PROGRAMIRANJE ZA POČETAK

PRIREDIO | BORISLAV B. MITROVIĆ

MINI BASIC

MAŠINAC ZA ZX SPECTRUM I COMMODORE

ALGORITMI

O DRUGIM JEZICIMA

PROGRAMI ZA COMMODORE I ZX SPECTRUM

KORISNE KNJIGE • BEOGRAD

UPOREDNI PREGLED NAREDBI: APPLE, BBC, TRS 80, SPECTRUM, COMMODORE, GALAKSIJA

PROGRAMIRANJE ZA POČETAK

PROGRAMIRANJE ZA POČETAK

PRIREDILI

BORISLAV B. MITROVIĆ
STANKO DELIĆ
OGNJEN DELIĆ
BOŽUR MILOŠEVIĆ

MINI BASIC

MAŠINAC ZA ZX SPECTRUM I COMMODORE

ALGORITMI

O DRUGIM JEZICIMA

PROGRAMI ZA COMMODORE I ZX SPECTRUM

KORISNE KNJIGE
BEOGRAD

KORISNE KNJIGE
Ediciju ustanovio Borislav B. Mitrović

Ostala izdanja:
LEKAR U VAŠIM PRSTIMA — priručnik za akupresuru

Autorsko izdanje
PROGRAMIRANJE ZA POČETAK — PZP

Priredio:
Borislav B. Mitrović

Priređivači-saradnici:
Stanko Delić, Ognjen Delić i Božur Milošević

Obrada materijala i programa:
Borivoje Perzić, Voja Marić

Stručno mišljenje:
Dr Vladimir Štambuk, profesor kibernetike
na Fakultetu političkih nauka u Beogradu

Naslovna strana:
Rajka Milović

Korektura:
Radojka Isaković

Štampa:
SLAVIJA-PRES, Novi Sad, Trg Toze Markovića 5
GRO »Kultura«, OOUR »Radiša Timotić«, Beograd, Jakšićeva 9

Prvo izdanje u maju 1985, u 3.000 primeraka

Adresa izdavača: Ohridska 11, Beograd

Pisma i porudžbine slati na:
KORISNE KNJIGE (za PZP)
poštanski fah 13, 11050 Beograd

SADRŽAJ

I/ PRE PROGRAMIRANJA

Pre nešto više od godinu dana, u leto 1983, Branislav Ivanović, Stanko Delić i ja odlučili smo da priredimo knjigu o ličnim računarima, namenjenu svim zainteresovanima — koji nemaju predznanja iz informatike. U oktobru iste godine izdali smo, kao autorsko izdanje, LIČNI KOMPJUTER — Zašto čekati budućnost!

Bila je to prva domaća knjiga o ličnim računarima, o tome za šta mogu da se koriste, kako rade i kakvih ima na tržištu. Do danas prodato je pet izdanja u ukupnom tiražu od 20.000 primeraka, pola od toga na srpskohrvatskom, a pola na slovenačkom jeziku.

U međuvremenu, nijedna oblast ljudske aktivnosti nije na ovim geografskim prostorima doživela tako buran i dinamičan rast interesovanja kakav su doživeli računari. Novine, radio i televizija posvećuju im prostor kakav nemaju ni objektivno mnogo važniji elementi života. Četiri specijalizovana časopisa obrađuju oblasti kućnih računara, nekoliko preduzeća sklapa (basnoslovno skupe) mikroračunare, a država je pokazala da ne može da se odupre široko stvorenom javnom mnenju i ukinula je carinske barijere za uvoz mikroračunara. Sredstva javnog informisanja puna su kurseva BASIC-a, a pištanje mašinskih programa iz radio-prijemnika odjekuje našim domovima na sablazan starije generacije.

Neminovno je bilo da se u sklopu svega toga jave i glasovi koji se pitaju: Da li nam to sve treba; zar neće računari otuđiti one koji ih koriste; nije li to samo još jedna skupa moda od koje nema velike koristi; treba to ostaviti profesionalcima... Ako izostavimo vreme koje će, naravno, pokazati sve efekte — evidentno je, barem što se tiče mladih, da učenje komuniciranja sa računarom (samo još jednom u nizu čovekovih alatki) mora da bude od koristi generacijama koje će raditi u 21. veku. Logičko zaključivanje umesto gomilanja enciklopedijskih znanja trebalo bi da bude najvažnija posledica igranja sa elektronskim računaljkama. Takođe je evidentno da je ovo što se događa sa ličnim računarima i jedna velika moda. Međutim, ako uporedimo modu ličnih računara sa modernim odevanjem ili sa video-rikorderima, onda je ovo jedna od jevtinijih moda, a jedna od najkorisnijih, sigurno. Jer, iako danas kućni računari predstavljaju status-simbol, vrlo brzo pomodarci će se zasititi, a neko drugi će im (tim kućnim računarima) naći korisnu primenu — za učenje, igru ili posao.

Što se profesionalaca tiče, posebno u relativno primitivnoj kulturnoj sredini kakva je naša, razumljiva je njihova nelagodnost što im se laici »mešaju u posao«. Objektivno posmatrano, hiljade mikroračunara u kućama, advokatskim kancelarijama, projektnim biroima, zanatskim radnjama i na sličnim mestima predstavljaju demokratizaciju informatike. Do danas veliki računari su bili u službi države, institucije ili preduzeća, a interes pojedinca, najblaže rečeno, bio je u drugom planu. Možda je to izrazito individualistički stav, no moje je mišljenje da svako od nas ima i svoj život, svoju istoriju, svoje posebne interese i interesovanja, svoje zbirke i svoje datoteke. Za taj deo nas bitno je da imamo novu alatku, uređaj koji može da nas oslobodi mnogih rutinskih poslova, da bi nam ostalo više vremena za kreativnije aktivnosti — u poslu, u profesionalnom usavršavanju, u zabavi. Uz to, otvara se mogućnost korišćenja masovnih datoteka u nacionalnim bibliotekama, naučnim i drugim institucijama u svetu. Pristupi javnim bankama podataka u našoj zemlji će biti realnost za godinu, dve. Dakle, nezamislivo se šire mogućnosti sticanja novih znanja, istraživanja i specijalizovanja u mnoštvu oblasti.

Upravo su profesionalni informatičari ti koji moraju da stvore podlogu za one koji ne žele da osvajaju novu profesiju nego da računare (lične i društvene) koriste da bi bolje obavljali svoj posao.

Opšti trend u svetu je da se korišćenje računara približi laicima. U pogledu samih uređaja, prvi ozbiljan korak napravila je kuća Apple svojim proizvodima LISA i MACINTOSH koji korisniku omogućuju da bez ikakvih velikih predznanja, pomoću vizuelnih simbola, radi i sa najsloženijim programima. Drugi proizvođači najavljuju iste pro-

izvode. »Miš« osvaja svet. U programskom smislu najavljuju se velike novine sa istom namenom. Upravo se najavljuje nova generacija komercijalnih programa za klasične lične računare u kojima je posebno obrađeno »programsko okruženje« tojset, korišćenjem simbola pojednostavljeno je rukovanje programima.

PZP (Programiranje Za Početak) je slično zamišljeno. To nije udžbenik za profesiju programera. Potrudili smo se da vam na jednom mestu prikažemo najvažnije elemente programiranja mikro-računara. Pri tome nije bio cilj da se obučite u programiranju da biste sami rešavali sve probleme koje ste zamislili. Naprotiv, najvažnije će biti ako smo uspeli dalje u naporu da prikažemo kako radi računar, da biste bolje koristili programe koje pripremaju profesionalci, da biste bili u stanju da im date preciznije postavljene zahteve. Ako se neko od čitalaca i sam upusti u ozbiljnije programerske zahvate, počev uz PZP, pa nastavljajući saznavanje iz nekih većih knjiga, tim bolje. Ako je uopšte potreban neki savet pre čitanja, onda je to: prođite kroz PZP prvo površno, da vidite o čemu je reč. Potom uzmite papir i olovku i krenite ponovo kroz poglavlja O RAČUNARSKOJ LOGICI i ISPECI PA ... PROGRAMIRAJ. Tu je suština, sve ostalo je stvar srednjoškolskog znanja iz matematike i rutine koju ćete postići jedino eksperimentisanjem na papiru i sa računarom.

U Beogradu, marta 1985. Borislav B. Mitrović

PROGRAMIRANJE ZA POČETAK posvećeno je Mirjani Mitrović, matematičarki koja je uvek uspevala da ulije ljubav prema matematici (čak i svom sinu!).

Posebnu zahvalnost dugujemo Prvoslavu Milinkoviću za razradu ideje za PZP, Borivoju Perziću za bravure sa programima kojima je rukopis obrađen i otkucan i Voji Mariću koji je obradio programe za ZX Spectrum i Commodore 64.

II/ O RAČUNARU I LOGICI

Upoređivanje i brojanje — Od abakusa do procesora — Binarni brojevi — Bit, bajt — Sistem — Principi logike

Baratanje brojevima — računanje je jedna od najvažnijih aktivnosti i preokupacija čoveka, od nastanka prvih civilizacija. Piramide i rezultati astronomskih proračuna stari nekoliko hiljada godina samo su simbol velikog duhovnog i materijalnog kulturnog nasleđa zasnovanog na brojevima.

Najopštije rečeno, od tih vremena do danas čovek koristi dva osnovna principa u računanju: upoređivanje i brojanje. Upoređivanje (princip analogije) sastoji se u tome da posmatranjem jedne situacije donosimo zaključke o drugoj. Posredno, dakle. Jedan od svakodnevnih primera analognog sistema je termometar na kome povećanje ili sniženje temperature merimo povećanjem ili smanjenjem dužine (visine) živinog ili alkoholnog stuba. Digitalni sistemi (od latinske reči: digitalis — od prsta, prstni) zasnovani su na brojanju, znači neposredno se dolazi do rezultata. Pošto se brojalo na prste nije ni čudo što većina od nas misli da postoje samo desetni ili decimalni brojevi kod kojih se sve izražava sa deset cifara: 0, 1, 2, 3, 4, 5, 6, 7, 8 i 9. Međutim postoje sistemi brojeva koji su zasnovani na 12, na 16, čak i samo na 2 broja. Svi oni omogućuju razna računanja, a koriste se zavisno od potrebe. U PZP će dosta biti reči o brojnom sistemu sa 16 cifara — heksadecimalnom (skraćeno: heksa) i brojnom sistemu sa dve cifre

— binarnom. Binarni sistem je, reklo bi se, važniji. Na njemu je zasnovan rad svih današnjih digitalnih računara. Binarni (dvocifreni) sistem ima dva simbola: 0 i 1. Ta dva simbola ne koriste se samo za izražavanje brojeva, oni su prevashodno simboli dva stanja: uključeno i isključeno; da i ne; ima i nema, itd. To je, pokazalo se, tehnološki najpogodnije za rad digitalnih računara, zasnovanih na elektrici i elektronici. Postoji stanje kada ima napona (1) i kada nema napona (0) u provodniku. Tako je danas i na to ćemo se još vratiti, ali da pogledamo šta je prethodilo današnjem digitalnom računaru.

Prvi digitalni mehanizam, naprava zvana abakus koristi se već više od pet hiljada godina. Principijelno je objašnjena u LIČNOM KOMPJUTERU. Abakus ili naprave zasnovane na istom principu — kao što je školska računaljka — nisu imale premca sve do 17. veka, a čoveku je bilo potrebno da sve brže dobije rezultat neke računice. Matematičari Blez Paskal (njemu u čast jedan viši programski jezik nosi ime PASKAL) i, nešto kasnije Vilhelm Lajbnic dali su najozbiljnije doprinose. Paskalov patent mašine omogućavao je sabiranje i oduzimanje, a Lajbnic je dodao množenje, deljenje, čak i vađenje kvadratnog korena. Onda se čitav vek i po ništa značajnije nije dogodilo, dok Čarls Bebidž nije zaslužio današnju titulu »otac modernih računara« projektom svoje analitičke mašine koja je, nažalost, samo delimično konstruisana. Od kraja 19. veka i u prvoj polovini našeg veka razvoj je bio sve intenzivniji zahvaljujući električnoj energiji. Pred II svetski rat Konard Cuse uvodi u upotrebu binarni sistem i pokretni decimalni zarez, a razvija i prvi algoritamski programski jezik. Godine 1944. predstavljen je prvi univerzalni digitalni računar Mark 1, a dve godine kasnije proradio je prvi elektronski računar, sa 18.000 elektronskih cevi, težak 30 tona. Razvoj ratne tehnike zahtevao je minijaturizaciju, pa su smišljene zamene za cevi — tranzistori. Za njima su došla integralna kola i mikroprocesori kao osnova mikroračunara . . .

Kao što rekosmo, rad digitalnog računara zasnovan je na principu dva stanja: uključeno i isključeno. Idealno za binarni sistem brojeva. Broj cifara kojima se iskazuje jedna brojka je baza brojnog sistema. Desetni sistem, kao što znamo, ima bazu 10, jer se sa deset cifara: 0—9, predstavljaju svi brojevi. On se koristi u svakodnevnom životu i zato unutar svakog računara moraju desetni brojevi da se prevedu na binarne, jer to je računarova svakodnevica — baratanje binarnim brojevima.

Binarna cifra zove se BIT. Svaka informacija predstavlja se grupama bita. Pomoću različitih metoda kodiranja te grupe bita ne predstavljaju samo binarne brojeve nego i slova, decimalne brojeve

i grafičke simbole. Pravilnim izborom binarnih rasporeda i tehnika kodiranja, grupe bita, drugim rečima binarni brojevi, mogu se koristiti kao setovi različitih instrukcija za razna izračunavanja.

Grupa od 8 bita zove se BAJT i daje $2^8=256$ mogućnosti za kodiranje što je dovoljno za sva velika i mala slova latinice, cifre od 0 do 9, znake interpunkcije i znake operacija.

Računarski sistem

Da bi jedan računar bio to što jeste on mora imati jedinicu putem koje se u njega unose podaci. Takođe, na kraju rada mora da postoji i jedinica koja će prikazati rezultat — izlazna jedinica. Uneti podaci moraju da se obrade i za to služi centralna jedinica obrade (detaljnije objašnjena u VI poglavlju). Pošto jedinica obrade može u jednom trenutku da obrće mali broj podataka, svaki računar mora da ima memoriju u kojoj drži odložene podatke za rad i rezultate rada. Sve te jedinice predstavljaju opremu (eng. hardware) računara. Oprema bi ostala bez funkcije kad ne bi bilo programa (eng. software) koji određuju šta, kako i kada će nešto biti urađeno. Tako dolazimo do četiri osnovne funkcije svakog računara, bez obzira na veličinu:

a) ulazne i izlazne radnje ili operacije,
b) aritmetičke operacije,
c) upoređivanje i logičke operacije i
d) prenos podataka u, unutar i izvan jedinice za obradu.

Umetnost ili zanat programiranja sastoji se u tome da se problem koji je postavljen razloži u mnogo malih koraka, koristeći jednu ili više ovih osnovnih radnji.

Sve osnovne funkcije detaljnije su obrađene u narednim poglavljima, izuzev upoređivanja i logičkih operacija kojima ćemo posvetiti pažnju odmah, jer to i zaslužuju. Računar može uporediti dva broja i odlučiti koji je veći ili manji, ili su jednaki. To se odnosi i na upoređivanje slova abecede. Tehnika kodiranja je takva da su slova bliža početku abecede manja, pa je, na primer, »B« manje od »M«. Upoređivanje dve vrednosti je veoma važno za rad računara, jer ne postoji samo jedan mogući tok događaja i moguće su dve ili više narednih radnji. Zato je i računaru bilo potrebno usaditi osnovne logičke principe. Pogledajmo pojednostavljeno kako oni glase. Uzgred, ako vam se naredni redovi učine suvoparnim, vi ih preskočite i idite na naredno poglavlje, ali imajte na umu da nema ozbiljnijeg postavljanja bilo kog problema ako programeru nije jasno kako će to računar logički obrađivati.

