inteligentă cu o strategie invincibilă.

```
TO MUTAREINT :BEȚE
  MAKE "REM REMAINDER :BEȚE 4
  IF :REM = 0 [OUTPUT 1]
  OUTPUT :REM
END
```

Listingul final al programului este:

```
TO JOCNIM :bețe :jucător oponent
  PRINT SENTENCE [numărul de bețe este] :bețe
  IF NOT (:jucător = [calculator]) [PRINT SENTENCE :jucător [care este mutarea ta ]]
  MAKE "beten :bețe — (PRELMUTARE: jucător :bețe)
  IF :beten = 0 [print sentence :jucător [este câștigător !] STOP]
  JOCNIM :beten: oponent :jucător
END
```

```
TO PRELMUTARE: jucător :bețe
  IF :jucător = [calculator] [make "mutare mutareint] [print [puteți extrage 1, 2 sau 3 bețe MAKE "mutare CITNUMĂR]
  IF NOT MEMBERP :mutare [1 2 3] [output prelmutare :jucător :bețe]
  IF :mutare >: bețe [output prelmutare :jucător :bețe]
  OUTPUT :mutare
END
```

```
TO MUTAREINT
  MAKE "rem REMAINDER :bețe 4
  IF :rem = 0 [output 1]
  OUTPUT :rem
END
```

```
TO CITNUMĂR
  MAKE "in FIRST READLIST
  IF NUMBERP :in [output :in] [print [va rog să introduceți un număr] OUTPUT CITNUMĂR]
  END
```

```
JOCNIM 17 [calculator] [Gelu]
numărul de bețe este 17
numărul de bețe este 16
Gelu care este mutarea ta?
puteți extrage 1, 2 sau 3 bețe
1
numărul de bețe este 15
numărul de bețe este 12
Gelu care este mutarea ta?
puteți extrage 1, 2 sau 3 bețe
6
```

puteți extrage 1, 2 sau 3 bețe
2
numărul de bețe este 10
numărul de bețe este 8
Gelu care este mutarea ta?
puteți extrage 1, 2 sau 3 bețe
2
vă rog să introduceți un număr
1
numărul de bețe este 7
numărul de bețe este 4
Gelu care este mutarea ta?
puteți extrage 1, 2 sau 3 bețe
2
numărul de bețe este 2
calculator este câștigător
?

În exemplele prezentate s-au utilizat proceduri recursive.

Pentru a înțelege bine exemplele trebuie parcurs cu atenție paragraful referitor la recursivitate.

12.2. Recursivitatea în LOGO

Recursivitatea constituie o modalitate de a scrie proceduri care se apelează pe ele înseși.

Pentru a înțelege mai bine procesul de recursivitate să reamintim ce se întâmplă în momentul în care o procedură apelează o altă procedură, (figura 12.3):



Fig. 12.3. Apelul unei proceduri

Trebuie reamintit că fiecare procedură are un set de date proprii (private) care în general nu sînt asociate ei ci apelului procedurii respective.

În momentul execuției procedurii A, se ajunge la apelul procedurii B, moment în care au loc următoarele operații, (figura 12.4).

— suspendarea temporară a execuției procedurii A și salvarea contextului (valorile variabilelor);

— activarea setului de date proprii procedurii B, asociat apelului respectiv.



Fig. 12.4. Operațiile de bază într-un apel de procedură

Se continuă cu execuția procedurii B și în momentul terminării acesteia se desfășoară următoarele operații:

— se reface contextul procedurii A (context care a fost salvat în momentul trecerii la procedura B);

— revenirea în procedura A la prima instrucțiune de după apelul procedurii B.

Salvarea și refacerea contextului se face în mod implicit, de către limbajul LOGO, fără specificarea explicită de către utilizator.

Limbajul LOGO, față de alte limbaje de programare cum ar fi BASIC, FORTRAN etc., asigură mecanismul ca o procedură să se apeleze pe ea însăși, altfel spus are implementat mecanismul de recursivitate.

Având în vedere că o procedură se poate chema pe ea însăși, prin apel recursiv, rezultă faptul că setul de date proprii nu trebuie să fie asociat producerii ci apelului procedurii respective.

La fiecare apel al procedurii respective, se activează un set de date proprii aceluși apel. Rutina care se cheamă pe ea însăși, coexistă cu apelurile anterioare de date proprii asupra cărora lucrează (figura 12.5).

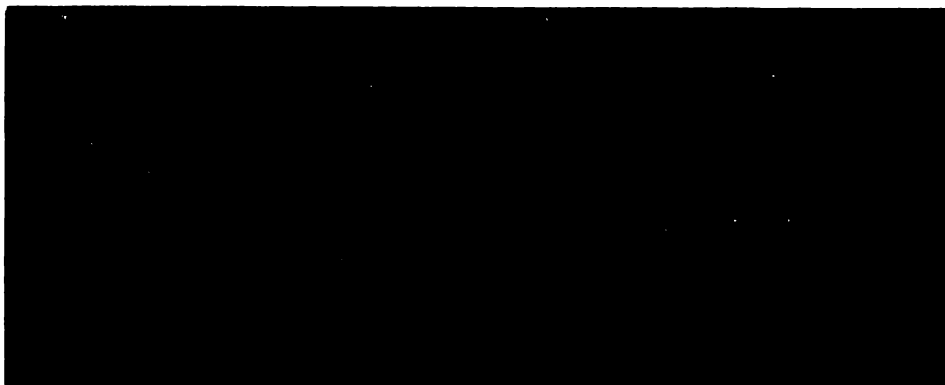


Fig. 12.5. Coexistența unei proceduri în apelul recursiv

În limbajul LOGO sînt implementate două tipuri de apeluri recursive și anume:

12. LOGO-TEHNICI, APLICAȚII

- mod 1 — în care apelul recursiv (autoapelul) se face ca ultima instrucțiune din procedură (figura 12.6.a)
- mod 2 — în care apelul recursiv (autoapelul) se face în cadrul procedurii, însă nu ca ultima instrucțiune din procedură (figura 12.6.b).



Fig. 12.6. Tipuri de implementare a apelurilor recursive

În alte limbaje de programare care permit recursivitatea nu există deosebire între cele două moduri de implementare.

În LOGO există o deosebire importantă care constă în:

- modul 1 — nu reprezintă un apel recursiv propriu-zis (nu se face salvare/refacere context) ci se lucrează pe același set de date proprii.
- modul 2 — reprezintă un apel recursiv cu salvare context, activare set de date proprii apelului curent, refacere context.

Modul 1 este mai degrabă o extensie a primitivei REPEAT, fiind asemănătoare cu un salt necondiționat la prima instrucțiune din procedură.

Grupul de instrucțiuni: A
END

apel de procedură și END sînt tratate împreună și echivalate cu transferul controlului la prima instrucțiune din procedură, fără a se mai face salvare context, activare set de date proprii apelului, refacere context, ci numai actualizarea setului de date proprii.

În acest fel se poate repeta un grup de instrucțiuni, de un număr de ori nespecificat la primul apel de procedură, fără a avea constrîngerii din punct de vedere al memoriei.

EXEMPLU:

Grupul de instrucțiuni FD 10 RT 30 se repetă la infinit în secvența:

TO A		FD 10
FD 10		RT 30
RT 30	echivalent cu	FD 10
A		RT 30
END		.
		.
		FD 10
		RT 30
		.
		.
		.

și de N ori în secvența:

TO A :N		FD 10 :	}1
IF :N = 0 [STOP]		RT 30 :	
FD 10		FD 10 :	}2
RT 30	echivalent cu	RT 30 :	
A :N-1		.	
END		.	
		FD 10 :	}N
		RT 30 :	

În cazul general de recursivitate, modul 2, datorită faptului că se salvează contextul (fiind necesar în momentul revenirii în procedură) este nevoie de memorie și după un număr mare de apeluri se poate ajunge la epuizarea spațiului de memorie disponibil.

Reamintim că o procedură se termină în trei moduri:

- prin execuția primitivei END
- prin execuția primitivei STOP
- prin execuția primitivei OUTPUT

Așa cum am precizat mai înainte, o altă posibilitate de a repeta un grup de instrucțiuni din cadrul unui program sau a unei proceduri, decât utilizarea primitivei REPEAT, este aceea de a include în procedura respectivă o apelare a ei însăși.

EXEMPLU:

```
TO TRIUNGHI :L
FORWARD :L
RIGHT 120
TRIUNGHI :L
END
```

— va desena un triunghi cu latura L și va repeta mereu desenzarea acestuia pînă cînd se intrerupe programul cu BREAK.

Deci procedura de mai sus este echivalentă cu:

```
FORWARD :L
RIGHT 120
FORWARD :L
RIGHT 120
FORWARD :L
RIGHT 120
FORWARD :L
RIGHT 120
```

```
·
·
·
```

Practic se execută instrucțiunile:

```
FORWARD :L
RIGHT 120
```

— după care urmează o apelare a ei însăși, ce va conduce din nou la execuția instrucțiunilor:

```
FORWARD :L
RIGHT 120
```

și o nouă apelare etc.

Evident că în cazurile reale trebuie să existe o modalitate de repetare finită a instrucțiunilor respective, modalitate ce se realizează în general printr-o primitivă de testare a unei condiții.

Exemplu: Să construim un turn ca cel din figura 12.7, construit din plăcate a căror latură se micșorează continuu.

Operația de construire a turnului este un proces repetitiv. Să presupunem că nu ne interesează numărul de piese puse una peste alta, ci următoarele caracteristici:

- latura piesei de bază L_1 = valoarea specificată de utilizator;
- relația între latura piesei următoare și a celei curente:
 $L_{i+1} = 0.5 * L_i$ ($L_2 = 0.5 * L_1$; $L_3 = 0.5 * L_2$, etc.);
- ultima piesă să nu aibă latură mai mică de 2 puncte $L_n > 2$.

Construirea turnului se va face printr-o procedură recursivă (se va apela pe ea însăși) și de fiecare dată se va modifica parametrul de apelare și totodată se va testa dacă latura L_n a devenit mai mică decât 2. În caz afirmativ se va încheia procesul de desenare.

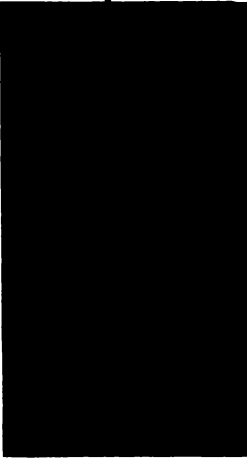


Fig. 12.7. TURN construit prin recursivitate

```

TO TURN :L
IF :L < 2 [STOP]
PĂTRAT :L
FORWARD :L
RIGHT 90
FORWARD :L *.25
LEFT 90
TURN :L *.5
END

```

Corpul procedurii va genera un desen de bază ca cel din figura 12.8, care va constitui piesa de bază a procesului de repetare.

În cadrul procedurii se apelează o procedură de construire a pătratului.

```

TO PĂTRAT :L
REPEAT 4 [FORWARD :L RIGHT 90]
END

```

- după care se mută penelul în poziția necesară construirii următoarei piese a turnului.

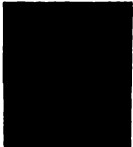


Fig. 12.8. Piesa de bază în construirea repetitivă a turnului

Construirea turnului se realizează apelînd procedura TURN 32

Procesul se repetă prin apelarea recursivă a procedurii TURN cu altă valoare a laturii piesei și de fiecare dată în cadrul procedurii se verifică dacă latura nu a devenit mai mică decât 2.

Ambele proceduri TRIUNGHI, TURN au fost realizate în primul mod de implementare (echivalente cu repetarea grupului de instrucțiuni).

EXEMPLU: Pentru a înțelege mai bine recursivitatea și pentru a scoate în evidență diferența dintre modul 1 și modul 2 de implementare să analizăm două proceduri care din punct de vedere al conținutului par asemănătoare însă din punct de vedere al execuției produc rezultate complet diferite:

```

TO NUMĂRĂ.INV :NR
IF :NR = 0 [STOP]
PRINT :NR
NUMĂRĂ.INV :NR-1
END

```

```
TO NUMĂRĂ.DIR :NR  
IF :NR = 0 [STOP]  
NUMĂRĂ.DIR :NR-1  
PRINT :NR  
END
```

Apelul acestor proceduri va genera următoarele rezultate:

NUMĂRĂ.DIR 4

1
2
3
4

NUMĂRĂ.INV 4

4
3
2
1

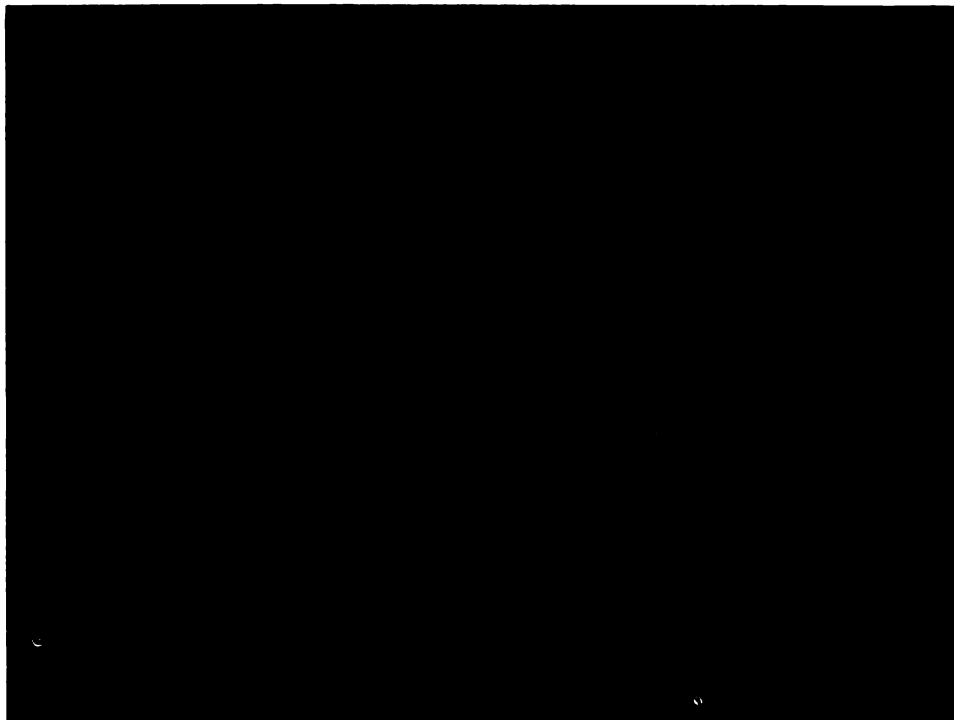


Fig. 12.9. Execuția instrucțiune cu instrucțiune a procedurii NUMARA.INV 4

Să analizăm procesul de execuție a primitivelor în ambele situații.

Execuția instrucțiune cu instrucțiune a procedurii NUMĂRĂ.INV 4 este prezentă în figura 12.9, iar a procedurii NUMĂRĂ.DIR 4 este prezentată în figura 12.10. Se recomandă ca parametrul NR să primească valori naturale în momentul apelării procedurilor.



Fig. 12.10. Execuția instrucțiune cu instrucțiune a procedurii NUMARA.DIR 4

Procedura NUMĂRĂ.INV are apelul recursiv ca ultimă instrucțiune din procedură, în timp ce procedura NUMĂRĂ.DIR, nu.

Procedura NUMĂRĂ.INV nu va salva contextul și nu va activa setul de date proprii apelului, ci va lucra pe același set de date pe care îl va actualiza în urma apelului.

Procedura NUMĂRĂ.DIR apelată cu parametrul 4, va stabili variabila NR la valoarea 4. Verifică dacă aceasta este 0. Fiind diferită de 0 continuă cu execuția instrucțiunii care apelează NUMĂRĂ.DIR :NR-1 (auto-apelare de procedură cu modificare de parametru).

LOGO va salva implicit contextul (valoarea variabilei :NR=4) și va activa setul de date proprii apelului, adică NR=3. Cele două seturi de date, cel al apelului cu NR=4 și al apelului NR=3 coexistă în memorie. În urma apelului se continuă cu instrucțiunea care verifică dacă variabila curentă (proprie apelului) a ajuns la zero. În caz contrar se continuă apelul recursiv până când variabila curentă ajunge la 0. În acel moment se execută instruc-

țiunea STOP care reprezintă terminarea apelului curent al procedurii, se reface implicit contextul și se revine în procedura chemătoare. Se continuă cu instrucțiunea PRINT :NR, care va produce afișarea valorii 1.

Întilnind instrucțiunea END se reface contextul și se revine în procedura chemătoare, etc.

În final se revine în regim de comandă.

EXEMPLU:

Recursivitatea se poate utiliza în mod eficient în proceduri care efectuează aceleași operații pentru toate elementele unui obiect. Vom exemplifica utilizarea recursivității cu două proceduri care scriu invers (de la dreapta la stînga) caracterele unui cuvînt sau cuvintele dintr-o propoziție (în general elementele unei liste).

```
TO DREAPTA.STÎNGA :C
IF EMPTY :C [OUTPUT " ]
OUTPUT WORD (LAST :C) (DREAPTA.STÎNGA . BUTLAST :C)
END
```

Procedura DREAPTA.STÎNGA generează la ieșire un cuvînt care are caracterele în ordinea inversă decît ale cuvîntului specificat ca argument de intrare.

```
PRINT DREAPTA.STÎNGA "MARE
```

ERAM

```
TO DREAPTA "STÎNGA.LISTA :L
IF EMPTY :L [OUTPUT :L]
OUTPUT SENTENCE (LAST :L) (DREAPTA.STÎNGA.LISTA BUTLAST:L)
END
```

Procedura DREAPTA.STÎNGA.LISTA generează la ieșire o listă care are elementele în ordine inversă, decît lista specificată ca argument de intrare.

```
PRINT DREAPTA.STÎNGA.LISTA [ȘCOALA ESTE FRUMOASĂ ȘI MARE]
MARE ȘI FRUMOASĂ ESTE ȘCOALA
```

Vom analiza ce se întîmplă pas cu pas în timpul execuției procedurii DREAPTA.STÎNGA "MARE și în mod asemănător se poate analiza și pentru DREAPTA.STÎNGA.LISTA.