U računarskoj logici koriste se binarne promenljive veličine i operacije logičkog značenja. Manipulacija binarnih informacija obav-

lja se takozvanim logičkim kolima. Logičko kolo ili prolaz je deo uređaja koji proizvodi binarne signale 1 ili 0, u zavisnosti od toga da li su zadati logički zahtevi ispunjeni ili ne. Uglavnom se ista logička kola koriste kod svih digitalnih računara. Svako logičko kolo ima svoj grafički simbol koji ovde nećemo upotrebljavati, nego ćemo kolo i ulazno-izlazni odnos promenljivih veličina prikazati logičkom tablicom. Osam osnovnih logičkih kola su: I, ILI, INVERZIJA (Negacija), BAFER, NE I, NITI, isključivo ILI, isključivo NITI. Svaki osnovni logički prolaz ima jednu ili dve ulazne binarne promenljive i jednu izlaznu binarnu promenljivu.

Logičko kolo I izvršava takvu logičku operaciju da izlazna promenljiva ima vrednost 1 ako obe ulazne imaju vrednost 1, a u svim ostalim slučajevima izlazna vrednost je 0:

	ulazne promenljive		izlazna promenljiva
	A	B	x
kolo: **I**	0	0	0
	0	1	0
	1	0	0
	1	1	1

Možemo to opisati i rečima, koristeći 1 za označavanje da je nešto istina, a 0 za označavanje da je nešto laž: Ako su A i B istina, onda je i X istinit. Ako je A ili B (ili, A i B) lažno i rezultat će biti lažan.

Logičko kolo ILI izvršava takvu logičku operaciju da je vrednost izlaza jednaka 0 samo ako su i ulazno A i ulazno B jednaki 0, a u svim ostalim slučajevima izlazna vrednost je 1:

	ulazne promenljive		izlazna promenljiva
	A	B	x
kolo: **ILI**	0	0	0
	0	1	1
	1	0	1
	1	1	1

Ako opet upotrebimo 1 za istinu, a 0 za laž, onda će x biti istinito u svim slučajevima izuzev kad su i A i B laž. Logičko ILI prikazuje se algebarskom funkcijom sabiranja $x=A+B$. Logičko kolo ILI može imati i više od dve promenljive i tada je izlazna vrednost jednaka 1 ako je bilo koja od ulaznih vrednosti jednaka 1.

Logički prolaz inverzije izvrće logički smisao ulaznog binarnog signala. On izvršava funkciju negacije — NE ili komplementarnu funkciju. To znači da će u ovom kolu biti:

	ulazna promenljiva A	izlazna promenljiva x
kolo: **INVERZIJA**	0	1
	1	0

BAFER ne izvršava nikakvu posebnu logičku funkciju, jer je binarna vrednost izlazne promenljive uvek jednaka binarnoj vrednosti ulazne promenljive. BAFER se, uglavnom, koristi za pojačavanje signala:

	ulazna promenljiva A	izlazna promenljiva x
kolo: **BAFER**	0	0
	1	1

Logičko NE I, često zvano NI, komplementarno je logičkom I. To znači da je izlazna binarna vrednost uvek 1 ako je jedna ili obe binarne vrednosti jednaka 0:

	ulazne promenljive A	B	izlazna promenljiva x
	0	0	1
kolo: **NE I**	0	1	1
	1	0	1
	1	1	0

Logički prolaz NITI predstavlja komplement logičkog ILI. To znači da je izlazna binarna vrednost uvek 0 ako su ulazne A i/ili B jednake 1:

	ulazne promenljive A	B	izlazna promenljiva x
	0	0	1
kolo: **NITI**	0	1	0
	1	0	0
	1	1	0

Kod logičkog prolaza isključivo ILI izlazna vrednost je 1 ako je binarna vrednost bilo koje ulazne promenljive jednaka 1, ali isključujući kombinaciju kada su obe ulazne promenljive 1:

	ulazne promenljive		izlazna promenljiva
	A	B	x
	0	0	0
kolo: **Isključivo ILI**	0	1	1
	1	0	1
	1	1	0

Isključivo NITI je komplementarna ili inverzna funkcija Isključivog ILI, dakle izlazna vrednost jednaka je 1 samo ako su obe ulazne promenljive međusobno jednake. Isključivo NITI zove se i funkcija Ekvivalencije:

	ulazne promenljive		izlazna promenljiva
	A	B	x
	0	0	1
kolo: **Isključivo NITI**	0	1	0
	1	0	0
	1	1	1

Ako ste prebrodili logička kola, pogledajmo jedan primer iz života u kojem računar primenjuje logičke funkcije. Semaforom na raskrsnici upravlja računar. Ispred ulaza u raskrsnicu, na glavnoj ulici, nalaze se u asfaltnom zastoru senzori koji računaru javljaju koliko kola uđe sa jedne strane u raskrsnicu tokom jedne sekvence zelenog svetla. Pošto smo mi poznati kao lenji vozači, recimo da je programski predviđeno da normalno u toku jedne sekvence prođu tri vozila. Međutim, ako senzor javi da je prošlo više vozila, to znači da je saobraćaj gušći i da treba menjati vremensku sekvencu, tojest dati više vremena za zeleno svetlo na glavnoj ulici. Opisno, taj program logički to izvodi na sledeći način:

AKO je broj kola veći od 3 ONDA pređi na duže zeleno na glavnoj ulici

AKO broj kola nije veći od 3 ONDA ostani na istoj sekvenci.

Recimo da je programski predviđeno logičko kolo INVERZIJE. Programski brojač dobija od senzora podatak o prolasku svakog vozila. U određenim vremenskim periodima program proverava da li je stanje brojača veće od 3. Ako jeste, binarna promenljiva u ulazu dobija vrednost 1, a izlazna promenljiva dobija vrednost 0 što dalje vodi na nalog promene sekvence zelenog svetla. Ako, pak, stanje brojača bude između 0 i 3 u desetnim brojevima ulazna promenljiva imaće binarnu vrednost 0, pa će se na izlazu logičke operacije dobiti binarna vrednost 1, a to će računar protumačiti kao nalog da ostane na istom režimu rada.

III/ ISPECI, PA... PROGRAMIRAJ

Kako reći računaru šta da radi — algoritam

Programiranje je proces koji se zasniva na nizu instrukcija prirеđenih za određeni računar koji, izvršavajući te instrukcije, obavlja neku aktivnost i daje traženi rezultat. Ta aktivnost zavisi od zahteva i može biti sve ono što je moguće izraziti matematičkim i logičkim izrazima. Međutim, kako računar ne misli i ne može da donese ni jednu neplaniranu odluku, svaki korak u rešavanju nekog zadatka mora biti uključen u program. Problem koji se postavi pred računar (ustvari, problem koji sebi postavite) mora biti opisan na tačno utvrđen način i izražen »rečima« koje računar može da »razume«. Ako je to problem koji zahteva intuiciju ili pogađanje, ili je tako loše definisan da ga je teško iskazati rečima — računar ne može da ga savlada. Zato je potrebno dosta razmišljanja u pripremi, da se problem precizno postavi do detalja, a instrukcije koje mu se daju moraju biti razrađene do najsitnijeg koraka.

Nikada ne zaboravite činjenicu da se računar koristi da izvede rešenje problema onako kako je postavljen. Računar ne rešava problem — vi morate da ga rešite!

Razvoj ovog posla ima faze koje se ne mogu zameniti ni isključiti:

1. ANALIZIRANJE PROBLEMA
   Proučavate problem i definišete ga, razvijajući tako metod kojim će se doći do rešenja
2. RAZVIJANJE ALGORITMA
   Rešenje problema se predstavlja algoritmom ili prikazom toka rešavanja u grafičkom obliku
3. PISANJE PROGRAMA
   Svaki korak u rešavanju problema se pretvara u instrukcije računaru uz upotrebu nekog programskog jezika kao što je BASIC
4. IZVRŠAVANJE PROGRAMA
   Niz instrukcija (program) se unosi u računar i izdaje se naredba da se program izvrši
5. PROVERA PROGRAMA
   Tokom izvršavanja i na osnovu rezultata proverava se da li program radi ispravno
6. DOKUMENTOVANJE PROGRAMA
   Zabeleške, liste programa, algoritam i slični materijali se odlažu zajedno da bi poslužili nekom drugom ko koristi program ili vama ako zaželite da ga promenite na neki način.

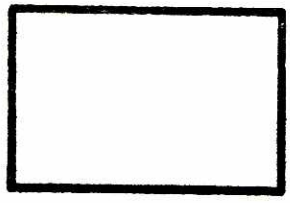
Greška u bilo kojem koraku, izuzev prvog, može se ispraviti lakše nego ako nastane na samom početku. Zato pažljivo analizirajte problem, ali, prethodno, postavite sebi neka pitanja. Na primer: Znam li da rešim problem uz pomoć računara; može li se on uopšte rešiti na taj način; može li moj računar da savlada taj problem (s obzirom na programski jezik i memorijski kapacitet); šta mu treba dati kao ulazne podatke, a šta mi treba da dobijemo na izlazu... Kad odgovorite sebi na takva pitanja bićete mnogo bliži načinu rešavanja problema.

Analiza problema je razumevanje zadatka i rezultat tog procesa je formalna i logička postavka problema na način na koji računar može da ga reši. Analiza problema je jedan individualni proces za koji ne postoje obrasci. Ipak, red u razmišljanju bi mogao da bude sledeći: Prvo, određuju se svi elementi problema i preciziraju izlazni i ulazni podaci; drugo, nužno je da sve elemente pretvorite u kvantifikovane izraze kao »1000 dinara« ili »18 učenika«; treće, odredite koja sve izračunavanja i logičke operacije moraju da se izvedu da donesu željene rezultate, pa te operacije izrazite kao određene korake; četvrto, poređajte te operacije redom kojim treba da budu izvođene i, na kraju, uspostavite odnose među elementima i faktorima koji ulaze u način rešavanja problema.

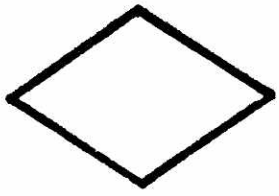
Pošto ste sve to uradili, cela stvar mora da vam bude još jasnija. Da biste to postigli morate sve elemente i njihove međusobne odnose da izrazite u vidu koraka koje računar može da izvede. Taj niz kora-

ka za rešavanje problema zove se ALGORITAM. Pojednostavljeno rečeno, algoritam je recept ili lista instrukcija za obavljanje nečega. Preciznije, to je kompletna procedura ili plan za rešavanje problema. Algoritam za simulaciju leta aviona ima nekoliko hiljada koraka. Vi počnite sa deset i budite zadovoljni (za početak!).

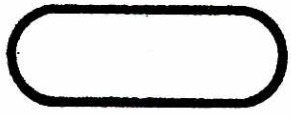
Algoritam se može izraziti na više načina. Najprostiji je da napravite listu svih koraka koji moraju biti učinjeni, redom kako slede. Grafikon toka je drugi način. On se koristi uz pomoć simbola koji su standardizovani i koriste ih programeri širom sveta. Svaki simbol označuje određenu operaciju, a međusobno su povezani linijama sa strelicom koja označuje mogući tok.



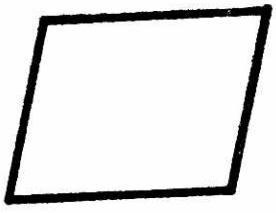
SIMBOL PROCESA — grupa naredbi koja proizvodi informaciju. Često označuje jedan korak. Koristi se za obeležavanje neke obrade.



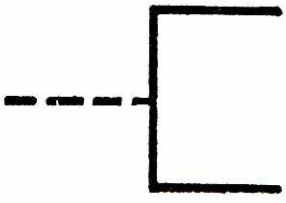
SIMBOL ODLUKE — obeležava tačku u proceduri na kojoj se donosi neka odluka.



TERMINAL SIMBOL — služi za obeležavanje početka i kraja procedure.



SIMBOL ULAZA/IZLAZA — obeležava tačku u proceduri na kojoj se nešto unosi u uređaj ili on daje neki izlazni podatak.



SIMBOL NAPOMENE — sadrži opaske ili objašnjenja o proceduri.

Kod crtanja grafikona toka valja primeniti neka pravila:

1. koristite samo standardne simbole,

2. razvijte tok tako da ide odozgo na dole i s leva na desno, kad god je to moguće,

3. neka grafikon bude jasan, čitljiv i prost. Ostavite dosta prostora između simbola. Ako je rešenje dugo i složeno, razbijte ga na nekoliko grafikona,

4. pišite jednostavne, opisne poruke unutar simbola,

5. upotrebljavajte samo prave uspravne i vodoravne linije. Ne koristite dijagonalne linije (izuzev za napomene) kao ni krive linije,

6. u jedan simbol može se usmeriti samo po jedna linija.

Na kraju ovog kratkog objašnjenja o pravilima programiranja evo jednog primera algoritma u obliku grafikona toka. Problem se sastoji u tome da se odredi koji od tri data broja je najveći (strana 20).

SISTEMSKE NAREDBE

Pre nego što pređemo na objašnjavanje samog BASIC-a nužno je da se upoznamo sa nekim naredbama koje će nam služiti od trenutka kada uključimo napajanje uređaja.

Sistemske naredbe daju instrukcije računaru šta da radi sa programom. Dakle, to nisu naredbe koje se koriste za rešavanje nekog problema. One se razlikuju od računara do računara, a mi ćemo pomenuti one koje se najčešće sreću. Pošto je reč o programima u BASIC-u, kad unesete svoj program on se smešta u radni, privremeni deo memorije. U taj deo memorije uvek se smešta svaki program koji čeka na izvršavanje. Samo po jedan program, bez obzira na dužinu, može u jednom trenutku zauzimati taj prostor. Naravno ako više programa povežete tako da čine celinu, računar će ih tretirati kao jedan program.

Sve sistemske naredbe aktiviraju se kad pritisnete dirku RETURN, ili, kod nekih računara ENTER. Sistemske naredbe nemaju broj programskog reda.

LIST

Ova sistemska naredba (od reči: listaj) donosi na ekran listu programa koji je u radnom delu memorije. To je neophodno uvek po unošenju programa i tokom unošenja da biste proverili da li ima greške u kucanju. Programski red koji treba izbrisati, jer je pogrešan, briše se na taj način što ukucate broj tog reda i potom kucate RETURN (ili ENTER).

Kad hoćete da unesete programski red među već postojeće, kucate broj reda koji je između postojećih i potom sadržaj reda. Kad je red gotov, kucate RETURN i red se smešta na svoje mesto.