Execuția instrucțiune cu instrucțiune se arată în figura 12.11.

Pentru a extrage caracterele unui cuvînt în ordine inversă și a realiza un nou cuvînt cu această ordine s-a utilizat o procedură care se apelează pe ea însăși, numai că de fiecare dată va acționa asupra unui cuvînt din care s-a extras ultimul caracter.

Procesul recursiv continuă pînă cînd se epuizează toate caracterele din cuvînt.

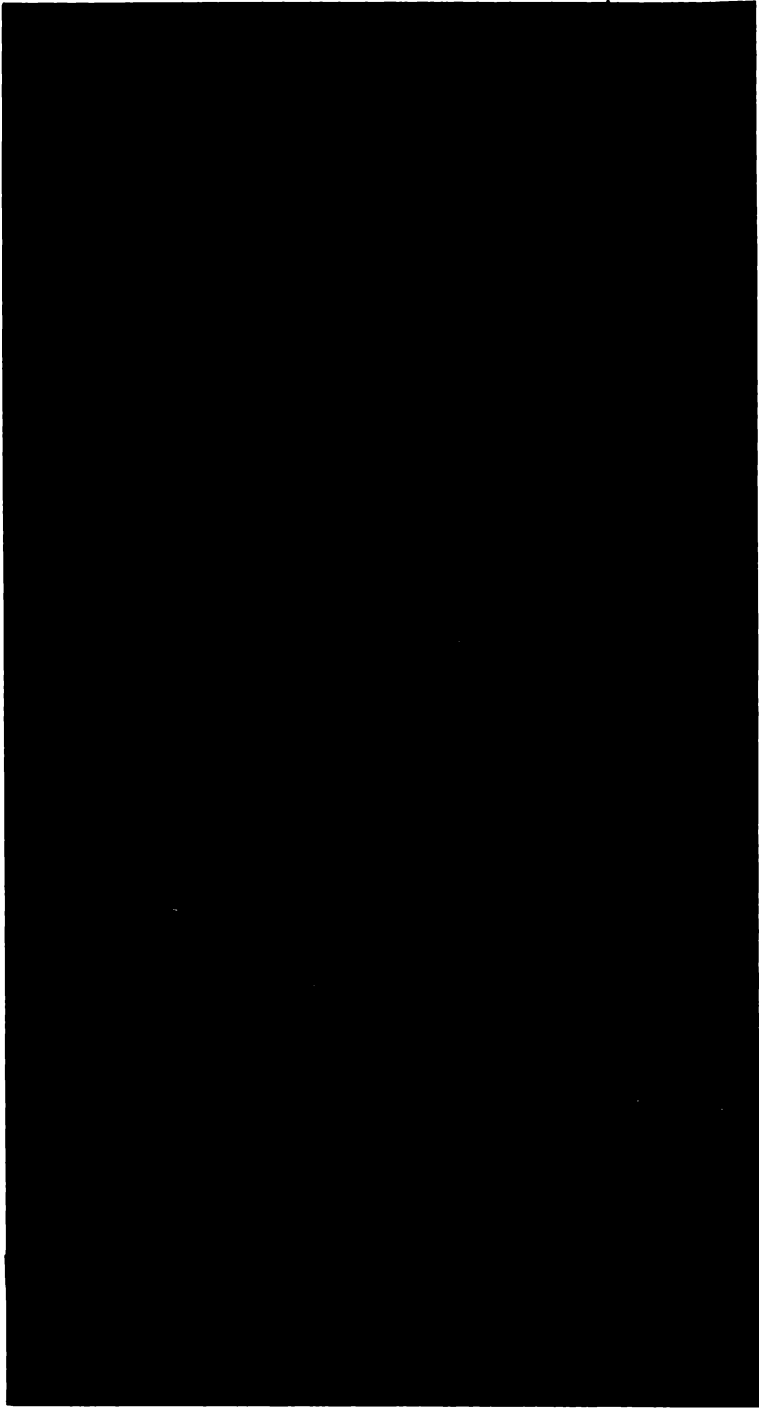


Fig. 12.11. Execuția instrucțiune cu instrucțiune a procedurii DREAPTA. STÎNGA

EXEMPLU:

Un alt exemplu de utilizare a recursivității este acela în care se desenează un arbore ca cel din figura 12.12.

Dacă analizăm atent, se observă că în fiecare nod putem considera că se construiește un nou arbore ale cărui ramuri au dimensiuni din ce în ce mai mici.

În nodurile extreme arborele se reduce la un arbore elementar format numai din două ramuri (figura 12.13).



Fig. 12.12. ARBORE desenat prin
recursivitate

Fig. 12.13. ARBORE elementar

Este evident faptul că pentru a construi arborele total se apelează recursiv în fiecare nod construirea unui arbore cu dimensiuni reduse.

Primitivele pentru construirea arborelui elementar sînt:

```
LEFT 45 ; desenarea ramurii stînga
FORWARD :L
-----
BACK :L
RIGHT 90 ;revenire și desenare ramura dreapta
FORWARD :L
-----
BACK :L
LEFT 45 ;revenire în poziția inițială
```

O precizare importantă care trebuie făcută este aceea că în procesul de recursivitate este necesar să se stabilească precis starea inițială a penelului.

În descrierea arborelui elementar s-a revenit în aceeași poziție și cu aceeași direcție ca în momentul începerii lui.

În procesul construirii arborelui total, desenăm mai întîi o ramură stînga a arborelui elementar, după care am ajuns într-un nod în care trebuie să considerăm că pornim procesul de construire a unui nou arbore.

Deci după:

```
TO ARBORE: L
LEFT 45 ;prima ramură stînga a arborelui
FORWARD :L
```

— trebuie să apelăm procedura ARBORE de construire a unui nou arbore cu un parametru de latură modificat.

Vom considera că un subarbore are ramura inițială jumătate cît cea a arborelui pe nivelul anterior.

12. LOGO-TEHNICI, APLICAȚII

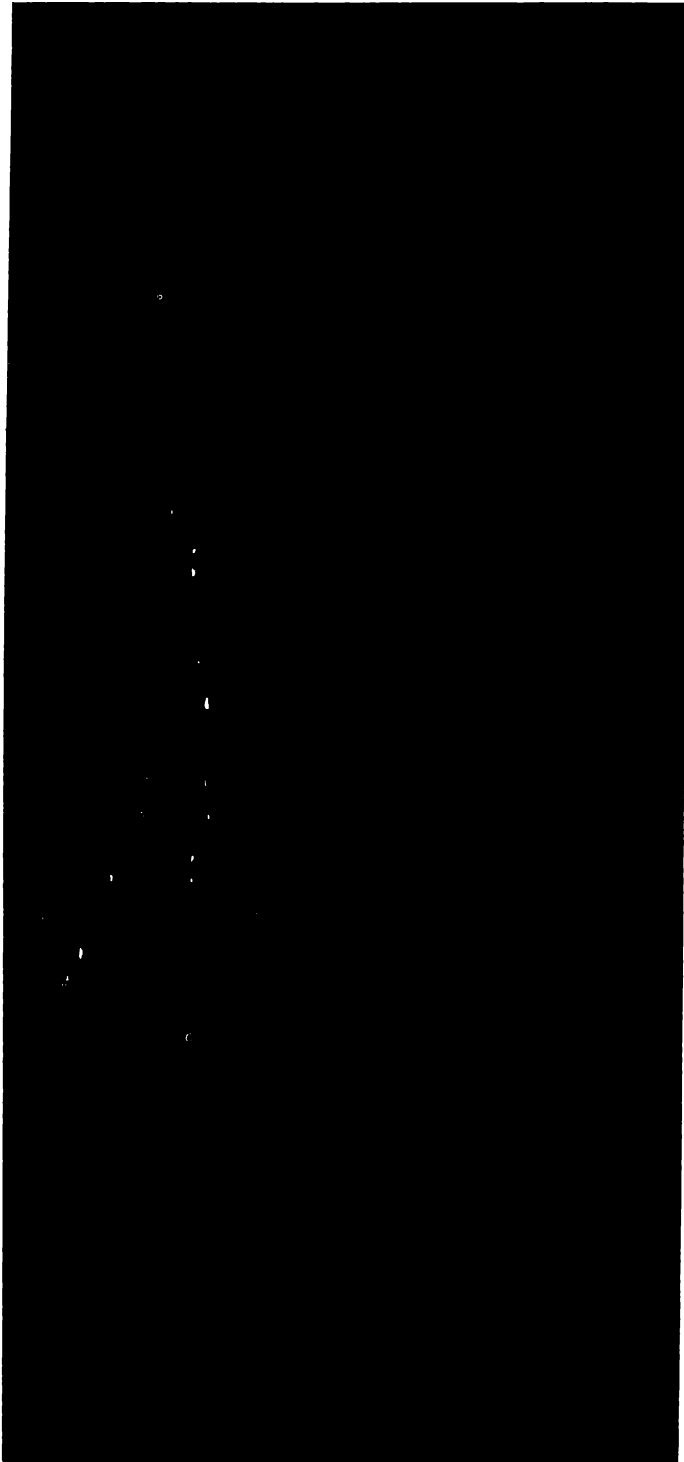
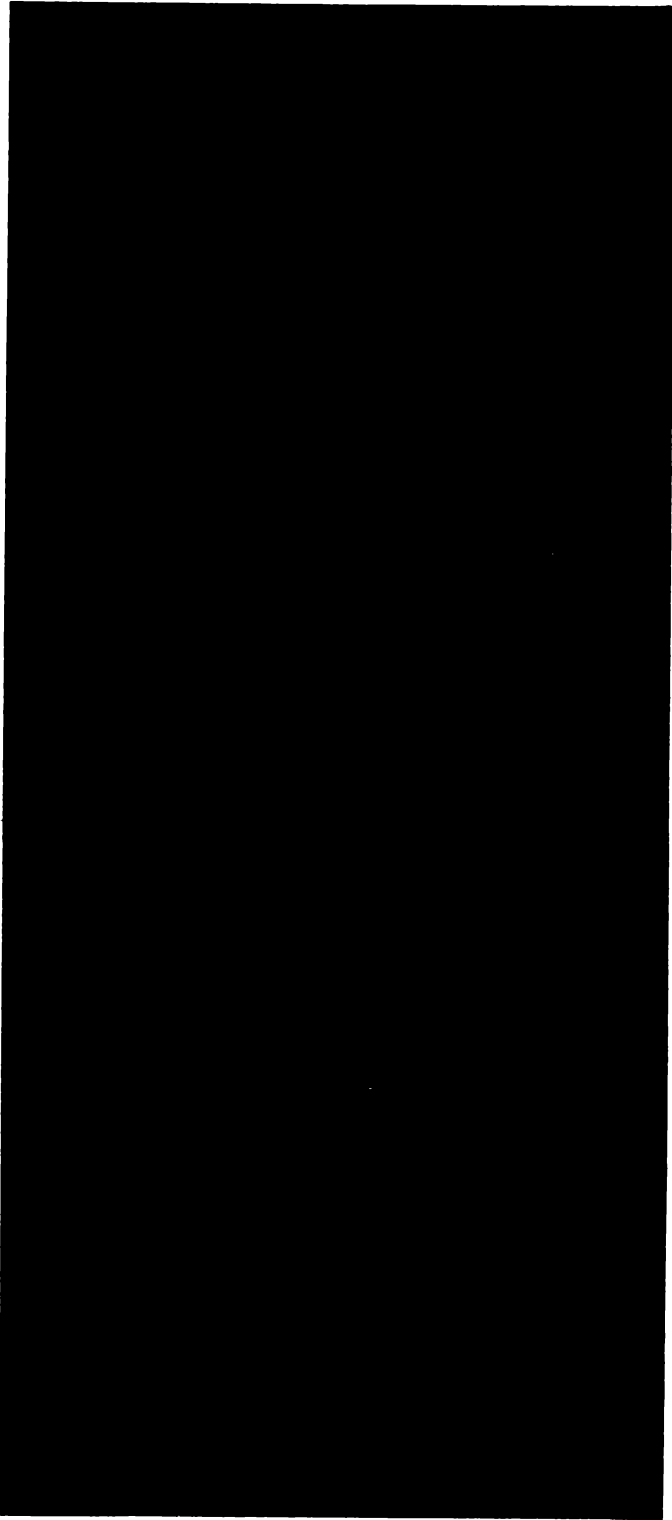


Fig. 12.14. Execuția in-
strucțiune cu instrucțiune
a procedurii ARBORE



```

TO ARBORE :L
LEFT 45 ;construirea ramurilor stnga a tuturor subarborilor
FORWARD :L
ARBORE :L/2

```

În momentul cînd s-a construit o ramură stînga extremă trebuie să revenim și să construim cealaltă ramură extremă (ramura dreapta).

```

TO ARBORE :L
LEFT 45
FORWARD :L
ARBORE :L/2
BACK :L
RIGHT 90
FORWARD :L

```

Se încearcă construirea unui arbore și pe ramura dreapta, după care revenim în poziția inițială.

```

TO ARBORE :L
LEFT 45 ; construiește ramurile stînga
FORWARD :L
ARBORE :L/2
.....
BACK :L
RIGHT 90 ; construiește ramurile dreapta
FORWARD :L
ARBORE :L/2
.....
BACK :L
LEFT 45 ; revine în poziția inițială
END

```

Procedura descrisă trebuie completată cu o posibilitate de terminare, deoarece procesul de construire a unei ramuri în speță (ramura stînga) nu se termină nici o dată.

Ca posibilitate de terminare a procesului recursiv se poate stabili momentul în care latura extremă ajunge sub o anumită valoare, de exemplu valoarea 5.

Procedura generală de desenare este:

```

TO ARBORE :L
IF :L < 5 [STOP]
LEFT 45
FORWARD :L
ARBORE :L/2
BACK :L
RIGHT 90
FORWARD :L
ARBORE :L/2
BACK :L
LEFT 45
END

```

sau

```

TO ARBORE :L
IF :L < 5 [STOP]
LT 45
FD :L
ARBORE :L/2
BK :L
RT 90
FD :L
ARBORE :L/2
BK :L
LT 45
END

```

Parcursul instrucțiune cu instrucțiune a procedurii ARBORE 32 este arătată în figura 12.14.

Rezultatele grafice intermediare ale diverselor puncte din procedură sînt arătate în figura 12.15, unde în f1...f21 sînt desenele intermediare.

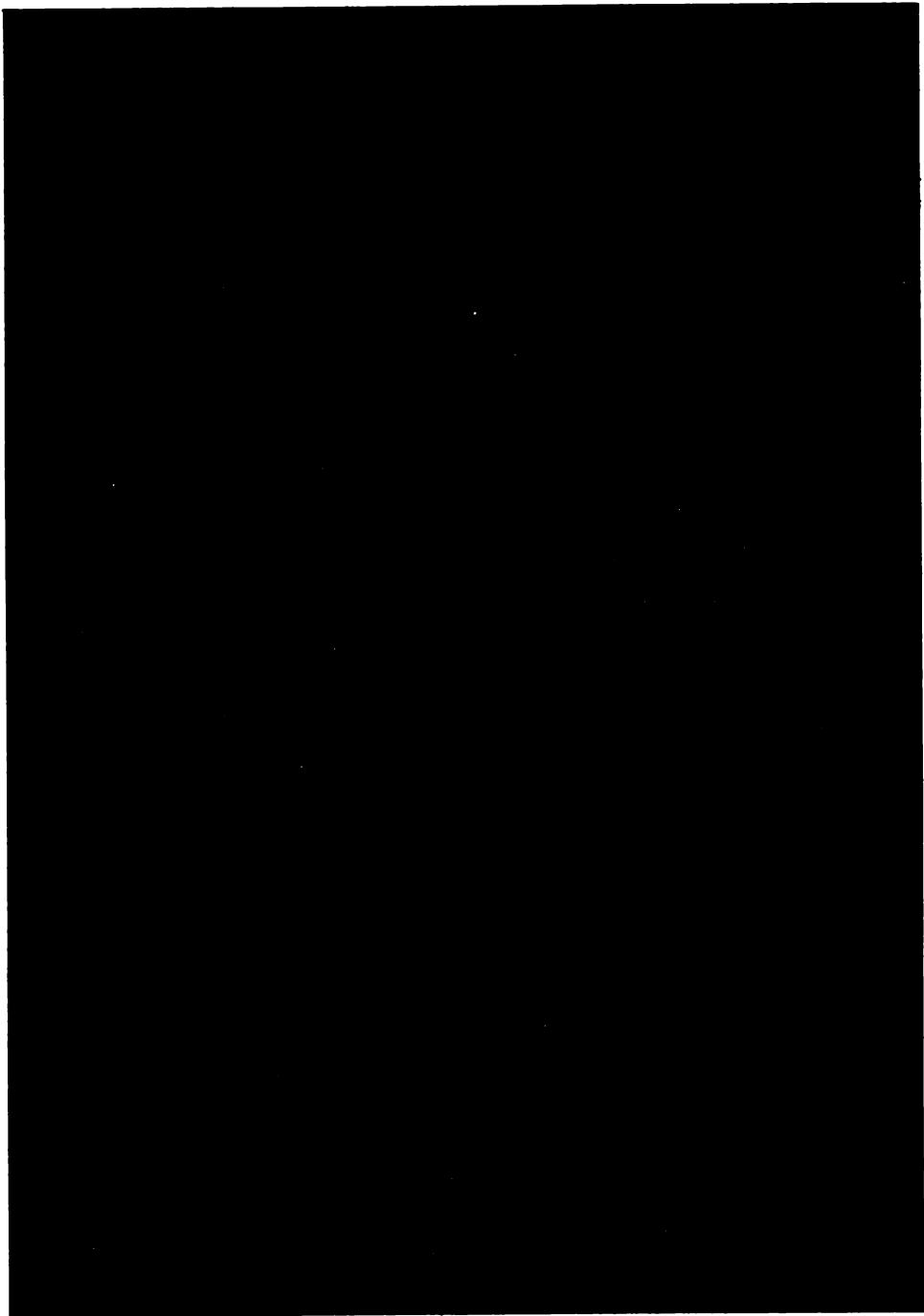


Fig. 12.15_{1,20} Desenarea pas cu pas a arborelui (12.15_{1,20})