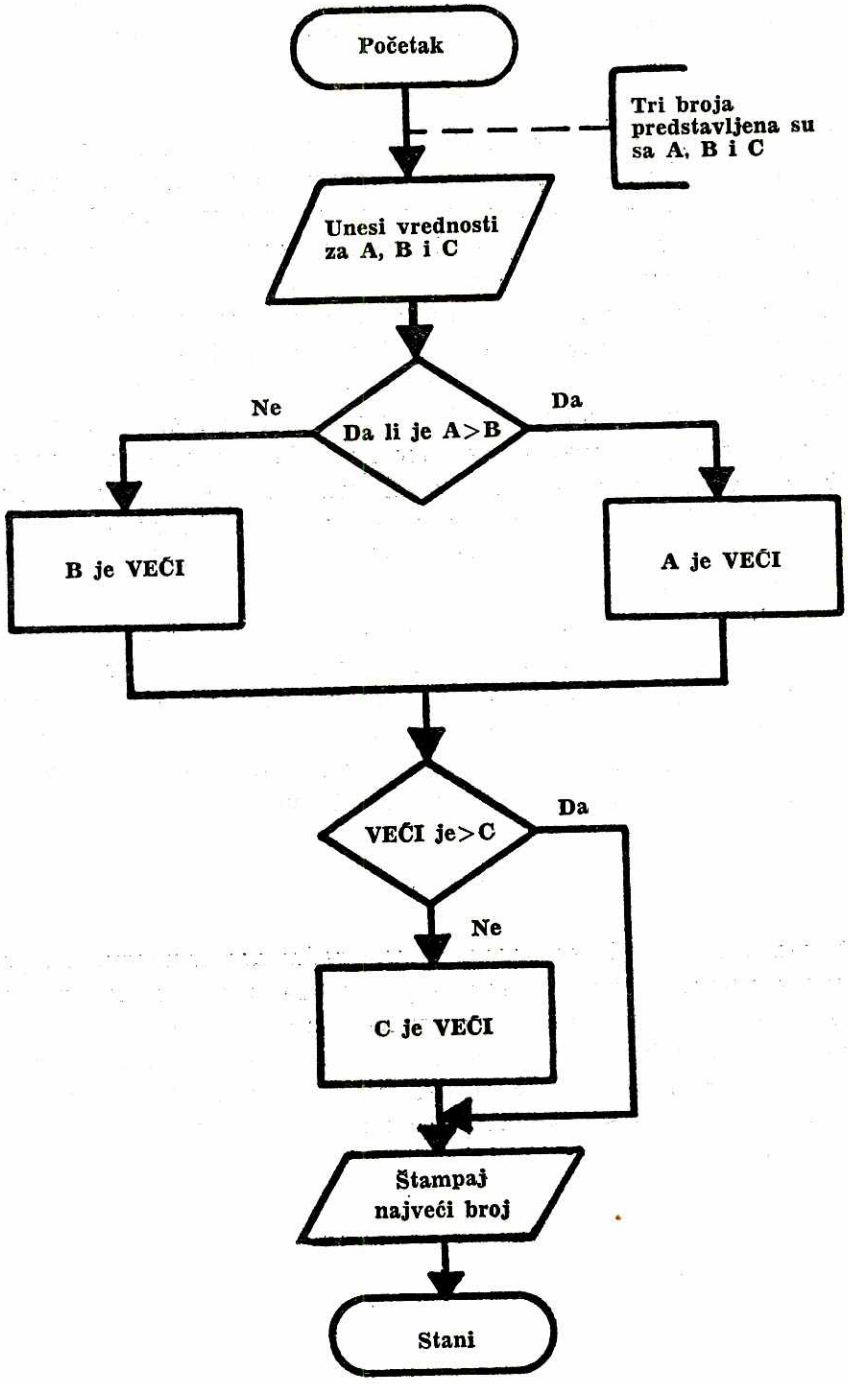
RUN

Pošto ste ispravili greške u programu (one se popularno nazivaju bugs ili buve) rešili ste da startujete program. Za to vam služi naredba RUN. Ona kaže: Računaru, molim te, izvrši moj program! Na osnovu ove naredbe program se prevodi na instrukcije koje prihvata mikroprocesor koji izvršava program. RUN dolazi od reči: trči! Ukoliko u programu ima pravopisnih greški (u odnosu na programski jezik koji ste koristili) izvršavanje će se prekinuti i na ekranu će se pojaviti standardno objašnjenje greške sa napomenom u kojem programskom redu se javila. Uz pomoć LIST stižete do tog mesta i pokušate da utvrdite kakva je greška. Ne zaboravite da obaveštenje o greški daje broj reda u kojem se greška manifestuje, a ona ne mora da nastane u tom redu. Na primer, ako se koristi neka funkcija (o tome će kasnije biti više reči) onda tražite red u kojem je ta funkcija definisana, jer se greška tu krije.

Poseban oblik naredbe RUN je RUN br, gde je br broj reda od kojeg želite da počne izvršavanje. Ta naredba se koristi zbog toga što početak programa ne mora da bude početak izvršavanja, pa se naznačivanjem početka preskaču nepotrebni redovi. Ili, kad testirate pojedine delove programa, vi startujete samo te delove da biste videli kako funkcionišu (ako je logika tako postavljena).

NEW

Naredba NEW (od reči: novo) »čisti« radni deo memorije od postojećeg programa i time se stvara prostor za unošenje novog programa. Ona se odnosi samo na programe u BASIC-u, jer ne uništava programe u mašinskom jeziku. Za njih se kod većine mikroračunara koristi sistemska naredba RESET (od izraza: postavi ponovo) čime se briše sadržaj cele RAM memorije i sistem dovodi u stanje kao da je tek uključen.



IV/ BASIC, najpopularniji jezik

Najpopularniji programski jezik

BASIC je najpopularniji programski jezik ne samo zbog toga što se najviše koristi u svetu mikro računara već i zato što ga proizvođači velikih računarskih sistema skoro uvek stavljaju na raspolaganje korisnicima, a naročito zbog toga što ga je veoma lako naučiti. Njegovo ime nastalo je od početnih slova izraza »Begginers All-purpose Symbolic Instruction Code« što bi se moglo prevesti kao »Zbirka simbolički izraženih instrukcija opšte namene za početnike«. BASIC je veoma jednostavan jezik, skromnog rečnika, zasnovan na engleskim rečima ali veoma pogodan za početnike i pri tom prilično univerzalan u smislu mogućnosti primene na veoma širok spektar problema. Danas u svetu postoji veoma mnogo različitih dijalekata BASIC-a koji su i pored razlika veoma slični, o čemu će kasnije biti reči.

BASIC koji će detaljnije biti izložen u ovoj glavi zove se »Minimalni BASIC« i sadržan je uglavnom u svim dijalektima tako da je i najpogodniji za početnike.

Umesto nabrajanja naredbi po abecednom redu i njihovog objašnjavanja, opredelili smo se da se BASIC uči logičkim redosledom od jednostavnijih ka komplikovanijim programima, kroz primere i koristeći znanja iz II poglavlja. Možda prvo čitanje teksta neće biti dovoljno za potpuno samostalno pisanje programa ali će razumevanje

datih primera i ponovno čitanje sigurno dati neophodnu osnovu za samostalno pisanje različitih programa.

Osnovni elementi

BASIC programima obrađuju se numerički i tekstualni podaci. Brojeve zovemo numeričkim konstantama a nizove znakova koji čine tekst zovemo string konstantama.
Brojevi tj. numeričke konstante mogu se predstaviti u sledećim formama:

5 —7 27.34 .890 —0.00123 +869423

Ako ispred broja ne stoji znak podrazumeva se da je broj pozitivan. Decimalni zarez piše se isključivo kao tačka. Koristi se i bilo koji unapred definisani format kao i eksponencijalni:

8.00000E+5	što znači	800000
4.33000E—4		0.000433
—7.034E—2		—0.07034
—6.300E+7		—63000000

Ekstremne vrednosti i tačnost brojeva koji se mogu čuvati u memoriji zavise od karakteristika računara, a najčešće su između 1E—38 tj. 10^{-38} i 1E+38 tj. 10^{38}, uz minimalnu tačnost od šest decimala.

String konstanta u BASIC-u jeste niz znakova između znakova navoda (" "). Za string se mogu koristiti svi raspoloživi znaci osim, naravno, navodnica koje označavaju početak i kraj string konstante. Čak i prazno polje je znak koji se može koristiti. Primeri string konstanti su:
„PROGRAMIRANJE ZA POČETAK"
"1. AVGUST 1984."
"RAČUNAR"
" ": Ovo je tzv. nulti string, tj, prazno mesto tj. blank.

Iz primera se može zaključiti da string konstante mogu sadržati i brojeve ali je važno naglasiti da se nikakve aritmetičke operacije ne mogu vršiti sa podacima koji su memorisani kao string. Postoje i dva izuzetka u BASIC-u kad su u pitanju string konstante: one se mogu unositi i bez znakova navoda i to u okviru naredbe DATA i kao odgovor na zahtev naredbe INPUT za unošenje stringa.

Osim brojeva i stringova u programima se često koriste i promenljive, koje opet delimo na numeričke i string promenljive. Numeričke ili brojne promenljive označuju se slovom ili kombinacijom

X M S2 C5

Promenljiva je ustvari oznaka određenog dela memorije računara u koji se smeštaju njene vrednosti koje se po želji mogu i menjati. Kod nekih računara sve numeričke promenljive automatski pre izvršenja programa dobiju vrednost 0, pa se preporučuje da u programima sve numeričke promenljive dobiju vrednost pre njihove upotrebe u programu.

String promenljive koriste se za identifikaciju delova memorije koji sadrže nizove slovnih znakova. Prosta string promenljiva obeležava se sa dva znaka: jednim slovom iza koga sledi dolarski znak:

A$ B$ Y$ M$

U Mini BASIC-u maksimalan broj znakova tj. dužina string promenljive je 18 znakova, dok se u pojedinim dijalektima taj broj povećava do 256 znakova.

U različitim vrstama programa često se koriste nizovi promenljivih. Naime često se koriste grupe podataka koje su u međusobnoj vezi. Pogodnije je koristiti jedno ime koje označava skup više podataka nego davati posebno ime svakom podatku. Niz je niz promenljivih istog imena u kome se svaki član identifikuje pomoću indeksa. Na primer:

A(5) D(N) G$(4) M$(M—2)

Prvi član niza obeležava se indeksom 0 ili 1, zavisno od računara. Važno je napomenuti da u istom programu obična promenljiva i niz-promenljiva ne mogu imati isto ime. Kroz naredbu DIM određuje se dužina niza. Gornja granica dužine niza zavisi od vrste računara. Ako se dužina niza ne definiše onda se niz-promenljivoj automatski dodeljuju indeksi od 0 do 10 ili do 1 do 10. Opšti izraz za niz je:

A(0) A(1) A(2) A(3) A(4) A(5) ... A(n)

BASIC omogućuje i upotrebu dvodimenzionalnih nizova. Određeni element dvodimenzionog niza označuje se sa dva indeksa odvojena zarezom, na primer:

A(5,2) X3(N—1, L) R$(14,9)

Prvi indeks označuje red, a drugi kolonu. Dakle, numerički niz A(5,2) ima 5 redova i dve kolone tj. 5 puta 2=10 elemenata.
Sledeći znaci koriste se za označavanje aritmetičkih operacija:

+	sabiranje	*	množenje
—	oduzimanje	/	deljenje
**	stepenovanje		

Kod nekih računara umesto dve zvezdice za stepenovanje se koristi strelica naviše. Starešinstvo računskih radnji isto je kao u matematici tj. u složenom aritmetičkom izrazu prvo se izvršava stepenovanje, zatim množenje i deljenje po redu po kome se u izrazu pojavljuju, gledajući sa leva na desno, i zatim sabiranje i oduzimanje, takođe po redu sa leva na desno. Zagrade se koriste kada treba odstupiti od iznetog prioriteta operacija i to tako što se prvo računaju izrazi u zagradama. Evo i primera:

Brojni izraz	prioritet	operacija	rezultat
A+B*C/D**E	1	D**E	R1
	2	B*C	R2
	3	R2/R1	R3
	4	A+R3	R4
(A+B)*C/D**E	1	A+B	S1
	2	D**E	S2
	3	S1*C	S3
	4	S3/S2	S4

Dve aritmetičke operacije ne mogu se pisati jedna do druge. Na primer A/—B korektno se piše A/(—B). Izostavljanje operatora ne znači ništa, pa ni množenje, dakle izraz A(B+1) mora se napisati kao: A*(B+1).

Nema aritmetičkih operacija sa stringovima.

U BASIC-u postoji i šest različitih relacija između dva numerička izraza ili dva string izraza:

Operator-relacija	Značenje
=	jednako
<>	različito
<	manje
>	veće
<=	manje ili jednako
>=	veće ili jednako

Ove relacije koriste se samo sa naredbom IF . . .THEN.

Takođe, koriste se i logički operatori (vidi: Logička kola, II poglavlje) AND, OR i NOT tj. davanje više mogućnosti u okviru jednog izraza. O svemu ovome biće detaljno reči tokom ovog poglavlja PZP.

Struktura

Program u BASIC-u sastoji se iz niza naredbi, instrukcija koje su pisane u vidu tvrdnji ili iskaza. Svaka tvrdnja ili naredba pisana je

u jednom redu, a svaki red počinje brojem naredbe. Broj reda je veoma važan jer služi kao oznaka na osnovu koje se uspostavlja redosled izvršavanja naredbi. Takođe, na osnovu broja reda vrlo lako se vrše izmene u programu. Program se izvršava redom kako idu brojevi tj. od manjih ka većim brojevima redova sve dok se ne dođe do naredbe u samom programu koja zahteva da se izvršavanje nastavi od određenog reda koji je već prošao ili se nalazi daleko napred. Tako izvršavanje teče dok se u programu ne javi naredba STOP ili se dođe do najvišeg broja reda u kome stoji iskaz END.

Svaki iskaz-tvrdnja u programu zauzima unapred utvrđeno mesto, zauzima određeni programski red. Prvo se piše broj reda. Iza njega sledi ključ — reč u BASIC-u koja određuje šta će biti izvršeno. Za njom sledi izraz koji sadrži numeričke i string konstante i promenljive povezane numeričkim i logičkim operatorima. Jedan iskaz zauzima jedan programski red. To znači da red može imati onoliko znakova koliko je propisao proizvođač računara. Razmaci (prazni prostori) mogu se unositi u programski red zavisno od toga kako je propisao proizvođač. Opšte je pravilo da se razmak ne pojavi u okviru broja reda, u okviru nekog broja ili usred ključne reči. S druge strane, preporučljivo je upotrebljavati razmak u pisanju naredbi i izraza, jer je dobrodošlo sve što doprinosi čitljivosti programa (bilo da čitate vi ili da ga »čita« mikroprocesor).

Broj reda mora biti ceo broj u rasponu od 1 do 9999. Kad pišete program nemojte obeležavati brojeve redom po 1. Svaki deo programa koji je nekakva celina odvojite za nekoliko desetina brojeva od sledećeg da biste mogli da umetnete nove naredbe. Teško je da se od prve dobije program koji radi. Mnogo je verovatnije da ćete nešto menjati i dodavati. Uobičajeno je da se za brojeve redova uzimaju cele desetice: na primer 100, 110, 120, 130 itd.

NAREDBE

Vreme je da vam predstavimo BASIC. Postoji nekoliko načina učenja. Opredelili smo se za prikazivanje naredbi logičkim redom uz objašnjavnje gramatičkih pravila na osnovu kojih se upotrebljavaju. Uz svaku naredbu dati su programi koji ilustruju samu naredbu i iskaze u kojima se koristi. Primeri su dati kao liste programa. IZLAZ (to je ono što se dobija izvršavanjem programa) je dat tačno onako kako će ga dati računar kad unesete dati primer radi vežbe. Liste programa i izlazi su dati velikim slovima, a ako program zahteva da vi unesete neki podatak, to je dato malim slovima.

Pošto je ovo mini-BASIC, osnova programskog jezika, to nisu date mnoge sistemske naredbe koje sadrži dijalekt koji koristi vaš

računar. Te naredbe doći će na red posle ovih koje slede, a imate ih u priručniku koji ste dobili uz uređaj. Uz to, ne zaboravite da se tastature računara razlikuju među sobom. Na primer, dirka na kojoj piše ENTER koristi se za unošenje programskog reda ili nekog podatka. Međutim to važi samo za neke računare dok se kod drugih isti efekat postiže dirkom RETURN. Dakle, ovo je osnova na koju će te vi nadgraditi dijalekt BASIC-a vašeg računara.