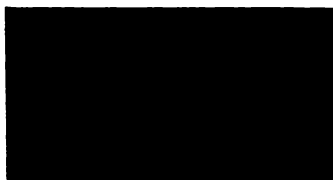


Fig. 12.15_{1,21} Desenarea pas cu pas a arborelui utilizind recursivitatea

12.3. Mici aplicații în LOGO

12.3.1. Exemple grafice

PĂTRAT CU LATURA DE 40:

TO PĂTRAT	sau	TO PĂTRAT
FORWARD 40		FD 40
RIGHT 90		RT 90
FORWARD 40		FD 40
RIGHT 90		RT 90
FORWARD 40		FD 40
RIGHT 90		RT 90
FORWARD 40		FD 40
RIGHT 90		RT 90
END		END

sau

```
TO PĂTRAT  
REPEAT 4[FORWARD 40 RIGHT 90]  
END
```

sau

```
TO PĂTRAT  
REPEAT 4 [FD 40 RT 90]  
END
```

PĂTRAT CU LATURA VRIABILĂ:

```
TO PĂTRAT :L  
REPEAT 4[FD :L RT 90]  
END
```

DREPTUNGHI cu lățimea de 40
și lungimea de 90:

```
TO DREPTUNGHI  
FD 40  
RT 90  
FD 90  
RT 90  
FD 40  
RT 90  
FD 90  
RT 90  
END
```


TO DREPTUNGHI
REPEÁT 2 [FD 40 RT 90 FD 60 RT 90]
END

DREPTUNGHI CU LATURILE VARIABLE:

TO DREPTUNGHI :I :L
REPEAT 2[FD :I RT 90 FD :L RT 90]
END

TRIUNGHI ECHILATERAL:

TO TRIUNGHI
REPEAT 3[FD 40 RT 120]
END

TRIUNGHI ECHILATERAL CU LATURA VARIABILĂ:

TO TRIUNGHI :L
REPEAT 3[FD :L RT 120]
END

TRIUNGHI ÎN CARE SE CUNOSC LATURILE ȘI DOUĂ UNGHIURI:

TO TRIUNGHI :LA :LB :LC :B :C
FD :LA
RT 180—:B
FD :LB
RT 180—:C
FD :LC
END

ARC DE CERC DREAPTA CU DESCHIDERE 10°

TO ARCD :R
RT 5
FD :R * (3.14159/18)
RT 5
END

ARC DE CERC STÎNGA CU DESCHIDEREA 10°

TO ARCS :R
LT 5
FD :R * (3.14159/18)
LT 5
END

CERC DREAPTA DE RAZA R

TO CERCD :R
REPEAT 36 [ARCD :R]
END

CERC STÎNGA DE RAZA R

TO CERCS :R
REPEAT 36 [ARCS :R]
END

CERC DE ORIGINE X, Y ȘI RAZĂ R

TO CERC :x :y :R
PU
SEPTOS SE :x — :R :y
PD
CERCD :R
END

```

sau
TO CERC :x :y :R
PU
SEPTOS SE :x + :R :y
PD
CERCS :R
END

```

**EXEMPLU DE UTILIZARE ARCD ȘI ARCS
RAZA DE SOARE**

```

TO RAZAS :L
REPEAT 2[ARCD :L ARCS :L]
END

```

SOARE

```

TO SOARE :L
REPEAT 9[RAZAS :L RT 160]
END

```

Se recomandă SOARE 18

POLIGON DE LATURA L ȘI UNGHI U

```

TO POLIGON :L :U
FD :L
RT :U
POLIGON :L :U
END

```

De remarcat că desenul nu se termină nici o dată.
Trebuie oprit cu break.

Se obțin figuri interesante pentru următoarele valori:

L	U
50	160
60	80
80	144
20	40
80	156
60	90
50	60

SPIRALA

```

TO SPIRALA :L :U :INC
FD :L
RT :U
SPIRALA (:L + :INC) :U :INC
END

```

De remarcat că programul nu se termină decât cu break
Figurile interesante se obțin pentru:

L	U	INC
1	75	1
1	45	1
1	45	3
5	120	3
1	144	3
1	150	1
1	88	1

BRAD

```
TO RAMURA :L
FD :L
RT 120
FD :L
BK :L
LT 60
BK :L
FD :L
LT 60
END

TO BRAD :L
IF :L < 2 [STOP]
RAMURA :L
BRAD :L . 0.6
END

BRAD 30
```

12.3.2. Suma a două numere aleatoare

Vom scrie un program simplu care permite verificarea cunoștințelor despre operațiile aritmetice elementare.

Programul va genera două numere aleatoare între 0 și 1000 și va întreba utilizatorul cât este suma acestora. În funcție de răspunsul primit va preciza dacă acesta este corect sau nu și în cazul în care răspunsul este incorect va specifica răspunsul bun.

Procesul de întrebare / răspuns va continua pînă cînd utilizatorul va întrerupe execuția programului cu BREAK.

TO SUMA

```
MAKE "NUM1 RANDOM 1000
MAKE "NUM2 RANDOM 1000
MAKE "REZ :NUM1 + :NUM2
PRINT (SENTENCE [CÎT FACE] :NUM1 [+] : NUM2)
MAKE "RASP READLIST
IF :REZ = FIRST :RASP [PRINT( RĂSPUNS CORECT )SUMA]
PRINT SENTENCE [RĂSPUNSUL CORECT ESTE] :REZ
SUMA
END
```

În primele două instrucțiuni se atribuie variabilelor NUM1 și NUM2 cîte un număr, aleator, cuprins între 0 și 1000.

Suma acestor două numere, care constituie rezultatul se atribuie variabilei REZ.

Se constituie o listă formată din elementele

- lista [CÎT FACE]
- variabila NUM1
- lista [+]
- variabila NUM2

care se tipărește, reprezentînd întrebarea la care trebuie să răspundă utilizatorul.

Răspunsul utilizatorului se atribuie variabilei RĂSP.

Se verifică dacă răspunsul RĂSP (care fiind sub forma de listă trebuie să-i extragem primul element FIRST :RĂSP) este egal cu rezultatul REZ. Dacă da, se tipărește mesajul răspuns corect și se continuă. În caz contrar, se indică care este răspunsul corect și se continuă.

Trebuie menționat faptul că acest program simplu nu este protejat la introducerea unor răspunsuri care nu sînt numere.

Pentru acesta este necesar să se introducă o procedură care verifică acest lucru.

```
TO CITNR
MAKE "NR FIRST READLIST
IF NUMBERP :NR [OUTPUT :NR]
PRINT [VA ROG INTRODUCEȚI NUMĂR]
OUTPUT CITNR
END
```

Procedura verifică dacă răspunsul este număr.

În caz afirmativ, generează ca ieșire numărul respectiv iar în caz contrar anunță faptul că trebuie introdus, ca răspuns, un număr și așteaptă introducerea acestuia.

Programul se modifică în felul următor:

```
TO SUMA
MAKE "NUM1 RANDOM 1000
MAKE "NUM2 RANDOM 1000
MAKE "REZ :NUM1 + :NUM2
PRIN (SENTENCE [CIT FACE] :NUM1 [+] :NUM2)
MAKE "RASP CITNR
IF :REZ = :RASP [PRINT [RĂSPUNS CORECT] SUMA]
PRINT SENTENCE [RĂSPUNSUL CORECT ESTE] :REZ
SUMA
END
```

Acest program simplu poate fi extins. Ca exercițiu pentru cititor sugerăm completarea programului cu următoarele facilități:

— dacă utilizatorul greșește răspunsul, programul să nu indice imediat care este rezultatul, ci să întrebe de 2—3 ori pe utilizator și numai după aceea să-i indice răspunsul corect.

— să se alcătuiască un punctaj al răspunsurilor corecte/incorecte și să se configureze pentru diverse operații +, —, *, /.

— dacă utilizatorul răspunde corect, întrebările să nu mai fie simple, ci să le transforme în expresii din ce în ce mai complicate.

12.3.3. Ordonare alfabetică

Să se scrie un program LOGO care are ca intrare o listă de elemente într-o ordine oarecare și generează la ieșire o altă listă ordonată alfabetic

Principiul de sortare constă în extragerea element cu element din lista sursă (inițială) și transferarea lor în lista destinație (finală). În timpul transferării elementul curent este inserat în lista destinație astfel încît aceasta să fie ordonată alfabetic.

Programul poate fi utilizat și pentru a concatena două liste, iar rezultatul să fie ordonat alfabetic, cu condiția ca una din liste să fie deja ordonată.

În procesul de transferare este necesară o procedură de inserare a unui element în lista destinație.

```
TO INSERARE :ELEM :LISTA
IF EMPTY? :LISTA [ OUTPUT LIST :ELEM " ]
IF ORDONAT :ELEM FIRST :LISTA [ OUTPUT FPUT :ELEM :LISTA ]
OUTPUT FPUT FIRST :LISTA INSERARE :ELEM BUTFIRST :LISTA
END
```

Dacă lista destinație LISTA este vidă, atunci se creează o listă cu elementul ELEM pe care dorim să-l introducem. În caz contrar verificăm dacă elementul își are locul înaintea tuturor elementelor din lista destinație. Dacă da, îl introducem în listă. Dacă nu, verificăm pe rând dacă își are locul după primul, al doilea, etc. element din lista destinație, apelând recursiv procesul de inserare aplicat unei liste din care eliminăm rând pe rând elementele care trebuie să fie înaintea elementului ELEM specificat.

Procedura ORDONAT generează valoarea logică TRUE sau FALSE dacă primul argument este înaintea celui de-al doilea argument în ceea ce privește ordinea alfabetică. Compararea se face pe baza codurilor ASCII.

```
TO ORDONAT :ARG1 :ARG2
IF :ARG1 = " [OUTPUT TRUE ]
IF :ARG2 = " [ OUTPUT FALSE ]
IF (ASCII FIRST :ARG1) < (ASCII FIRST :ARG2) [ OUTPUT TRUE ]
IF (ASCII FIRST :ARG1) > (ASCII FIRST :ARG2) [OUTPUT FALSE ]
OUTPUT ORDONAT BUTFIRST :ARG1 BUTFIRST :ARG2
END
```

Procedura de sortare SORTARE se poate scrie astfel:

```
TO SORTARE: :LISTAI :LISTAF
IF EMPTY? :LISTAI [OUTPUT :LISTAF]
MAKE "LISTAF INSERARE FIRST :LISTAI :LISTAF
OUTPUT SORTARE BUTFIRST :LISTAI :LISTAF
END
```

Exemplu de utilizare:

```
MAKE "LISTA.SORTATA SORTARE [M C A B F] [ ]
PRINT :LISTA.SORTATA
A B C F M
MAKE "LISTA.SORTATA SORTARE [ELEV BINE NOTA] :LISTA.SORTATA
PRINT :LISTA.SORTATA
A B BINE C ELEV F NOTA M
```

12.3.4. Conversia numerelor naturale din baza 10 într-o bază < 16

Să se scrie un program care realizează conversia numărului natural N din baza 10 într-o bază B < 16.

Se știe că un număr în baza B folosește pentru reprezentare cifrele utilizate în baza respectivă. Astfel, în baza 16 se utilizează:

0 1 2 3 4 5 6 7 8 9 A B C D E F

iar într-o bază < 16 se utilizează un subset al acestora.

Conversia unui număr întreg se realizează prin concatenarea resturilor împărțirii numărului la baza în care dorim conversia, în ordinea inversă obținerii acestora.

```
TO CONVERSIE :N :B
IF :B < 2 [PRINT [BAZA DE CONVERSIE TREBUIE >=2] STOP]
IF :B > 16 [PRINT [BAZA DE CONVERSIE TREBUIE < 16] STOP]
IF :N = 0 [OUTPUT 0]
OUTPUT CONV :N :B
END
```

```
TO CONV :N :B
IF :N = 0 [OUTPUT "=" ]
OUTPUT WORD (CONV (INT DIV :N :B) :B) (CIFRA (REMAINDER :N :B))
END
```

```
TO CIFRA :R
IF :R < 10 [OUTPUT :R]
IF :R = 10 [OUTPUT "A"]
IF :R = 11 [OUTPUT "B"]
IF :R = 12 [OUTPUT "C"]
IF :R = 13 [OUTPUT "D"]
IF :R = 14 [OUTPUT "E"]
IF :R = 15 [OUTPUT "F"]
END
```

Procedura CIFRA are ca intrare restul curent obținut în procesul de conversie. Acesta este pus în corespondență cu cifra de reprezentare în baza B.

Procedura CONV este cea care realizează efectiv conversia din baza 10 în baza B a numărului N.

Printr-o chemare recursivă, având de fiecare dată ca intrare citul dintre număr și baza în care vrem să convertim, se generează resturile în ordinea inversă obținerii lor. De fiecare dată se apelează procedura CIFRA de asociere a restului cu cifra de reprezentare în baza B.

Programul principal verifică dacă baza de conversie este <16. În caz contrar, se specifică un mesaj și se abandonează procesul de conversie.

În caz afirmativ se verifică dacă numărul este 0.

Dacă numărul este diferit de 0 și baza mai mică decât 16 se apelează procedura CONV ce realizează conversia propriu-zisă.

Exemplu:
PRINT CONV 29 2
=11101

12.3.5. Conversia dintr-o bază oarecare în altă bază

Programul de conversie poate fi generalizat, pentru a realiza conversia unui număr reprezentat într-o bază oarecare în altă bază.

Pentru a evita efectuarea calculelor în altă bază decât baza 10, procesul de conversie va consta în două etape și anume:

etapa 1 — convertește numărul din baza în care este reprezentat, în baza 10, procedura BAZA.ZEC

etapa 2 — convertește reprezentarea în baza 10, obținută în urma etapei 1, în baza în care se dorește conversia finală, procedura ZEC. BAZA

Este necesar să existe o procedură care pune în corespondență valoarea în zecimal a unei cifre într-o bază oarecare cu codul ASCII al cifrei respective, pentru a putea realiza tipărirea ei (ZEC.ASCII).

De asemenea, trebuie să existe o procedură care avînd ca intrare caracterul ASCII al unei cifre într-o bază oarecare să genereze valoarea în zecimal corespunzătoare (ASCII.ZEC).

OBSERVAȚIE: Codul ASCII reprezintă codificarea binară a caracterelor, cifrelor și semnelor speciale

Simbolurile de reprezentare a cifrelor în diverse baze le putem considera astfel:

baza k	baza 10	baza m
0	0	0
1	1	1
2	2	2
3	3	3
4	4	4
5	5	5
6	6	6
7	7	7
8	8	8
9	9	9
A	10	A
B	11	B
C	12	C
D	13	D
E	14	E
F	15	F
G	16	G
.	17	H
.	18	I
.	.	.
ASCII.ZEC →	.	→ ZEC.ASCII
ZEC.ASCII ←	.	← ASCII.ZEC

```

TO ZEC.ASCII :N
IF :N < 10 [OUTPUT :N]
OUTPUT CHAR 55+:N
END

```

Procedura ZEC.ASCII are ca intrare un număr zecimal N și generează codul ASCII corespunzător unei baze de reprezentare oarecare.

Dacă N > 10 trebuie generat unul dintre simbolurile A, B, C, ...

Avînd în vedere că A are codul 65, B are codul 66, etc rezultă că pentru valorile mai mari decît 10 trebuie să adunăm 55 pentru a ajunge la codul ASCII corespunzător literelor mari A, B, C, ...

```

TO ASCII.ZEC :A
IF NUMBERP :A [OUTPUT :A]
OUTPUT (ASCII :A) - 55
END

```

12. LOGO-TEHNICI, APLICAȚII

Procedura ASCII.ZEC are ca intrare caracterul ASCII al unei cifre intr-o bază oarecare și generează la ieșire valoarea zecimală corespunzătoare. Realizează funcția inversă lui ZEC.ASCII.

```

TO ZEC.BAZA :N :BAZA
IF :N < :BAZA [OUTPUT ZEC.ASCII :N]
OUTPUT LIST ZEC.BAZA INT DIV :N :BAZA :BAZA
          ZEC.ASCII REMAINDER :N :BAZA
END

```

Procedura ZEC.BAZA convertește un număr din baza 10 într-o bază oarecare BAZA.

ZEC.ASCII REMAINDER :N :BAZA va genera caracterul ASCII asociat restului împărțirii numărului N la baza de conversie BAZA. Deci va genera cifra curentă a numărului în baza BAZA.

ZEC.BAZA INT DIV :N :BAZA :BAZA realizează printr-un proces recursiv operațiile de obținere a citurilor, pînă cînd citul curent este mai mic decît baza nouă de reprezentare.

```

TO      BAZA.ZEC :N :BAZA :P
IF      EMPTY   :N [OUTPUT 0]
OUTPUT (:P * ASCII.ZEC LAST :N)+
          BAZA.ZEC BUTLAST :N :BAZA :P * :BAZA
END

```

Procedura BAZA.ZEC convertește un număr N reprezentat în baza BAZA, în baza 10. Procedura se bazează pe următoarea relație:

$$\overline{(a_m a_{m-1} \dots a_0)} = a_m \cdot b^{m-1} + a_{m-1} \cdot b^{m-2} + \dots + a_1 \cdot b^1 + a_0$$

EXEMPLU:

$$176 = 1 \times 8 \times 8 + 7 \times 8 + 6 \times 1$$

Parametrul P, în momentul apelării rutinei va fi 1 iar în procesul recursiv de apelare va căpăta valorile BAZA, BAZA * BAZA, etc. ce constituie cantitățile cu care se înmulțește fiecare cifră de reprezentare în baza BAZAF.

```

TO CONVERSIE.BAZAI.BAZAF :N :BAZAI :BAZAF
IF :BAZAF < 2 [PRINT [BAZAI > 2] STOP]
OUTPUT ZEC.BAZA BAZA.ZEC :N :BAZAI 1 :BAZAF
END

```

Procedura CONVERSIE.BAZAI.BAZAF constituie programul principal de conversie a unui număr din baza inițială BAZAI în baza finală BAZAF.