PRINT

```
Program: 10 PRINT "MOJ PRVI RED"
         20 PRINT 2+5
         30 PRINT "2+5"
         40 PRINT 3—12/3
         50 END

Izlaz:   MOJ PRVI RED                    Izvršeno redom: 10
         7                                               20
         2+5                                             30
         —1                                              40
```

PRINT daje izlaznu informaciju. PRINT se može upotrebiti za izvršavanje aritmetičkih operacija. Naredba je zamišljena tako da prenese (štampa) informaciju iz računara na ekran ili štampač. Njen najjednostavniji oblik je:

n PRINT a

gde je n broj programskog reda, dok je a neki numerički ili slovni izraz.

PRINT A ———— Štampa vrednost odloženu na lokaciji u memoriji koja je povezana sa promenljivom A.

PRINT B*7/3+28 ——— Štampa vrednost izraza koristeći postojeću vrednost B iz memorije.

PRINT "PROGRAM" —— Štampa niz slovnih znakova koji se nalaze između znakova navoda.

PRINT C$ ———— Štampa string odložen u memoriji na lokaciju povezanu sa string-promenljivom C$.

PRINT 313 ———— Štampa numeričku konstantu 313.

Zahtev PRINT (štampaj) promenljivu prouzrokuje da se sadržaj lokacije vezane za tu promenljivu „izvadi" iz memorije i pošalje na izlaz, a kad je u pitanju konstanta znaci se slažu za izlaz onim redom

koji su imali kad su uneti u memoriju. Kada je ono što treba štampati dato u formi numeričkog izraza, prvo se obave sve operacije da bi izlaz bio u obliku rezultata. PRINT uvek daje novi red u izlazu, izuzev ako je na kraju izraza zarez ili tačka i zarez.

Linija štampanja može se podeliti na zone prema potrebi i mogućnosti računara. To su zone iste širine i kod većine verzija ima ih pet. U principu izrazi odvojeni zarezom (,) biće štampani u odvojenim zonama, a izrazi odvojeni tačkom i zarezom(;) biće štampani jedan za drugim.

Izlazni niz kao rezultat numeričkog izraza uvek počinje razmakom ako je rezultat pozitivan ili znakom minus ako je negativan. Za njim sledi vrednost u vidu celog, decimalnog ili broja u eksponencijalnom obliku. Posle njega uvek dolazi jedan razmak — numerički rezultat uvek je odvojen od narednog izraza bez obzira na upotrebljeni separator (, ili ;).

Ako vam je potreban prazan red u izlazu potrebno je da u programu predvidite PRINT bez ikakvog dodatka. To će za mikroprocesor značiti: štampaj ništa i biće „odštampan" prazan red.

Kao što znate, svaki izlaz ima određeni broj redova (u horizontali) i određeni broj kolona (po vertikali). To znači da je ekran nevidljivo podeljen na jednake prostore. U svaki takav prostor staje po jedan znak. Programski se može predvideti da štampanje ne počne na početku reda i to se postiže funkcijom TAB (vidi na narednim stranama). Varijante upotrebe naredbe PRINT u sklopu sa zarezom i tačkom i zarezom date su u sledećem primeru, uz pretpostavku da je ekran podeljen na dve zone štampanja:

Program:
```
 10 LET A=8
 20 LET B=4
 30 LET C=20
 40 LET D=C/A*B
 50 PRINT A, B—C
 60 PRINT A; B; C; D
 70 PRINT
 80 PRINT C, C
 90 PRINT
100 PRINT C↑/A; "U ISTOM REDU";
110 LET X$="RINGERINGERAJA"
120 LET Y$=X$
130 PRINT
140 PRINT
150 PRINT Y$, X$
160 END
```

```
Izlaz:                                          izvršeno redom
8                    —16                                50
8 4 20 .625                                             60
                                                        70
20                   20                                 80
                                                        90
100 U ISTOM REDU                                       100
                                                       130
                                                       140
RINGERINGERAJA RINGERINGERAJA                          150
```

Izlaz na osnovu izvršenja programskih redova 50 i 60 pokazuje razliku štampanja kada su izrazi odvojeni zarezom, odnosno tačkom i zarezom. Redovi 70, 90, 130 i 140 proizvode prazan red u štampanju.

LET

Ova naredba dolazi od reči »neka, neka bude«. U prethodnom primeru smo videli da se ona koristi da bi dodelila, dala određenu vrednost nekoj promenljivoj tokom izvršenja programa. Oblici LET n LET x=numerički izraz ili

n LET x$=slovni izraz (string)

gde je n broj reda, a x je promenljiva. Promenljiva, pri tome, može biti potpisana, tj. indeksirana. Numerički izraz može biti različite složenosti, bitno je da rezultat operacija bude jedna vrednost koja se dodeljuje promenljivoj. Primeri su sledeći:

```
LET A=3 ————————— Dodeljuje numeričkoj promenljivoj A vrednost 3.
LET B$="PRIMER" — Dodeljuje string-promenljivoj B$ niz slova između navodnica.
LET D=M*H—0.6 —— Prvo se iz memorije uzimaju vrednosti M i H, izvrše se aritmetičke operacije i rezultat se dodeli promenljivoj D.
LET Q$(3)="SREDA" — Dodeljuje niz SREDA matričnom vektoru Q$ koji ima indeks 3.
LET X=E5 ————————— Dodeljuje X-u vrednost koja je u memoriji odložena u E5.
LET G=G+2 ————————— Uzima iz memorije postojeću vrednost G, uvećava je za 2 i dodeljuje je G, brišući prvobitnu vrednost G.
```

Poslednji primer pokazuje da se znak jednakosti posle naredbe LET može tumačiti ne samo kao »jednako je« nego i kao »zamenjuje se sa«.

U nekim verzijama BASIC-a ključ LET nije obavezan, pa isto značenje ima B=3 kao i LET B=3.

END

Naredba END (od reči: kraj) označava kraj programa. Ona zaključuje niz programskih naredbi koje sačinjavaju program i, kada je tako predviđeno, završava izvršenje programa. Ona ima oblik:

n END

gde je n najveći broj programskog reda upotrebljen u programu.

REM

Naredba REM (skraćenica od reči REMark: napomena) služi za povećanje čitljivosti programa za programera. Ova naredba se ne izvršava u istom smislu kao ostale, ona u listi programa stoji ispred napomena i mikroprocesor, u izvršavanju, preskače programski red na čijem početku je REM. Napomene su vrlo važne za svakog programera i nemojte se stideti da ih koristite. REM ima oblik:
n REM napomena
gde je broj znakova u napomeni ograničen dužinom reda. Ako je napomena duža od jednog reda može se nastaviti u narednom programskom redu, na primer ovako:

```
Program: 10 REM IZOSTANCI SA NASTAVE
         20 REM BIOLOGIJE I DRUGIH NAUKA
         30 REM U PRVOM RAZREDU OSNOVNE
         40 PRINT "BROJ IZOSTANAKA:"
         50 REM POČETNE VREDNOSTI
         60 LET P=0
         70 LET Q=0
```

Jedini problem sa velikim brojem napomena u programu je u tome što svaki slovni znak zauzima lokaciju u memoriji i, ako je reč o velikom programu, svaki bajt može biti dragocen. Na sreću, do tog problema ima još dosta vremena, a do tada ćete već znati i neke načine kako da ga rešite.

READ

Naredba READ (od reči: čitaj) koristi se za dodeljivanje vrednosti promenljivima. Vrednosti se uzimaju iz niza podataka koji se nalaze pod kontrolom iskaza DATA. READ ima oblik:

n READ c1, c2, . . ., ci

gde je n broj reda, svako c je numerička promenljiva, prosta ili indeksirana. Ako se naredba READ odnosi na više promenljivih, koristi se zarez za razdvajanje jedne od drugih.

Prilikom izvršenja naredbe READ naznačenim promenljivima dodeljuju se vrednosti iz »bloka podataka« koji je snabdeven podacima uz naredbu DATA. Prvi podatak iz bloka dodeljuje se prvoj promenljivoj, drugi se dodeljuje drugoj i tako redom. Redosled podataka se održava uz pomoć pokazivača (poentera) koji na početku izvršavanja programa »pokazuje« na prvi podatak. Kako je koji podatak pročitan i dodeljen, pokazivač se pomera na sledeći. Naravno, sve se to događa u računaru, a vi dobijate ispravne podatke.

U pojedinim programima potrebno je da isti podaci budu čitani više puta. Tada se koristi naredba RESTORE koja vraća pokazivač na početak bloka podataka i čitanje niza počinje opet od prvog podatka.

Važno je napomenuti da konstanta (vrednost) dodeljena nekoj promenljivoj na osnovu naredbe READ mora da bude iste vrste kao i promenljiva. To znači da se string-promenljivoj sme dodeliti samo niz slovnih znakova, a numeričkoj promenljivoj numerička konstanta. Greška se napravi lako tokom kucanja programa, a rezultat je da program prekida izvršenje!

Program:	Poenter	Blok podataka	Dodeljeno:
10 READ A, B, C, D	na početku	⟶ 8	⟶ A
20 LET Y=A+B+C+D		13	⟶ B
30 PRINT A, B		3	⟶ C
40 PRINT C, D	Poenter	167	⟶ D
50 PRINT Y	na kraju	⟶ 83	- - - - nije dodeljeno
60 DATA 8, 13, 3			
70 DATA 167, 83			
80 END			

Izlaz: 8 13
3 167
191

Primetili ste verovatno da se uz pomoć READ i DATA naredbi može postići isto kao da ste na početku programa sa LET dodeljivali vrednosti promenljivima A, B, C i D.

DATA

Naredba DATA (od reči: podaci) se koristi da unese podatke u neku vrstu internog, programskog bloka podataka. Ti podaci stoje na raspolaganju da bi se tokom izvršenja programa iz bloka dodeljivali promenjivima koje su određene kroz naredbu READ. Oblik ove naredbe je:

n DATA d1, d2, d3, ... di

gde je svako d poseban podatak u obliku numeričke ili string-konstante. Podaci se obavezno odvajaju zarezom. Podaci se unose prilikom unošenja programa. Tokom izvršenja ova naredba se ignoriše, tj. izvršava se prva naredna naredba, kao da je reč o REM. Kad naiđe naredba READ podaci će biti pronađeni i dodeljeni. Zbog toga se programski red sa DATA može staviti na bilo koje mesto u programu. Međutim, veoma je važno da podaci budu složeni tačno onim redom kojim će biti korišćeni u programu.

Preporučuje se da sve DATA naredbe budu smeštene jedna za drugom na kraju programa, jer su tako najuočljivije programeru.

Program:
```
 10 REM OVAJ PROGRAM UZIMA DATE PODATKE,
 20 REM RADI MALO ARITMETIKU
 30 REM I DAJE REZULAT
 40 READ P, A$, M$, C
 50 PRINT A$; M$; "IZNOSE "; P; " cm"
 60 LET C=C+P
 70 PRINT
 80 PRINT "UKUPNO "; A$; " IZNOSE "; C; " cm"
 90 DATA 17, "PADAVINE", "U APRILU"
100 DATA 29
110 END
```

Izlaz:
```
PADAVINE U APRILU IZNOSE 17 cm

UKUPNO PADAVINE IZNOSE 46 cm
```

Da bi se reči odvojile jedna od druge neophodno je da se u string ugardi razmak kao što je učinjeno u redu 50, 80 i 90. String-konstanta kao podatak u okviru DATA ne mora biti označena navodnicima ali, u tom slučaju, string ne može početi razmakom, cifrom ili nekim znakom. Znači, moguć je oblik:

```
DATA PONEDELJAK, UTORAK, SREDA  ili
DATA "3. AVGUST", "25. AVGUST"      ali nije moguć oblik
DATA 3. AVGUST, 25. AVGUST
```

RESTORE

Naredba RESTORE (od reči: vrati) koristi se za vraćanje poentera na početak bloka podataka. To omogućuje da se sledećom naredbom READ ponovo koriste podaci od početka niza. Ona ima oblik:

n RESTORE

Deo programa:	Blok podataka		Dodeljeno:
10 READ M	12	⟶	M
20 READ N, Y	0.7	⟶	N
30 RESTORE	111	⟶	Y
40 READ A1, A2	4	⟶	A1
...	7.2	⟶	A2
...	29		
100 DATA 12	—		
110 DATA 0.7, 111, 4			
120 DATA 7.2, 29			

Ako poenter pokazuje prazno mesto u bloku podataka (kao na kraju gornjeg bloka) izvršenje naredbe READ prouzrokovaće prekid u izvršenju programa što nikad nije prijatno. Neke verzije BASIC-a omogućuju NODATA test koji sprečava takvu grešku, ali to nije standardna naredba. Zato obratite pažnju prilikom postavljanja nizova podataka za DATA naredbu i uvek proverite da li ima više promenljivih nego podataka. Neki podatak više neće škoditi, jednostavno neće biti upotrebljen(kao u gornjem primeru).

INPUT

Naredba INPUT (od reči: unesi) omogućuje dodeljivanje vrednosti promenljivima tako što ih sami unosite tokom izvršenja programa. To je naredba koja zahteva interakciju sa korisnikom tokom izvršavanja programa. Ona ima oblik:

n INPUT p1, p2, ..., pi

gde je svako p numerička ili string-promenljiva, prosta ili indeksirana. INPUT je sinonim za svaki ulaz u računar, a reč OUTPUT znači izlaz. To se, naravno, ne odnosi na bilo kakav otvor ili vrata na računaru, nego na ulaz i izlaz podataka i rezultata operacija. Izvršavajući naredbu INPUT program pravi pauzu i čeka unošenje podatka, tj. dodeljivanje vrednosti promenljivoj koja je naznačena posle ključa INPUT. Bez posebnog unošenja u program BASIC uvek na ekranu ili štampaču opominje korisnika da očekuje podatak na taj način što se

pojavi znak pitanja (?). Izvršavanje programa ne može da se nastavi dok ne unesete podatak.

Kada na zahtev INPUT unosite podatke, svaki odvojite zarezom. Kod nekih verzija jezika pojavljuje se razmak između numeričkih podataka. Pri unošenju stringa ne moraju se kucati navodnice (kod nekih verzija navodnice su već na ekranu da bi naznačile da se očekuje string). Ako je string podatak bez navodnica ne može početi cifrom, znakom plus ili minus, tačkom ili zarezom.

Svaki podatak mora oblikom odgovarati promenljivoj kojoj se dodeljuje. Ako ubacite, na primer, numerički podatak, a program očekuje string, dobićete naznaku da je greška ili će program ignorisati uneti podatak i tražiti novi. U velikim programima ili u programu koji dugo niste koristili lako se zaboravlja koji tip podatka očekuje program kad se na ekranu pojavi znak pitanja. Za to je vrlo preporučljivo da programski predvidite da se štampa pitanje pre naredbe INPUT. Nema ničeg goreg od žmirkajućeg znaka pitanja kad ne možete da se setite šta se od vas traži.

Program:
```
10 REM PROGRAM KOJI DAJE DAN,
20 REM DATUM I MAX TEMPERATURU
30 PRINT "DAN U NEDELJI"
40 INPUT D$
50 PRINT "DATUM"
60 INPUT M$
70 PRINT "MAX TEMPERATURA";
80 INPUT T
90 PRINT
100 PRINT
110 PRINT "MAKSIMALNA TEMPERTURA U"; D$; ","; M$
120 PRINT "BILA JE"; T; "STEPENI CELZIJUSOVIH"
```

Izlaz:
```
DAN U NEDELJI
?ponedeljak
DATUM
?16. novembra 1984.
MAX TEMPERATURA?9

MAKSIMALNA TEMPERATURA U PONEDELJAK, 16. NOVEMBRA 1984. BILA JE 9 STEPENI CELZIJUSOVIH
```

String "ponedeljak" u drugom redu izlaza je podatak koji je sam korisnik uneo kao odgovor na naredbu INPUT u 40. redu. String "16. novembra 1984." u četvrtom redu izlaza je odgovor na 60. red, a broj 9 odgovor na 80. red programa. Pitanja štampana na osnovu 30, 50 i 70.

reda služe za bolju komunikaciju korisnika sa računarom. Kod pitanja "max temperatura" i znak pitanja i odgovor su dati u nastavku pitanja zahvaljujući znaku (;) na kraju reda 70.

Pojedine verzije BASIC-a omogućuju da se pitanje postavi u okviru iskaza sa naredbom INPUT: INPUT "DAN U NEDELJI"; D$. To eliminiše potrebu da se pitanje postavlja naredbom PRINT pre naredbe INPUT.

GO TO

Naredba GO TO ili GOTO (od izraza: idi do . . .) bezuslovno prenosi izvršavanje programa na red označen uz naredbu. Time se menja niz naredbi od manjih ka većim brojevima redova. Oblik je:

n GO TO br

gde je br broj reda na koji se skreće izvršenje. Izvršavanjem naredbe GO TO program se skreće i nastavlja od reda koji je naznačen, a ne od reda koji sledi iza GO TO. Na ovaj način izvršavanje se nastavlja na višem ili nižem programskom redu, tj. kontrola se prenosi napred ili nazad. Kod korišćenja ove naredbe morate biti pažljivi, jer se vrlo lako desi da skretanjem izvršenja napravite tzv. beskonačnu petlju. U tom slučaju program se neprestano vrti u istim redovima, jer ga GO TO na to primorava.

Beskonačna petlja može se izbeći programskim prekidom za koji postoji mogućnost kod pojedinih verzija. Prekid programa nije predviđen BASIC-om nego je lokalna osobina pojedinih sistema.

Program:
```
10 REM ZBIR BROJEVA OD 1 DO 100
20 REM B=SLEDEĆI BROJ Z=ZBIR
30 B=0
40 Z=0
50 B=B+1
60 Z=Z+B
70 IF B=100 THEN 90
80 GO TO 50
90 PRINT "ZBIR BROJEVA OD 1 DO 100";
100 PRINT "JE"; Z
110 END
```

Izlaz: ZBIR BROJEVA OD 1 DO 100 JE 5050

Izvršenje programa ide redom do 80. reda otkuda se vraća na 50. red dok se ne saberu svi brojevi. Program bi se i dalje vrteo od 50. do 80. reda kad ne bilo 70. reda koji sadrži naredbu koja ga izvodi iz petlje.

Petlju ćemo posebno razmotriti kasnije. Obratite pažnju na to da smo isti efekat programa mogli dobiti i da smo redove 70 i 80 zamenili jednim redom:

```
70 IF B<100 THEN 50
```

IF . . . THEN

Naredba IF ... THEN (od izraza: ako je ... onda ...) usmerava izvršavanje programa zavisno od toga da li je zadovoljen uslov koji se postavi uz naredbu. Oblik je:

```
n IF (izraz-relacija) THEN br
```

gde je br broj reda na koji se skreće izvršavanje, ali samo pod uslovom da je izraz-relacija ISTINIT.
Izraz-relacija sastoji se od dva numerička ili dva string-izraza povezana relacionim operatorom. Prilikom izvršavanja IF ... THEN vrednosti izraza se upoređuju na način koji je određen relacionim operatorom. Kada je rezultat upoređivanja ISTINA izvršavanje se prenosi na programski red koji je označen iza ključa THEN. U slučaju da se upoređivanjem dođe do toga da je to neistina ili LAŽ, program se nastavlja od programskog reda koji sledi iza reda u kojem je IF ... THEN. Relacioni operatori mogu biti:

= jednako	<> nije jednako
< manje od	<= manje ili jednako
> veće od	>= veće ili jednako

Pre upoređivanja izraza povezanih relacionim operatorom neophodno je izvršiti izračunavanja koja će svaki izraz svesti na jednu vrednost. Tek tada relacioni operator stupa na scenu. Uslov je zadovoljen ili nije, tj. izraz je ISTINA ili LAŽ. Na primer:

```
IF(A+1)**2=B*3 THEN br
```

kada je	A=1 i	B=4	izraz je LAŽ	jer 4 nije >=12
	A=2	B=3	ISTINA	9 je =9
	A=3	B=2	ISTINA	16 je >6
	A=4	B=1	ISTINA	25 je >3

Sve ovo do sada i nije bilo preterano nerazumljivo, jer smo to učili u školi, samo tada su to bili matematički znaci, a sada su dobili zvučnije ime: relacioni operatori.

Nešto složenije pitanje predstavlja upoređivanje string-izraza. Prvo i najbitnije je da prihvatite da svako slovo i grafički znak ima

svoju numeričku vrednost, tj. svoj kod. Razni sistemi kodiranja se koriste, ali je najšire korišćen ASCII — američki standard za obeležavanje grafičkih simbola i ključnih reči koje se koriste u BASIC-u. Znači, svaki znak se u računaru dekodira kao broj i kao takav smešta na neku lokaciju u memoriji. Taj numerički kod je osnova upoređivanja string-izraza. Dva stringa bivaju upoređivana tako što procesor prvo upoređuje prve znake u oba stringa, ako su jednaki prelazi na drugi znak u oba stringa i tako redom. Da bi stringovi bili jednaki moraju biti iste dužine i sadržavati iste znake istim redosledom. Ako bilo koji od ovih uslova nije zadovoljen, oni su nejednaki — izraz je LAŽAN. Slova su kodirana po abecedi od nižih ka višim brojevima: A je, znači najmanje, a Z je najveće. Potom dolaze mala slova istim redosledom od a do z. Prema tome, kad se upoređuju dva string-izraza tada se upoređuju kodovi znakova od kojih su sastavljeni.

U stringovima iste dužine prvi par znakova koji nisu jednaki određuje koji string je veći, na primer, pošto je "R" kasnije u abecedi od "J", ako upoređujete sledeće string-izraze

"BORAN" je veće od "BOJAN"

S obzirom na metod upoređivanja, kraći string može da bude veći od dužeg, na primer, "BOSA" je veće od "BOJANA". Međutim, kada su u pitanju dva stringa nejednake dužine, a znaci koji se mogu uporediti su identični, obavezno je veći onaj string koji je duži. Na primer, "BOBAN" je veće od „BOB".

Važno je imati na umu i razmake. Razmak isto ima svoj numerički kod i učestvuje u upoređivanju: "MLADJO " je veće od „MLADJO".

FOR . . . NEXT

FOR (od reči: za, ili kad je . . .) i NEXT (od reči: sledeće) predstavljaju par naredbi za uspostavljanje kontrole u petlji. Naredba FOR postavlja petlju i određuje koliko puta će petlja biti izvedena, tj. koliko puta će se program izvršavati od određenog reda do određenog reda, ignorišući ostale programske redove.

NEXT označava krajnji red niza koji sačinjava petlju i vraća izvršenje na prvi programski red posle naredbe FOR, povećavajući pri tome stanje brojača petlje. Naredbe u petlji su sve one koje se nalaze između FOR i NEXT. Oblik naredbe za FOR je:

n FOR np=nil TO ni2 STEP ni3

gde je np prosta numerička promenljiva koju zovemo kontrolna promenljiva petlje ili brojač petlje; ni1 je numerički izraz koji označava

početnu vrednost kontrolne promenljive-brojača petlje; ni2 je maksimum ili gornja vrednost brojača petlje, a ni3 predstavlja broj za koji se brojač petlje povećava pri svakom prolasku izvršavanja programa kroz petlju. Jednostavno, zar ne?

Obratite pažnju na ključeve TO (od reči: do) i STEP (od reči: korakni). Oblik naredbe za FOR rečima bi izgledao ovako: "Za promenljivu np koja počinje od najniže vrednosti (ni1) i završava se najvišom vrednošću (ni2), korakni-povećaj vrednost np za vrednost (ni3) svaki put kad počinješ petlju" . Morate priznati da je izražavanje u BASIC-u nešto kraće, a tako i mora da bude da ne bismo svu raspoloživu memoriju potrošili na opisivanje onoga što treba uraditi pa onda više nema prostora za podatke.
Ne zaboravite da BASIC, u slučaju kad vi ne naznačite korak tj. vrednost za koju će se uvećavati brojač petlje pri svakom prolazu — podrazumeva STEP 1, pa tako i uvećava brojač.

Da vidimo kako to izgleda u programu, pre nego što nastavimo objašnjavanje najvažnijeg para naredbi u BASIC-u:

```
Program: 10 REM STEPENOVANJE BROJA 3
         20 FOR N=1 TO 10
         30 PRINT "3 NA ";N;". STEPEN=";3**N
         40 NEXT N
         50 END

Izlaz:   3 NA 1. STEPEN=3
         3 NA 2. STEPEN=9
         3 NA 3. STEPEN=27
         3 NA 4. STEPEN=81
         3 NA 5. STEPEN=243
         3 NA 6. STEPEN=729
         3 NA 7. STEPEN=2187
         3 NA 8. STEPEN=6561
         3 NA 9. STEPEN=19683
         3 NA 10. STEPEN=59049
```

Naredba PRINT se jedina nalazi u okviru petlje i ona je izvršena 10 puta uključujući stepenovanje i tako je dobijeno 10 redova rezultata.

Kada u iskazu sa naredbom FOR nije reč o prostim promenljivim, prvo se izvršavaju izračunavanja da bi se dobila početna, najveća vrednost i korak. Potom se početna vrednost dodeljuje brojaču petlje i proverava se da li je ona manja ili jednaka najvećoj vrednosti (za slučaj kada korak ima pozitivnu vrednost) ili se proverava da li je početna vrednost veća ili jednaka najvećoj vrednosti (za slučaj kada ko-

rak ima negativnu vrednost). Ako je stanje brojača u datom rasponu izvršavaju se programski redovi koji slede unutar petlje, sve dok izvršenje ne dođe do NEXT:

```
n NEXT np
```

gde je np ona ista numerička promenljiva koja je postavljena u naredbi uz FOR i koja predstavlja brojač petlje. U izvršenju naredbe NEXT brojač petlje je uvećan (ili smanjen) za vrednost koraka uz proveru maksimalne vrednosti brojača. Ukoliko je novo stanje brojača u okviru datog raspona, izvršavanje programa se ponovo vraća na programski red odmah iza reda u kojem je naredba FOR. Program se dalje vrti u petlji sve dok je kontrolna promenljiva-brojač petlje u datom rasponu. U trenutku kada provera pokaže da je stanje brojača različito od zadate najveće vrednosti izvršavanje programa se nastavlja od reda koji sledi iza reda u kojem je naredba NEXT, tj. program izlazi iz petlje.

Primetili ste da postoji i negativni korak, ali, u tom slučaju, početna vrednost brojača petlje mora da bude veća od krajnje vrednosti brojača. Takođe, početna, granična vrednost i vrednost koraka mogu biti realni ili celi brojevi. Tako je ispravno i:

```
FOR L=0.5 TO 50 STEP 0.5
```

pri čemu će petlja biti izvršavana 100 puta dok uslov ne bude zadovoljen. Evo i primera za negativni korak brojača petlje:

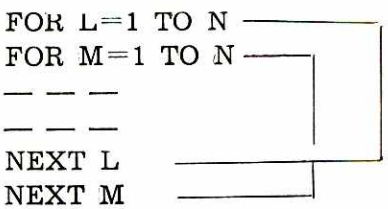
```
FOR L=100 TO 10 STEP —10
```

Petlja će biti izvršena 10 puta dok uslov ne bude zadovoljen. Kad malo uđete u tajne programiranja uverićete se da vam je za izvršavanje nekog zadatka neophodno da jednu petlju smestite unutar druge petlje. Ne brinite se zbog toga, jer BASIC je predvideo i takvu mogućnost. Jednu petlju možete ugnezditi u drugu petlju ali morate voditi računa da se ne preklapaju. Evo primera:

```
K=0
FOR L=1 TO N ————————
FOR M=1 TO N ———————|
K=K+A(L,M)          |
NEXT M ————————————|
NEXT L ————————————|
```

ovo je ispravan oblik jer je petlja M u potpunosti u okviru petlje L

Ako upotrebimo brojače obe petlje kao indekse, sadržaj svih lokacija jedne matrice n sa n biće sabiran jedan po jedan da bi se dobila zbirna vrednost svih brojeva u matrici. O tome će više biti reči kasnije.



ovo je neispravan oblik jer se petlja M preklapa sa petljom L

Vodite računa da uz pomoć nekog GO TO ne usmerite program usred neke petlje, jer ako petlja nije počela sa FOR ni brojač petlje nema nikakvu vrednost pa se stvar neće dobro završiti. Takođe izbegavajte promenu vrednosti brojača petlje tokom izvršenja petlje. I još nešto: nikad ne uzimajte istu kontrolnu promenljivu-brojač petlje i za spoljnu i za unutrašnju petlju.

STOP

Naredba STOP služi za prekid izvršavanja programa pre nego što je izvršenje stiglo do naredbe END. Njen oblik je:

n STOP

U toku izvršenja program se zaustavlja kad stigne do tog reda.

STOP se može pojaviti na više mesta, gde je potrebno da se spreči nastavak izvršenja normalnim redom. Ako program ima takvu strukturu da normalno teče do END — nema potrebe za naredbom STOP.

```
Program:  10 PRINT "UNESI SVOJE GODINE"
          20 INPUT G
          30 IF G>30 THEN 60
          40 PRINT "VREME JE NA TVOJOJ STRANI"
          50 STOP
          60 IF G>55 THEN 90
          70 PRINT "ŽIVOT BRZO PROLAZI"
          80 STOP
          90 PRINT "VREME JE ZA PENZIJU"
         100 END

Izlaz:    UNESI SVOJE GODINE
          ?21
          VREME JE NA TVOJOJ STRANI
```

Pošto je ulaz bio manji od 30 uslov u 30. redu nije zadovoljen tako da izvršenje nije preneto u 60. red nego je nastavljeno prvim sledećim redom i potom zaustavljeno, jer se ni jedna od naredbi ne odnosi na godine manje od 30. Da je uneta vrednost bila, na primer, 56 — izvršenje bi se iz reda 30 prenelo u red 60, pa, pošto je i taj uslov zadovoljen, u red 90. U gornji primer moguće je ugraditi naredbu koja će prekidati izvršenje kad se unese negativna vrednost za godine:

```
25 IF G<0 THEN 100
```

Pojedini operativni sistemi omogućuju da se izvršavanje prekine pritiskom na dirku STOP. To nije BASIC, a i nije kreativno.

ON . . . GO TO

Naredba ON ... GO TO (od izraza: kad je... idi do) omogućuje uslovno skretanje izvršenja na veći broj programskih redova. Oblik je

```
n ON ni GO TO br1, br2, ..., bri
```

gde je ni numerički izraz, a br1, br2 do bri su brojevi redova na koje može izvršenje da se prenese zavisno od rezultata numeričkog izraza.

Kad izvršavanje dođe do reda koji sadrži ovu naredbu prvo se izračunava vrednost numeričkog izraza. Ako je rezultat decimalni broj, zaokružuje se na prvi manji ceo broj. Ako je vrednost 1, izvršenje se prenosi na red br1, ako je rezultat 2 prenosi se na red br2, a kad je vrednost i, na red bri. Na primer:

```
ON X GO TO 300, 335, 450, 800
```

ako je X=1 ide na red 300
ako je X=2 ide na red 335
ako je X=3 ide na red 450
ako je X=4 ide na red 800

U gornjem primeru vrednost X mora biti između 1 i 4. Broj mogućih grananja programa na osnovu ove naredbe je teoretski ograničen samo brojem znakova u jednom redu. Praktično, neće vam nikada biti potreban ceo red.