Convertește numărul N din baza inițială BAZAI în zecimal apelînd procedura BAZA.ZEC după care se convertește numărul zecimal obținut, în baza finală BAZAF apelînd procedura ZEC.BAZA.

Ca exercițiu, propunem cititorilor să rescrie procedura BAZA.ZEC bazîndu-se pe relația:

$$\overline{(a_m a_{m-1} \dots a_0)} = (((a_m * b + a_{m-1}) * b + \dots + a_1) * b + a_0)$$

Ca exemplu de utilizare a procedurii de conversie generalizată arătăm cum se folosește în realizarea conversiei din zecimal în hexa și invers.


```

TO ZECIMAL.HEX :N
OUTPUT CONVERSIE.BAZAI.BAZAF :N 10 16
END

```

```

TO HEX.ZECIMAL :N
OUTPUT CONVERSIE.BAZAI.BAZAF :N 16 10
END

```

12.3.6. Desenează interactiv

Să se scrie un program LOGO care are următoarele specificații:

— citește starea claviaturii și în funcție de tasta apăsată execută una din comenzile:

```

F — deplasează penelul înainte cu valoarea 10
R — rotește dreapta penelul cu 30 grade
L — rotește stînga penelul cu 30 grade
C — șterge ecranul
S — anulează ultima comandă executată
N — definește o figură
D — desenează o figură al cărei nume se specifică
A — afișează programul asociat unei figuri
M — afișează meniul de comenzi

```

— în funcție de comanda specificată de la tastatură execută operația asociată și în același timp generează o listă de primitive ce în final va constitui un program LOGO.

Utilizatorul nu va trebui să scrie un program care să deseneze o figură, ci acesta va fi generat automat în urma mișcării penelului pe ecran cu comenzile F, R, L, D, etc.

```

TO MENU
TEXTSCREEN
CLEARTEXT
PRINT [F: DEPLASEAZĂ PENELUL ÎNAINTE 10 PUNCTE]
PRINT [R: ROTEȘTE PENELUL DREAPTA CU 30 GRADE]
PRINT [L: ROTEȘTE PENELUL STÎNGA CU 30 GRADE]
PRINT [C: ȘTERGE ECRANUL (CLEARSCREEN)]
PRINT [S: ȘTERGE ULTIMA COMANDĂ]
PRINT [N: DEFINEȘTE O FIGURĂ]
PRINT [D: DESENEAZĂ O FIGURĂ SPECIFICATĂ]
PRINT [A: AFIȘEAZĂ PROGRAMUL REZULTAT PENTRU FIGURA]
PRINT [M: AFIȘEAZĂ MENIUL]
PRINT [ ] PRINT [ ]
PRINT [ ] APĂSAȚI ORICE TASTĂ PENTRU A CONTINUA]
PRINT READCHAR
END

```

Procedura MENU (asociată comenzii M) afișează pe ecran comenzile asociate diferitelor taste F, R, L, etc.

Pentru a scrie programul este necesar să scriem câte o procedură pentru fiecare comandă în parte

```

TO ȘTERGE.ECRAN
CLEARSCREEN
MAKE "FIGURA [ ]
END

```

12. LOGO-TEHNICI, APLICAȚII

Procedura ȘTERGE.ECRAN (asociată comenzii C) are ca efect ștergerea ecranului în regim grafic și anularea listei de primitive ce se asociază figurii ce se desenează.

Variabila FIGURA este o listă ce va păstra istoria comenzilor ce se execută în timpul desenării unei figuri.

În final se va constitui programul ce va desena figura, care a fost pregătită în mod interactiv.

```
TO ANULEAZĂ.COM
  IF :FIGURA = [ ] [STOP]
  MAKE "FIGURA BUTLAST :FIGURA
  CLEARSCREEN
  TRASEAZĂ.FIG :FIGURA
END
```

Procedura ANULEAZĂ.COM (asociată comenzii S) șterge ultima comandă care a fost introdusă (eliminând primitiva din lista asociată figurii) șterge ecranul și redesenează figura.

```
TO TRASEAZĂ.FIG :COMENZI
  IF :COMENZI = [ ] [STOP]
  RUN FIRST :COMENZI
  TRASEAZĂ.FIG BUTFIRST :COMENZI
END
```

Procedura TRASEAZĂ.FIG trasează o figură, executând comandă cu comandă dintr-o listă de comenzi specificată.

```
TO NUME.FIG
  PRINT [CARE ESTE NUMELE FIGURII?]
  TRASEAZĂ.ȘI.ÎNREG FPUT FIRST READLIST [ ]
END
```

Procedura NUME.FIG (asociată execuției comenzii D — desenează o figură al cărei nume a fost specificat) cere utilizatorului numele figurii ce se dorește să fie afișată și o trasează.

```
TO TRASEAZĂ.ȘI.ÎNREG :ACȚIUNE
  RUN :ACȚIUNE
  MAKE "FIGURA (LPUT :ACȚIUNE :FIGURA)
END
```

Procedura TRASEAZĂ.ȘI.ÎNREG (asociată comenzilor F, R, L) are ca efect executarea acțiunii asociată comenzilor F, R, L, și înregistrarea primitivei corespunzătoare în lista ce va constitui în final programul de desenare a figurii.

```
TO ÎNVAȚĂ
  PRINT [CUM SE NUMEȘTE PROGRAMUL ASOCIAT FIGURII?]
  MAKE "NUME (FIRST READLIST)
  DEFINE :NUME (FPUT [ ] :FIGURA)
  ȘTERGE.ECRAN
END
```

Procedura ÎNVAȚĂ (asociată comenzii N) cere utilizatorului numele programului ce se va crea pe baza listei de comenzi creată interactiv. Se creează programul ce generează figura respectivă după care se șterge ecranul, și se așteaptă începutul unei noi sesiuni de lucru pentru definirea unei alte figuri.

```

TO AFIS.PROG
TEXTSCREEN
PRINT [NUMELE PROGRAMULUI ASOCIAT FIGURII]
MAKE "NUME FIRST READLIST
PO :NUME
END

```

Procedura AFIS.PROG (asociată comenzii A) afișează programul LOGO rezultat pentru figura desenată interactiv și salvată cu comanda N.

```

TO EXECUTĂ.COMANDA :COM
IF :COM = "F [TRASEAZĂ.ȘI.ÎNREG [FORWARD 10] STOP]
IF :COM = "R [TRASEAZĂ.ȘI.ÎNREG [RIGHT 30] STOP]
IF :COM = "L [TRASEAZĂ.ȘI.ÎNREG [LEFT 30] STOP]
IF :COM = "C [ȘTERGE.ECRAN STOP]
IF :COM = "S [ANULEAZĂ.COM STOP]
IF :COM = "N [ÎNVATĂ STOP]
IF :COM = "D [NUME.FIG STOP]
IF :COM = "A [AFIS.PROG STOP]
IF :COM = "M [MENIU STOP]
END

```

Procedura EXECUTA.COMANDA are ca intrare numele comenzii ce trebuie executată (F, R, L, C, S, N, D, A, M), asociază și execută procedura corespunzătoare.

```

TO EXECUTĂ.COMENZI
EXECUTA.COMANDA READCHAR
EXECUTĂ.COMENZI
END

```

Procedura EXECUTA.COMENZI execută comanda ce a fost specificată de utilizator de la tastatură și repetă acest proces.

```

TO DESENEAZĂ
MENIU
ȘTERGE.ECRAN
EXECUTĂ.COMENZI
END

```

Procedura DESENEAZĂ constituie programul principal care implementează specificațiile precizate.

La început afișează meniul de comenzi utilizabile în program, inițializează lista de primitive pe baza căreia se va genera programul de desenare a figurii, șterge ecranul și cheamă rutina ce va executa comenzile specificate de utilizator.

12.3.7. Turnurile din Hanoi

Se consideră 3 tije denumite STÎNGA, MIJLOC, DREAPTA și n discuri de dimensiuni diferite, așezate pe tija din STÎNGA, în ordine descrescătoare a dimensiunilor lor, formînd un "turn" ca în fig. 12.6.

12. LOGO-T) ÎNICI, APLICAȚII



Fig. 12.6 Poziția inițială a discurilor

Se pune problema să mutăm cele n discuri din poziția inițială STÎNGA în poziția finală DREAPTA, respectând regulile:

- la fiecare mișcare se mută un singur disc;
- un disc nu poate fi așezat peste unul mai mic decât el;
- tija MIJLOC se poate folosi pentru mutări intermediare.