POTPROGRAMI

Veoma često se događa da jedan deo programa treba izvršavati dva i više puta u toku celog programa. Takozvani "pešački" način za to je da svaki put kad su vam potrebne određene naredbe istim redom

vi ih ponovo ukucate u program i one se izvršavaju redom od nižih brojeva redova ka višim. Da bi se izbegao ovaj način koji odnosi vreme i memoriju računara, deo iskaza-programskih redova koji se koriste više od jedanput u jednom izvršavanju programa grupiše se kao poseban potprogram. Taj deo se izvršava na različitim stupnjevima odvijanja programa, kad god zatreba, rutinski. Zato se takvi delovi programa nazivaju potprogrami, subprogrami ili, najčešće, subrutine. Subrutina se samo jedanput unosi u program, a može biti izvršena nebrojeno mnogo puta, ako je potrebno. BASIC koristi naredbu GO SUB za skretanje na subrutinu, a naredbu RETURN za povratak u glavni deo programa.

GO SUB

Naredba GO SUB (od izraza: idi do subrutine) skreće izvršavanje programa na početak subrutine koju je programer smislio. Evo je:

n GO SUB br (kod nekih sistema zajedno: GOSUB)

gde je br broj reda prvog iskaza subrutine. Izvršavanjem GO SUB program skače na prvu naredbu subrutine i dalje ide redom dok ne stigne do naredbe RETURN. Sledeći primer programa nema praktičnu vrednost nego samo pokazuje korišćenje naredbe GO SUB i RETURN:

```
Program:  10 PRINT "PRIKAZ SUBRUTINE"
          20 LET A=10
          30 GO SUB 100
          40 PRINT A, B
          50 GO SUB 100
          60 PRINT A, B
          70 STOP
         100 A=A/2
         110 B=3*A
         120 RETURN
         130 END

Izlaz:    PRIKAZ SUBRUTINE
          5              15
          2.5            7.5
```

Subrutina se sastoji od dva reda, 100 i 110 i izvršava se dva puta. Posle prvog izvršenja program se vraća na 40. red, a drugi put na 60. red. Primetili ste, sigurno, da se program može skrenuti na istu subrutinu više puta. Izvršenje se svaki put vraća na onaj red glavnog programa koji sledi iza reda u kojem je naredba GO SUB.

Jednu subrutinu možete pozvati (skrenuti izvršenje na nju) u toku izvršenja druge subrutine. Znači, naredba GO SUB može da se nađe u okviru druge sekvence GO SUB ... RETURN. Takođe je važno znati da se naredba GO SUB može programirati u okviru sekvence FOR ... NEXT pri čemu izvršavanje subrutine ne ometa izvršavanje petlje.

RETURN

Naredba RETURN (od reči: vrati se) služi za vraćanje izvršavanja iz subrutine u glavni programski tok ili u subrutinu iz koje je pozvana. RETURN mora obavezno biti poslednji iskaz u svakoj subrutini. Oblik je:

n RETURN

Na naredbu RETURN izvršavanje se usmerava na prvi sledeći red koji sledi za redom u kojem je bila naredba GO SUB koja je izazvala izvršenje subrutine. Svaki put kad se programski izvršenje upućuje na određenu subrutinu operativni sistem zabeleži programski red u kojem je bila naredba GO SUB i posle RETURN automatski nastavlja izvršavanje od narednog reda. Zbog toga naredba RETURN ne sadrži broj reda na koji se vraća izvršenje programa. Ako to logika programa zahteva, jedna subrutina može imati više od jedne naredbe RETURN, ali je ona na kraju subrutine obavezna.

DIM

Pre objašnjenja ove naredbe treba napomenuti da se u mnogim prilikama podaci ne odlažu u memoriju i koriste pojedinačno nego u nizovima. Zbog toga BASIC predviđa mogućnost manipulisanja većim nizovima brojeva i naziva tj. numeričkih i string-promenljivih i konstanti. Kad je reč o većem broju podataka potreba za preciznošću se povećava, jer je veća i opasnost od toga da se podaci pomešaju. Predviđeno je da do toga ne dođe (ako ne pogrešite pri unosu podataka, ali to je već pitanje principa: Glupost na ulazu — glupost na izlazu).

Naredba DIM (skraćenica od reči: dimenzioniši) se koristi da bi se definisala dužina niza promenljivih i da bi se za svaku od njih rezervisala lokacija u memoriji. BASIC omogućuje korišćenje dvodimenzioalnih nizova koji se još zovu matrice i jednodimenzionih nizova koji se zovu i vektori. Oblik ove naredbe je:

n DIM p1(k), p2(k), ..., pi(k)

gde je p numerička ili string-promenljiva koja se deklariše kao niz promenljivih. Indeks k je numerička konstanta koja definiše veličinu

niza. Kada je reč o dvodimenzionom nizu — matrici potrebna su dva indeksa, međusobno odvojena zarezom: p(k1, k2). U tom slučaju k1 označuje broj redova u matrici, a k2 označuje broj kolona u matrici. Primeri za deklarisanje DIM su sledeći:

```
DIM B(25), A$(10), X(5, 20)
```

Kada u okviru jedne naredbe DIM deklarišemo više nizova, promenljive se odvajaju zarezom.

Niz promenljivih može se koristiti u programu i ako to nije deklarisano DIM naredbom. U tom slučaju BASIC podrazumeva da je gornja vrednost indeksa: 10. Činjenica da je neka promenljiva data sa indeksom dovoljna je da je BASIC prepozna kao niz, iako joj nije prethodilo DIM.

Raspon vrednosti indeksa za niz deklarisan kroz naredbu DIM je između 0 i broja naznačenog uz naredbu. Za nedeklarisani niz taj raspon je od 0 do 10, znači 11 lokacija u memoriji.

Naredba DIM mora u programu biti pre prvog korišćenja niza. Naredni program daje najmanji broj u jednom nizu od 20 brojeva:

Program:
```
 10 REM DEKLARACIJA VELIČINE NIZA
 20 DIM B(20)
 30 REM POSTAVLJA PETLJU DA BI NIZU
 40 REM BILE DODELJENE VREDNOSTI
 50 FOR K=1 TO 20
 60 READ B(K)
 70 NEXT K
 80 REM KRAJ PETLJE
 90 REM POZIVA BROJ SA PRVE LOKACIJE
100 REM KAO PRIVREMENI NAJMANJI BROJ
110 X=B(1)
120 REM PETLJA ZA PROVERU VREDNOSTI
130 REM IZ NIZA I POSTAVLJA NOVI
140 REM NAJMANJI BROJ KAD NAĐE
150 FOR I=2 TO 20
160 IF X<B(I) THEN 180
170 LET X=B(I)
180 NEXT I
190 REM KRAJ PETLJE
200 PRINT "NAJMANJI BROJ U DATOM NIZU:"; X
210 DATA 123, 18, 789, 13, 67, 920, 45, 101, 9, 99
220 DATA 33, 78, 313, 47, 19, 28, 127, 10, 665, 77
230 END
```

Izlaz: NAJMANJI BROJ U DATOM NIZU:9

Razne verzije jezika daju različita rešenja za najmanju vrednost indeksa ili bazu niza. Kod nekih ona je uvek 1 ako drugačije nije naznačeno. Na vama je da utvrdite kako je kod vašeg računara, ako vam to zatreba. Inače, pitanju indeksiranja posvećena je posebna pažnja zbog već pomenute potrebe da se isključi svako pogrešno dodeljivanje vrednosti nekoj promenljivoj u nizu. Evo primera tabelarne šeme indeksa u dvodimenzionom nizu (4×6):

1,1	1,2	1,3	1,4	1,5	1,6
2,1	2,2	2,3	2,4	2,5	2,6
3,1	3,2	3,3	3,4	3,5	3,6
4,1	4,2	4,3	4,4	4,5	4,6

Programski se ovim indeksima mogu dodeliti vrednosti na sledeći način (dat je segment mogućeg programa):

```
 70 FOR L=1 TO 4
 80 FOR M=1 TO 6
 90 READ A(L, M)
100 NEXT M
110 NEXT L
...
...
...
300 DATA 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
310 DATA 11, 12, 13, 14, 15, 16, 17, 18, 19, 20
320 DATA 21, 22, 23, 24
330 END
```

Petlja M (unutrašnja) se izvršava 6 puta kod svakog prolaza kroz petlju L koja je spoljna. Tako 6 podataka biva učitano u svaki red niza — matrice

Naredba DATA u 300, 310. i 320. redu smešta podatke redom u blok podataka. Potom naredba READ u 90. redu dodeljuje vrednosti nizu A tokom izvršenja na sledeći način:

kolone M

redovi L

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24

Evo sada jednog praktičnog programa koji ilustruje upotrebu DIM, FOR ... NEXT i IF ... THEN naredbi. On služi za slaganje-sortiranje liste do 100 imena po abecedi:

Program:

```
 10 REM POSTAVLJA DIMENZIJU NIZA
 20 REM I BROJAČ NA NULU
 30 DIM N$(101)
 40 LET K=0
 50 PRINT "UKUCAJ-UNESI IMENA"
 60 PRINT "NEMA — ZA KRAJ LISTE"
 70 PRINT
 80 REM POSTAVLJA PETLJU ZA UNOS PODATAKA
 90 REM NA LOKACIJE NIZA
100 FOR M=1 TO 101
110 INPUT N$(M)
120 REM TEST ZA KRAJ LISTE
130 IF N$(M)="NEMA" THEN 200
140 REM DODAJE 1 BROJAČU IMENA
150 K=K+1
160 NEXT M
170 REM KRAJ PETLJE
180 REM %%%%%%%%
190 REM UGNEZDI PETLJE
195 REM ZA SORTIRANJE IMENA
200 FOR M=1 TO K—1
210 FOR J=M+1 TO K
220 REM UPOREĐUJE IMENA I ZAMENJUJE MESTA
230 REM KAD NAĐE IME MANJE VREDNOSTI (ASCII)
240 IF N$(M)<=N$(J) THEN 280
250 LET X$=N$(M)
260 LET N$(M)=N$(J)
270 LET N$(J)=X$
280 NEXT J
290 REM KRAJ UNUTRAŠNJE PETLJE
300 NEXT M
310 REM KRAJ SPOLJNE PETLJE
320 PRINT
330 PRINT
340 PRINT "EVO SLOŽENE LISTE"
350 PRINT
360 REM POSTAVLJA PETLJU ZA IZLAZ PODATAKA
370 REM IMENA SU SADA PO ABECEDNOM REDU
380 FOR J=1 TO K
```

```
390 PRINT N$(J)
400 NEXT J
410 REM KRAJ PETLJE
420 END
```

Izlaz:

```
UKUCAJ-UNESI IMENA
NEMA-ZA KRAJ

?beba
?boka
?brana
?simo
?mira
?ana
?olga
?nema

EVO SLOŽENE LISTE
ANA
BEBA
BOKA
BRANA
MIRA
OLGA
SIMO
```

Možda se pitate zašto je niz dimenzionisan na 101 lokaciju. Razlog je to što pored 100 imena treba da primi i "nema", reč za završetak programa. Pažljivo analizirajte izvršenje redova od 250 do 270. Zamena koja se u njima izvodi je osnova svih programa sortiranja podataka i biće vam potrebna i kasnije.

Po izvršenju petlje (100—160) niz N$ sadrži sledeće članove:

N$(1)	N$(2)	N$(3)	N$(4)	N$(5)	N$(6)	N$(7)	N$(8)
Beba	Boka	Brana	Simo	Mira	Ana	Olga	nema

Tokom prolaska kroz petlju članovi niza menjaju mesta sve dok se ne dobije uzlazni niz po vrednosti — od A ka Z.

FUNKCIJE

Funkcije su definisane procedure koje mogu biti upotrebljene višekratno u programu jednostavnim pozivanjem njihovog imena. BASIC sadrži listu preddefinisanih funkcija za izračunavanje mnogih često korišćenih numeričkih funkcija, na primer za izračunavanje kvadratnog korena brojeva. Jezik takođe omogućuje korisniku da definiše funkcije koje su mu potrebne naredbom DEF (vidi str. 48). Kada je, nekom od naredbi, funkcija pozvana i izvršena dobija se rezultat koji se koristi za dalje izvršavanje programa. Funkcija se može koristiti na svakom mestu u programu gde je dozvoljen numerički izraz. Ona može stajati sama za sebe ili u okviru nekog izraza.

Standardne funkcije

Bilo koja funkcija se dobija naznačivanjem njenog imena i takozvanog argumenta funkcije koji se daje u zagradi. Argument može biti u obliku bilo kojeg numeričkog izraza. Izračunavanjem on daje prostu vrednost koja se koristi kao podatak za funkciju. Numerički izraz koji daje vrednost argumenta može sadržati druge funkcije.
U definicijama funkcija koje slede, X je argument funkcije:

Trigonometrijske funkcije

ATN(X) daje arkustangens od X, u radijanima,
COS(X) daje kosinus od X, u radijanima,
SIN(X) daje sinus od X, u radijanima,
TAN(X) daje tangens od X, u radijanima,

Eksponencijalne funkcije

EXP(X) eksponent od X, a to je vrednost baze prirodnih logaritama (e=2.71828) dignuta na stepen x,
LOG(X) prirodni logaritam od X. X mora biti >0.
SQR(X) kvadratni koren od X. X mora biti >Ø.

Aritmetičke funkcije

ABS(X) daje apsolutnu vrednost X, bez obzira na znak,
INT(X) najveći ceo broj koji nije veći od X,
SGN(X) utvrđuje znak za X i daje vrednost 1 ako je x>0, vrednost 0 ako je X=0 i vrednost —1 ako je X<0.

Naredni program nema drugu primenu izuzev da prikaže kako izgledaju pojedine funkcije:

Program:
```
10 LET A=17
20 LET B=—7.62
30 LET C=SQR(A)
40 LET D=EXP(A)
50 LET E=LOG(A)
60 LET F=SGN(B)
70 LET G=INT(A+B)
80 PRINT "KVADRATNI KOREN IZ ";A;" JE:";C
90 PRINT "VREDNOST E NA ";A". STEPEN JE:";D
100 PRINT "LOGARITAM OD ";A;" JE:";E
110 PRINT "FUNKCIJA SGN OD ";B;" JE:";F
120 PRINT "NAJVEĆI CEO BROJ KOJI NIJE >(A+B): ";G
130 END
```

Izlaz:
```
KVADRATNI KOREN IZ 17 JE:4.1231
VREDNOST E NA 17. STEPEN JE:24154953
LOGARITAM OD 17 JE:2.8332
```

FUNKCIJA SGN OD —7.62 JE:—1
NAJVEĆI CEO BROJ KOJI NIJE >(A+B): 9

Korišćenje standardnih funkcija olakšava značajno posao pri programiranju. Program koji bi sadržao sve formule za izračunavanje ovih funkcija bio bi znatno duži i složeniji. Ovako, samo upotrebite ime funkcije, a sve ostalo je već u računaru.
Pomoćna funkcija RND (skraćenica za random: slučajni) odabira tzv. pseudo-slučajni broj iz niza koji je uvek na raspolaganju. Taj niz je od 0 do 1. U mini-BASIC-u nije potreban argument uz RND ali ga mnoge verzije imaju da bi se na taj način odredila početna vrednost niza iz kojeg računar uzima slučajni broj.

RANDOMIZE

Ova naredba (od reči: uproseči) menja početnu vrednost niza iz kojeg računar bira pseudo-slučajni broj. Na taj način pri svakom izvršenju RANDOMIZE naredbe zajedno sa funkcijom RND dobija se drugi (nepredvidljiv) broj.
Naredba RANDOMIZE ima oblik:

n RANDOMIZE (RANDOM ili RND u drugim verzijama)

Izvršenjem ove naredbe dobija se nova početna vrednost niza slučajnih brojeva. Evo primera za RANDOMIZE i RND:

Program:
```
10 REM SIMULACIJA BACANJA KOCKE
20 PRINT "KOCKA JE PALA NA:"
30 PRINT
40 RANDOMIZE
50 REM PETLJA ZA 10 BACANJA
60 FOR J=1 TO 10
70 LET K=INT(6*RND)+1
80 PRINT K;
90 NEXT J
100 END
```

Izlaz:
```
KOCKA JE PALA NA:
2 3 1 4 6 1 5 2 3 4
```

Ako ponovite izvršenje programa dobićete drugi niz brojeva, jer je naredba RANDOMIZE promenila početnu vrednost. Da nije bilo te naredbe u izlazu bi bili isti brojevi. U 70. redu vrednost slučajnog broja je pomnožena sa 6, jer osnovni niz je, rekli smo, između 0 i 1. Toj vrednosti dodaje se 1, pošto funkcija INT zaokrugljuje na prvi manji ceo broj, pa bi izgledalo kao da naša kocka ima pet strana.

Možda vam se čini da je previše prostora u ovom kratkom kursu BASIC-a posvećeno nečemu što nema veliku važnost. RANDOMIZE i RND su korisni u raznim programima, od simuliranja podataka u poslovnim i naučnim programima do dodeljivanja slučajnih koordinata figurama u nekoj od igara. U nekim verzijama ne koristi se RANDOMIZE jer RND ima argument: RND(X) koji menja početnu vrednost niza svaki put kad je funkcija pozvana.

TAB

Funkcija TAB (skraćenica od: tabulator) simulira ono što imate na svakoj pisaćoj mašini. Ona kontroliše mesto štampanja i zato se koristi uvek uz naredbu PRINT. TAB(X) pomera mesto početka štampanja na kolonu koja se dobija izračunavanjem vrednosti X, zaokrugljene na prvi manji ceo broj. To znači da X može biti samo pozitivan broj. Pod uslovom da vaš računar ima 32 znaka u jednom redu, gledano po vertikali on ima 32 kolone. Kad u programu predvidite TAB(13) to znači da će štampa početi u 14. koloni, gledao s leva na desno. Separator (;) je obavezan posle TAB(X), jer obezbeđuje nastavak bez razmaka. Najbolje je da sami malo eksperimentišete, jer TAB je vrlo važna funkcija, naročito u grafičkim programima i programima kod kojih je važna pozicija izlaza na ekranu.

```
Program: 10 REM FUNKCIJA TAB
         20 PRINT "01234567890123456789012345678901"
         30 LET A=6
         40 PRINT TAB(A);"B";TAB(10)"M"
         50 END

Izlaz:   01234567890123456789012345678901
              B     M
```

Korisničke funkcije

BASIC predviđa da sam korisnik definiše funkcije koje su mu potrebne u programu pored standardnih funkcija. Kada ih korisnik definiše na jednom mestu u programu ove, korisničke funkcije se izvršavaju na svakom mestu na kojem se naznači njihov naziv.

DEF FN

Naredba DEF (skraćenica od reči: definiši) je preduslov za upotrebu korisničke funkcije. Ona ima oblik:

n DEF FN A(P)=e

gde je FN A ime funkcije koje daje korisnik, argument p je jedna ili više prostih numeričkih promenljivih, dok je e simbol matematičke definicije funkcije u obliku numeričkog izraza. Prva dva slova korisničke funkcije uvek moraju da budu FN, a za njima sledi jedno slovo koje izdvaja jednu funkciju od druge. Za naziv funkcije može se uzeti bilo koje slovo abecede. Funkcija A(P) ili, kako su nas učili u školi, funkcija "A od P" se koristi u programu tako da se prvo dodeli vrednost p, potom se izračuna vrednost izraza sa desne strane znaka jednakosti, a onda se dobijena vrednost funkcije A koristi u programu na mestima na kojima je naznačeno. Definicija funkcije uvek mora programski da se predvidi pre prvog pozivanja te funkcije.

U narednom primeru data je funkcija za zaokruživanje brojeva na dve decimale:

```
Program: 10 REM FN DVE DECIMALE
         20 DEF FN A(Y)=INT(100*Y+0.5)/100
         30 FOR K=1 TO 2
         40 PRINT "NEZAOKRUŽEN BROJ JE:";
         50 INPUT B
         60 PRINT "ZAOKRUŽEN NA DVE DECIMALE:";FN A(B)
         65 PRINT
         70 NEXT K
         80 END

Izlaz:   NEZAOKRUŽEN BROJ JE:?32.537139
         ZAOKRUŽEN NA DVE DECIMALE:32.54


         NEZAOKRUŽEN BROJ JE:9.9944678
         ZAOKRUŽEN NA DVE DECIMALE:9.99
```

* * *

Ovim smo završili prikaz BASIC-a u njegovom minimalnom obliku. BASIC se neprestano razvija i što je savremeniji računar koji imate pred sobom to je savršeniji BASIC ugrađen u njega. U priručniku koji korisnik dobija uz uređaj opisane su sve naredbe i funkcije koje korisnik može upotrebiti u programiranju.

Jedini način za učenje plivanja je da se uđe u vodu. Tako je i sa programiranjem. Stisnite nos sa dva prsta i... skočite OTVORENIH OČIJU u ovaj svet koji nije tako komplikovan kako izgleda na prvi pogled!

V/ ZA POLIGLOTE

Po nekoliko reči o programskim jezicima: FORTRAN, ALGOL, PASCAL, APL, LISP, LOGO, FORTH — Poređenje sa BASIC-om

Ovo poglavlje nema za cilj da prikaže sve programske jezike niti da da potpun uvid u prikazane jezike. Prikazujući nekoliko poznatih i manje poznatih jezika dajemo pregled važnih principa programiranja da bismo vam olakšali sagledavanje razlika među njima, radi kasnijeg opredeljivanja za dublje upoznavanje. Jezici su prikazani iz četiri osnovna aspekta.

Koliko je potrebno truda da početnik nauči da programira u nekom jeziku i to najmanje do stepena da može da rešava jednostavnije probleme putem računara, predstavlja JEDNOSTAVNOST jezika. Sa kojom lakoćom se dati programski jezik može upotrebiti za rešavanje komplikovanih problema, govori MOĆ jezika. Jednostavnost i lakoća su relativno nezavisni. Neki programski jezici su teški za učenje ali omogućuju relativno lako rešavanje problema, dok je kod drugih jezika situacija upravo obrnuta.

Treći aspekt programskog jezika je KOMPATIBILNOST kako među sistemima sa istim jezikom tako i među samim jezicima, tojest, koliko je lako preći na drugi jezik kada se nauči prvi.

PROBLEMSKA ORIJENTACIJA objašnjava do koje mere programski jezik pomaže ljudsko razmišljanje o problemima. Neki jezici svojom strukturom zapravo pomažu razmišljanje o problemu i omo-

gucuju programeru da relativno lako opiše problem dok drugi jezici mogu upravo da blokiraju put do rešenja problema tako što su u suprotnosti sa prirodnim načinom mišljenja. Ovaj aspekt je u bliskoj vezi sa jednostavnošću i moći jednog jezika jer lakoća učenja i lakoća primene zahtevaju raznovrsno obeležavanje radi opisivanja problema.

Klasični programski jezici

Njihova kratka karakteristika je da nisu ni jednostavni ni mnogo moćni. Jezici ove grupe bi bili FORTRAN, ALGOL i PASCAL. Oni su dosta rasprostranjeni na visokim školama i u privredi. Njihova popularnost u ovim oblastima je sasvim razumljiva ako se zna da privreda uzima školovan kadar sa univerziteta, a planeri školskih programa su izvanredno osetljivi na potrebe privrede (misli se, naravno, na planiranje u SAD, gde je to najrazvijenije i otkuda se preslikava u svet).

FORTRAN (od FORmula TRANslation) je prvi viši programski jezik i još uvek dominira, pre svega, u fizici, matematici i tehničkim oblastima uopšte. Ipak, danas se za FORTRAN može reći da mu nedostaju opšta povezanost i dobro rešene komande za kontrolu toka programa. Programi u FORTRAN-u upotrebljavaju veoma mnogo uslovne prelaze koji šalju izvršenje programa na razne delove programskog toka, tako da se svi programi, osim onih najjednostavnijih, čitaju uz veliki napor, jer se ne mogu čitati jednostavno od vrha na dole. Neka bude pomenuto da nove verzije jezika FORTRAN, kao FORTRAN 77, imaju naredbe za kontrolu toka koje su veoma slične onima u ALGOL-u.

ALGOL (ALGOrithmic Language) pokazuje viši stepen unutrašnje povezanosti i osmišljenije kontrolne strukture. ALGOL se koristi kao univerzalni jezik za opis i razmenu algoritama među ljudima koji se bave naukom o računarima. Strukture kao što su BEGIN ... END, IF ... THEN ... ELSE, FOR ... DO, i WHILE ... DO su predstavljale prethodnicu za nova rešenja kontrole programskog toka u mnogim kasnijim programskim jezicima. Ipak, ALGOL-u nedostaje standardni set naredbi za ulaz i izlaz podataka.

PASCAL, naslednik ALGOL-a, zauzeo je značajne pozicije u akademskim krugovima. PASCAL je dovoljno kompaktan da se prilagodi ograničenim memorijama mikro računara od 48 ili 64 kilobajta. Dovoljno je mali da se lako primenjuje i njegova sintaksa i semantika su dobro određene. Brzina izvršavanja programa je nekoliko puta veća nego kod BASIC-a.

Glavni nedostatak FORTRAN-a, ALGOL-a i PASCAL-a je to što nisu interaktivni. Da biste probali čak i najjednostavnije naredbe po-

trebno je prvo kreirati program u izvornom obliku, prevesti ga u mašinski jezik, a zatim ga povezati u sistem — da bi se mogao izvršavati. Iz toga proizlazi da eksperimentisanje sa jednom ili nekoliko naredbi zahteva neproporcionalno mnogo vremena i napora. Bilo bi mnogo efikasnije eksperimente obavljati u takozvanom interaktivnom radu gde je moguće testiranje samo nekoliko naredbi bez sklapanja i pokretanja celog programa.

Drugi važan nedostatak ovih programskih jezika je u tome što se kompleksne procedure moraju razraditi do najsitnijih detalja, do nivoa manipulacije sa elementarnim memorijskim ćelijama u računaru. Iako računar po samoj svojoj strukturi i radi tako što pristupa jednom mestu u memoriji u jednom trenutku, to ne mora da se odražava u radu sa nekim programskim jezikom. Ovo se rešava naredbama koje čitave liste brojeva ili slova shvataju kao elementarni podatak.

Princip »samo jedna radnja u jednom trenutku« je, kao ograničavajući faktor, ugrađen u mnoge programske jezike. To se posebno odnosi na korišćenje kod definisanja funkcija od strane samog korisnika. Problem je u tome što te funkcije ne mogu vratiti vrednost niza ili liste po pozivanju. U FORTRAN-u, ALGOL-u i PASCAL-u se kompleksni problemi ovog tipa rešavaju primenom potprograma ili procedura. Kao rezultat dobijamo više poziva procedura nego samih funkcijskih naredbi u programu. Takav način rešavanja je na neki način nužno zlo jer je mnogo pogodnija lista čistih funkcija nego odgovarajuća lista poziva potprograma ili procedura. Kao jedan primer principa o kojima se ovde govori navešćemo problem kvadriranja svakog elementa neke matrice i zatim transponovanja dobijene međumatrice (ako vam primer nije najjasniji, preskočite ga!). Ako bi kvadriranje i transponovanje bili definisani kao funkcije mogao bi se napisati sledeći program:

```
MATRICA2=TRANSPONOVANA (KVADRIRANA (MATRICA1))
```

Međutim, to nije moguće ni u jednom od navedenih jezika, pa bi rešenje izgledalo ovako:

```
KVADRIRAJ (MATRICA1, PRIVREMENA MATRICA)
TRANSPONUJ (PRIVREMENA MATRICA, MATRICA2)
```

gde je prvi argument svake procedure matrica nad kojom se operiše, a drugi argument matrica koja se dobija kao rezultat. U FORTRAN-u bi naredba CALL prethodila svakoj od procedura. Kao što se vidi, obeležavanje po funkcijama bilo bi mnogo jasnije.

BASIC — Sigurno jednostavan ali daleko od moćnog.

Visok stepen interaktivnog rada je svakako najvažniji kod prosuđivanja o jednostavnosti nekog jezika. BASIC je poznat kao jedan od najsavršenijih jezika u smislu interaktivnosti unutar samog jezika. Linije BASIC programa se ukucavaju direktno u BASIC sistem i program može biti izvršen u svakom momentu bez trošenja vremena na prevođenje u mašinski jezik i ostale operacije vezane za povezivanje izvršnog koda u operativni sistem. Kao dodatna pogodnost se javlja mogućnost da većina BASIC sistema može da izvršava pojedine programske linije i van same definicije programa.

BASIC je postao programski jezik za mikroračunare sredinom i krajem sedamdesetih godina iz prostog razloga jer je bio dovoljno mali da se smesti u ograničene memorije prvih mikroračunara. Cena ovoga je naravno bila plaćena velikim ograničenjima u mogućnostima samog jezika.

Isto kao FORTRAN-u i BASIC-u nedostaju prave naredbe koje rešavaju strukturu samog programa. Mnoge verzije ograničavaju imena promenljivih na samo dva slova i čine upotrebu smisaonih imena skoro nemogućom. Ipak, najveća mana BASIC-a je nedostatak procedura ili potprograma. Mnoga uputstva pojednostavljeno navode GOSUB kao naredbu za poziv potprograma. Ustvari to nije ništa drugo nego bezuslovni prelaz sa jednog programskog bloka na drugi sa mogućnošću kasnijeg povratka na prvobitni blok.

Kako vreme prolazi iskustava je sve više pa se sada mnogo više vodi računa o tim aspektima jezika. ANSI standard (SAD) već je uveo većinu tih struktura u listu naredbi BASIC-a.

APL i **LISP** su moćni ali ne i jednostavni jezici.

Svi do sada razmatrani jezici su relativno skromni po snazi uglavnom zbog osobine da samo jednu stvar rade u datom momentu. Ako razmotrimo dostupne jezike koji rešavaju ovaj problem, nailazimo na APL i LISP. Na realizacije APL-a (A Programming Language) i LISP-a (LISt Processing) nailazimo i kod sasvim jevtinih mini i mikro računara. APL i LISP su interaktivni i izuzetno moćni jezici ali većinu potencijalnih korisnika odbija njihovo neuobičajeno obeležavanje.

APL i LISP imaju fanatične pobornike, ali ni jedan nije široko prihvaćen, verovatno zbog već pomenutog problema obeležavanja. Ipak, ispod tih osobina leže programski sistemi koji bi mogli biti nazvani futurističkim u poređenju sa FORTRAN-om, ALGOL-om i PASCAL-om.

APL i LISP omogućuju korisniku da misli na nivou strukture podataka. Strukture koje podržava APL su nizovi (skalari, vektori, matrice i višedimenzionalni nizovi). U LISPu osnovna struktura podataka je lista (pri tom elementi neke liste takođe mogu biti liste). Oba navedena programska jezika omogućuju korisniku da definiše funkcije koje manipulišu kompleksnim strukturama podataka. Primer kvadriranja i transponovanja matrice bi u APL-u izgledao ovako:

MATRICA2 — TRANSPONUJ KVADRIRAJ MATRICA1

APL i LISP su takođe i visoko interaktivni. Funkcija može biti izvršena odmah po definisanju. Naredbe se, takođe, mogu izvršavati van definicije samog programa tako da je vrlo lako probati razne naredbe da bi se ispitao njihov efekat. To je naročito dragoceno kod moćnih jezika kao što su APL i LISP gde efekat samo jedne programske linije može biti relativno dalekosežan.