Rezolvare:

Pentru început să exemplificăm mutările pentru cazul $n=3$, în vederea stabilirii unui algoritm. Presupunem discurile notate cu 1, 2, 3 și vom considera că dimensiunile sînt puse în corespondență cu aceste numere, adică discul 1 este cel mai mic iar discul 3 este cel mai mare.

Mutările ce trebuie efectuate sînt prezentate în fig. 12.7.



Fig. 12.7. Mutările efectuate pentru cazul $n=3$ discuri.

Se observă că pentru a putea muta discul cel mai mare (în cazul nostru 3) de pe tija inițială STÎNGA, pe tija finală DREAPTA este necesar să luăm toate discurile de deasupra lui (2 și 1) să și le mutăm pe tija MIJLOC, folosindu-ne de tija DREAPTA ca tijă intermediară. În figura 12.7 se ajunge în această situație prin efectuarea primelor trei mutări.

Problema mutării celor n discuri din poziția inițială STÎNGA în poziția finală DREAPTA poate fi redusă la problema mutării a $(n-1)$ discuri din poziția STÎNGA în poziția MIJLOC, folosind ca tijă intermediară DREAPTA.

Aceasta conduce la ideea realizării unei proceduri [mutare] ce va muta cele n discuri, care se va apela pe ea însăși ca să mute $n-1$ discuri, ș.a.m.d. Se observă că problema se poate rezolva prin recursivitate.

Mutarea efectivă a unui disc pe tija finală DREAPTA se poate efectua numai cînd numărul de discuri pe tija sursă (cu care lucrează procedura la un moment dat, în timpul apelării recursive) este 1.

Procedura de mutare efectivă a unui disc este:

```

TO mutadisc :s :d
(TYPE [Muta un disc din] " \ :s " \ in \ :d)
PRINT"
END

```

Pentru a indica efectuarea mutării se va afișa un mesaj:

Muta un disc din SURSA în DESTINAȚIE.

Operațiile ce trebuie efectuate de procedura ce realizează mutările [mutare] sînt următoarele:

— mută primele n-1 discuri din poziția STÎNGA în poziția MIJLOC; folosind tija DREAPTA ca tijă intermediară, prin apelare recursivă [continuă mutare];

— mută discul rămas pe tija STÎNGA în poziția finală DREAPTA, cînd apelarea procedurii se face pentru un disc, [muta disc];

În acest fel, unul din discuri ajunge în poziția finală și va trebui să ne ocupăm de celelalte n-1, pentru a le transfera de pe tija MIJLOC pe tija DREAPTA, prin intermediul tije STÎNGA (situație obținută în urma mutării 4 în exemplul considerat).

Procesul continuă ca și cum ne-am afla în faza inițială numai că sînt mai puține discuri (n-1) iar tijele sînt MIJLOC, STÎNGA, DREAPTA.

Procedura de efectuare mutare este:

```

TO mutare :n :s :m :d
IF :n=1 [mutadisc :s :d STOP] [continuă mutare]
END

```

unde cei patru parametri au următoarea semnificație:

:n — numărul de discuri de pe tija curentă;
:s — tija STÎNGĂ;
:m — tija MIJLOC;
:d — tija DREAPTA;

Dacă numărul de discuri pe tija curentă este 1 atunci mută discul respectiv în poziția finală, prin execuția procedurii [mutadisc], iar în caz contrar continuă cu mutarea a n-1 discuri prin apelarea procedurii [continuă mutare].

Procedura [continuă mutare] realizează operațiile:

— mută cele n-1 discuri de pe tija STÎNGA pe tija MIJLOC folosind ca tijă intermediară tija DREAPTA;

— mută efectiv discul din STÎNGA în DREAPTA;

— mută cele n-1 discuri, din cele rămase, de pe tija MIJLOC pe tija DREAPTA folosind, ca tijă intermediară, tija STÎNGA.

Procedura continuă mutare este:

```

TO continua_mutare
mutare :n-1 :s :d :m
mutadisc :s :d
mutare :n-1 :m :d
END

```

Numărul de mutări necesar pentru transferul celor n discuri de pe tija STÎNGA pe tija DREAPTA folosind, ca tijă intermediară tija MIJLOC, este 2ⁿ-1.

Programul final care interacționează cu utilizatorul în vederea stabilirii numărului de discuri și contorizează mutările efectuate este:

```

TO MUTADISC :s :d
MAKE "nr_mutare :nr_mutare +1
(TYPE :nr_mutare" [Muta din] " \ :s " \ in \ :d)
PRINT"
END

```

```

TO CONTINUA_MUTARE
MUTARE :n-1 :s :d :m
MUTADISC :s :d
MUTARE :n-1 :m :s :d
END

```

```

TO MUTARE :n :s :m :d
IF :n=1 (mutadisc :s :d stop) (continua_mutare)
END

```

```

TO HANOI
TYPE (Introduceți număr discuri :)
MAKE "nr_discuri READCHAR
TYPE :nr_discuri
MAKE "nr_mutare 0
PRINT"
MUTARE :nr_discuri "stnga "mijloc "dreapta
END

```

Exemplu de utilizare:

2 hanoi

Introduceți număr discuri :3

1 Mută din stnga în dreapta
 2 Mută din stnga în mijloc
 3 Mută din dreapta în mijloc

4 Mută din stnga în dreapta
 5 Mută din mijloc în stnga
 6 Mută din mijloc în dreapta
 7 Mută din stnga în dreapta

2.4. Mesaje de eroare generate de LOGO

Varianța în limba engleză

Not enough input to ...
 I don't know how to ...
 You don't say what to do with ...
 ... does not output to ...
 ... is used by Logo
 ... is already defined
 ... is not true or false
 ... is not word
 Too many inside parentheses
 ... open file problem
 ... file not found
 Bad file name
 'ou're at toplevel
 STOPPED
 Turtle out of field
 Not enough space to proceed
 ... doesn't like
 ... has no value
 ... is a primitive
 Not enough items in ...
 Overflow
 ... can't divide by zero
 ... number too big
 ... as input
 ... in

Obs. Traducerea în limba română este făcută în așa fel încît să se încadreze în același număr de caractere.

Varianța în limba română

Mai trebuie intrări ...
 Nu știu cum să fac ...
 Nu ai spus ce să fac cu ...
 ... nu are ieșiri în ...
 ... utilizat de Logo
 ... este deja definit
 ... nu este adevărat / fals
 ... nu este cuvînt
 Prea multe paranteze
 ... probleme la open
 ... fișier negăsit
 Nume greșit
 Sinteți în comandă
 ANULARE
 Penel afară ecran
 Nu e spațiu să prelucrez
 ... nu îi place
 ... nu are val.
 ... e o primitivă
 Mai puține elem. în ...
 Depășire
 ... nu pot împărți cu 0
 ... număr prea mare
 ... ca intrări
 ... în



● Cartea se poate subintitula „Totul despre... HC-85”, HC-85 fiind primul calculator personal românesc compatibil cu o familie larg răspândită pe plan internațional (Sinclair-Spectrum). Ea continuă ciclul început în 1985 cu „Totul despre... calculatorul personal AMIC” și continuat în 1989 cu „Totul despre... microprocesorul Z80” și „Totul despre... BASIC”, în redacția de informatică și tehnică de calcul a Editurii Tehnice.

● Este scrisă de specialiști din cercetare-proiectare-fabricație-învățământ, de la catedra de calculatoare a Institutului Politehnic București – Facultatea de Automatică (conceptori-proiectanți ai lui HC-85, în frunte cu Prof. dr. ing. Adrian Petrescu, coordonatorul colectivului), respectiv de la Fabrica de Calculatoare Electronice (care a realizat calculatorul HC-85), de la Institutul de Tehnică de Calcul (ITC), Institutul de Cercetări pentru Informatică (ICI), Liceul Dimitrie Cantemir, ș. a. (reprezenții din cei mai autorizați ai realizatorilor de software, inclusiv de programe de aplicații pentru HC-85, ai învățământului elementar și mediu și centrelor de instruire pentru diverse categorii de utilizatori de calculatoare, cum și ai editorilor de specialitate).



● Este alcătuită din două volume (XII părți, 25 capitole), o parte a tirajului fiind însoțită de 3 casete magnetice încărcate cu programe (interpretor LOGO și exemple LOGO, interpretor BETA BASIC și lecții BASIC, lecții de structură și utilizare a lui HC-85, programe de aplicații în BASIC etc.), ce acționează / utilizează calculatoare HC-85 sau compatibile.

● După un studiu introductiv al Acad. prof. Nicolae Teodorescu și o secțiune intitulată PRO-LOG-DIALOG-EPILOG (ce continuă la începutul și sfârșitul volumului 2) primul volum cuprinde: I. Calculatoare, microcalculatoare și calculatoare personale în țara noastră și pe plan mondial. II. Calculatoare numerice. Realizare fizică – baze aritmetice și logice. III. Calculatorul personal HC-85. Structură, componente, operare, programare. IV. Programarea în BASIC pe HC-85. V. Programarea în LOGO pe HC-85, volumul asigurând cunoștințe fundamentale și aplicative profunde, ultima parte fiind și o premieră în literatura românească.