I APL i LISP podržavaju modularno programiranje u kom se problemi razbijaju u nekoliko kratkih funkcijskih definicija. Pošto tako svaka funkcija može biti testirana odvojeno, logičke greške se relativno lako otkrivaju i ispravljaju.

APL i LISP omogućuju korisniku da uskladišti veliki broj definicija kako funkccija tako i podataka u posebnom delu memorije računara i tako smanji potrebu za obraćanje eksternoj memoriji. Taj deo memorije se dodeljuje dinamički, tj. proširuje se kada se dodaju nove definicije funkcija i nove strukture podataka. Dodeljivanje memorije je za korisnika potpuno nevidljivo tako da ne postoji potreba za komandama za dodeljivanje memorijskog prostora preko, na primer: DIM() naredbe. U bilo kom trenutku je moguće snimiti na disketu celokupnu memoriju kao jedinstvenu datoteku i tako kasnije u potpunositi rekonstruisati sistem.

Oba ova jezika predstavljaju zaokružene programske sisteme. I jedan i drugi imaju ugrađene rutine za upravljanje memorijom, oporavak od nastalih grešaka i parametre ulaza i izlaza koji se mogu prepraviti i korisniku omogućiti da napravi sopstvenu verziju razvojnog sistema.

Iako su oba jezika interaktivni i moćni oni koriste neuobičajeno obeležavanje i nestandardne editore. APL koristi nestandardne znake i zahteva specijalnu tastaturu dok LISP koristi standardne znake ali upotrebljava aritmetiku takozvane obrnute poljske notacije i upotrebljava zagrade da razdvoji strukturu od računa.

Ni APL ni LISP nemaju naredbe za strukture ponovljenih radnji. Na sreću, oba jezika podržavaju takav stil programiranja koji sma-

njuje potrebu za ponavljanjem. To je moguće jer je ugrađeno puno naredbi koje tretiraju kompletne strukture podataka odjedanput. Kod drugih jezika, većina naredbi za ponavljanje upotrebljava za pojedinačnu obradu elemenata većih struktura podataka; liste, vektora, ili matrice.

LOGO — Jednostavan i moćan.

Iako je inspirisan LISP-om, LOGO se ne oslanja toliko na obeležavanje sa upotrebom zagrada i dozvoljava upotrebu standardnog obeležavanja u aritmetici (pored obrnute poljske notacije).

Uprkos pojednostavljenju obeležavanja LOGO je zadržao mnoge napredne osobine jezika-prethodnika. Postoje i neke karakteristike LOGO-a koje ga izdavajaju:

1. Interaktivni interpretirani kod,
2. Moćne naredbe za kreiranje i modifikaciju celovitih struktura podataka,
3. Funkcionalno obeležavanje koje omogućuje isticanje hijerarhijske strukture programa,
4. Dinamička alokacija memorije,
5. Mogućnost pamćenja definicija podataka i funkcija na eksternu memoriju,
6. Korisniku je omogućen pristup sistemskim promenljivim.

Programski jezik LOGO je jednostavno i moćno oruđe kojim deca mogu ispitati svet geometrije, matematike i fizike. Ipak, daleko od toga da bude samo za decu, LOGO je jezik koji će mnogim svojim interesantnim funkcijama privući pažnju čak i profesionalnih programera.

FORTH, težak za učenje ali veoma moćan.

Iako postoji već desetak godina uglavnom je do pre par godina bio vezan za korišćenje u kontroli procesa i upravljanju složenim sistemima. Omasovljenjem mikro računara je stekao popularnost iako i dan danas ostaje manje-više egzotičan programski jezik u poređenju sa ostalima.

Neke od najvažnijih karakteristika FORTH-a su kompajlirani i interaktivni kod, skoro potpuna modularnost programa i njegovih delova i gotovo neverovatna prilagodljivost raznim problemima. FORTH se može iskoristiti za bilo šta, od pisanja jednostavne obrazovne igre do pisanja drugog programskog jezika (postoji PASCAL napisan u FORTH-u).

FORTH je relativno lak za početnika ali za potpuno korišćenje njegovog punog potencijala je potrebno odlično poznavanje funkcioni-

sanja računara i operativnog sistema. Kontrolne strukture kojima raspolaže su veoma moćne ali je u tipovima podataka relativno siromašan. Dozvoljava upotrebu proizvoljnih imena velike dužine što olakšava upotrebu smislenih imena za promenljive u programu.

FORTH se razlikuje od drugih jezika i po tome što je u njemu moguće dodavanje novih naredbi koje dobijaju ista prava kao i originalne naredbe koje postoje u definiciji samog sistema. To znači da i ako nema naredbe za kontrolu složenih struktura podataka one mu se mogu ugraditi i zatim koristiti kao da su oduvek bile prisutne.

Aritmetika FORTH-a koristi obrnutu poljsku notaciju i to se u njegovom slučaju može smatrati prednošću jer zahvaljujući njoj FORTH u svakom trenutku zadržava određenu dozu jednostavnosti. Važna osobina ovog jezika je i prenosivost, što znači da nije važno na kom se računaru razvija aplikacija ali je svakako jedna od velikih mana to što posle FORTH-a nije lako učiti novi programski jezik. Osnovni razlog je u tome što se u FORTH-u koristi prilično neuobičajen način obeležavanja.

Kao što smo napomenuli na početku ovog poglavlja, ono i nije trebalo da posluži da naučite da programirate u FORTH-u ili PASCAL-u. Bićemo zadovoljni ako ste stekli približnu sliku o tome koji od ovih programskih jezika može vama da bude koristan i sa kojim se vredi upoznavati.

VI/ MAŠINE GOVORE MAŠINSKI

Za one koji su spremni na najteže — osnove za programiranje u mašinskom kodu (za ZX Spectrum, Commodore 64 i druge)

Ovo poglavlje PZP verovatno je najviše tajanstveno svima koji se po prvi put susreću sa programiranjem. Mašinski jezik ili mašinski kod ili »mašinac«, kako ga od milošte zovu, ustvari bi se teško mogao nazvati programskim jezikom u onom smislu u kojem su to takozvani viši programski jezici, BASIC ili FORTH, na primer.

Znamo da su viši programski jezici smišljeni da bi korisnik mogao jednostavnije da daje instrukcije računaru, ali ono što je lakše i brže nama — za računar je teže. On mora da prevede naredbu datu u nekom višem jeziku na »mašinac« da bi mikroprocesor mogao da je izvrši, a to zahteva vreme. Nasuprot tome, programske naredbe date u »mašincu« direktno se izvršavaju pa su programi brži i zauzimaju manje prostora u memoriji. To je i normalno jer mašinski kod je niz naredbi datih tako da mikroprocesor može odmah da ih izvrši — bez prevođenja. Naredbe su razvijene u obliku najsitnijih koraka koje procesor treba da izvede. Ako bismo to prikazali na nekoj radnji koju treba obaviti, na primer na uzimanju olovke sa stola, upoređenje sa višim jezicima bilo bi ovako:

viši jezik:	mašinac:
— uzmi olovku sa stola	— da li je olovka na stolu
	— koja ruka može da je dohvati

— digni tu ruku
— pomeri je iznad stola
— dovedi je do olovke
— spusti je na olovku
— zgrči palac i kažiprst
— podigni olovku
— privuci olovku k sebi.

Zbog potrebe da se svaka radnja razbije na najsitnije segmente mašinski jezik je težak za rad i zahteva veliku metodičnost i strpljenje — sve mora da se proverava dva, tri puta.

Ukoliko nema velike potrebe — dugi programi se nikada ne pišu u mašincu. Međutim, kod određenih zadataka, na primer da bi se ubrzala radnja neke igre ili za pravljenje posebnih efekata na ekranu — mašinac je nezamenljiv, jer BASIC je desetak puta sporiji i igra nema dovoljnu dinamiku.

To su razlozi što je ovo poglavlje PZP posvećeno mašincu. Namera je da vam mašinac učinimo pristupačnijim i da vam prikažemo kako sami da napravite kratke mašinske potprograme koje ćete koristiti u BASIC-programima.

U mikroračunarima se koriste razne vrste mikroprocesora, tojest proizvođač bira onaj mikroprocesor koji najviše odgovara prema tome kakve radnje i koliko brzo može da ih obavi i, naravno, koliko košta. U mikroračunarima prve generacije koristi se mikroprocesor sa osam bita koji je najsporiji jer u jednom trenutku može da operiše samo sa osam signala. Na tržištu je i sve više računara (poslovnih) koji koriste 16-bitne procesore, a već su tu i »prve laste« sa 32-bitnim procesorima koji bi do početka poslednje decenije ovog veka trebalo da dominiraju tržištem. U kućnim računarima još dugo će »kucati osmobitno srce« pa smo se i opredelili za dva najviše korišćena mikroprocesora u kućnim računarima: Zajlogov Z80 i Intelov 6502. Z80 koristi se u Sinklerovom Spektrumu i ZX81, VIC 20, BBC, Atariju i Galaksiji. Mikroprocesor 6052 pokreće Apple, Oric, Commodore 64 koristi jedan derivat ovog procesora — 6510, koji razume mašinski kod za 6502. Da bismo bili u duhu mašinskog koda ovaj kratki kurs podelićemo na petnaestak koraka.

1. ŠTA JE MAŠINSKI JEZIK (KOD)?

U mašinskom kodu svaka naredba i podataka predstavljeni su binarnim brojevima. U računaru, binarni brojevi predstavljeni su električnim signalima. Kad ima napona to je 1, a kad nema napona to je 0.

Signal i ne-signal imaju isto ime: "bit" što je skraćenica od "binary digit" (binarna cifra). Biti prolaze kroz računar u grupama od 2, 4, 8, 16, 32 (prema broju bita koji mogu da se obrade u jednom trenutku u procesoru računari se i dele na osmobitne, šesnaestobitne, itd.). Mi se bavimo osmobitnim procesorima (Z80 i 6502) tako da dalje govorimo samo o grupama od po osam bita. Grupa od osam bita zove se »bajt«. Svaki bajt, sastavljen od osam signala i ne-signala predstavlja jedan binarni broj koji nosi jednu naredbu ili podatak u mašinskom kodu.

Postoji nekoliko načina za pisanje programa u mašinskom kodu:

a) sve naredbe i podaci dati su binarnim brojevima,

b) naredbe i podaci dati su u heksadecimalnim brojevima, što je kraće i jednostavnije za rad nego sa binarnim brojevima,

c) uz pomoć »asemblera: — programa u kome su sve naredbe date u vidu mnemonika koji se prevode na mašinski kod. Mnemonik je skraćenica smišljena radi lakšeg pamćenja. Na primer mnemonik LD je skraćenika za LOAD (eng. napuni).

Pogledajmo kako izgleda mašinski program za sabiranje 2 i 4:

u heksa-brojevima	u asembleru
3E,02	LD A,02
C6,04	ADD A,04
32,577F	LD (7F57),A
C9	RET

Levi program (program je za Z80) je u heksa-kodu o kome će više biti reči na narednim stranama). Prva dva znaka (3E, C6, itd.) su naredbe u heksa kodu, a iza zareza dati su podaci (02,04) ili adresa (577E). Desni program sastavljen je od mnemonika koji su ekvivalenti za naredbe u heksa-brojevima iz levog primera. Tako je 3E isto što i LD A, C6 je isto što ADD A, itd.

Pojedini računari imaju ugrađen asembler dok kod drugih morate da ga unesete u memoriju da bi se koristio. Neki računari prihvataju heksa-brojeve. Kod drugih morate da koristite poseban program zvani "heksa ubacivač" koji prevodi heksa-brojeve za potrebe procesora. Kasnije ćemo dati jedan takav program.

2. UPOZNAJTE SVOJ RAČUNAR

Osnovu za rad računara predstavljaju čipovi, što je popularno ime za »integralna kola«. Integralno kolo je sinonim za mikroskopski mala električna kola smeštena na prostoru od jednog kvadratnog centimetra (za nekoliko godina ovo će moći da se kaže za ceo računar).

Čipovi su smešteni na jednoj tablici i međusobno povezani električnim provodnicima koji su štampani posebnim postupkom sa jedne ili sa obe strane te tablice.

Postoji nekoliko vrsta čipova u računaru. Jedni su ROM čipovi i u njima su stalno smešteni razni programi za rad računara. Drugi su RAM čipovi i u njih se, kad je računar uključen, odlažu vaši programi i podaci i podaci potrebni računaru za njihovu obradu. Treću vrstu čipova (ima ih po jedan ili dva u računaru) predstavlja mikroprocesor. On sadrži CPU, što je skraćenica za engleski naziv »centralna jedinica obrade«. Tu se obavljaju sve operacije računara, računske operacije, upoređivanje podataka, donose se odluke i koordiniraju sve aktivnosti računara. Informacije za rad CPU sadržane su u ROM-u.

Svaki računar mora da ima sat, ali to nije klasičan sat nego kvarcni kristal koji treperi velikom učestanošću, nekoliko miliona puta u sekundi i tako reguliše tok signala u računaru. Što je veća učestanost treperenja, signali idu kroz računar većom brzinom i on brže obavlja zadatke.

Bajti idu od čipa do čipa »pistama«-provodnicima na štampanom kolu. Postoje tri sistema pista i oni se zovu »bus«. Na primer, adresni bus nosi bajte sa memorijskim adresama između CPU i ROM čipova, a data-bus nosi bajte sa naredbama i podacima između CPU i radne memorije — RAM.

3. MAPA MEMORIJE

Svako mesto u memoriji ima svoj naziv — adresu. Takođe, svaki deo memorije služi za određenu namenu. Grafički prikaz memorije računara sa određenim prostorima prema nameni naziva se mapa memorije.

Memorijske adrese u računaru su predstavljene sa dva bajta, to jest sa 16 bita (ili: 16 signala i ne-signala). Najveća memorija u računaru koji koristi Z80 ili 6502 je 64 kilobajta (zajedno uzete ROM i RAM). To je zakon do koga se lako dolazi ako se zna da je najveći binarni broj koji se može sastaviti od 16 cifara — 65535, pa je to najviša moguća adresa u memoriji. To ujedno daje 65536 mesta u memoriji, obeleženih od 0 do 65535. Na svakom mestu u memoriji nalazi se jedan bajt, 1024 bajta su jedan kilobajt (K ili KB), a 65536 bajta je 64K (65536/1024=64).

Pripremili smo vam opšti izgled mape memorije nekog kućnog raunara koji ima 24K. Inače, ta mapa može biti vodoravna ili uspravna i raspored se može razlikovati od ovoga. Na mapi su date početne adrese za svaki blok memorije i to u decimalnim brojevima i u heksa-

-brojevima koji ispred moraju da imaju neki znak: &, $, %, # (u priručniku vašeg računara je naznačen znak koji on raspoznaje kao najavu heksa-broja).

Ova mesta su vezana za ulaze i izlaze računara	Ulaz/Izlaz	&6000	24576
Tu se drže informacije koje su na ekranu	Memorija ekrana	&5C00	23552
U korisničkom bloku memorije su svi vaši programi i podaci za promenljive veličine. Prostor za promenljive se menja zavisno od broja podataka koje one imaju i on može biti veći ili manji	Odlaganje promenljivih . . . . . . Korisnička memorija RAM		
Blok rezervisan za potrebe operativnog sistema sadrži podatke koji su mu potrebni tokom izvršenja programa: pozicija kursora, broj programskog reda, koja dirka je pritisnuta, itd.		&2E00	11776
U BASIC-bloku se nalazi program koji procesoru prevodi naredbe iz BASIC programa u binarne.	Rezervisano za potrebe operativnog sistema	&2400	9216
Blok za operativni sistem sadrži programe koji upravljaju radom računara. Svi su u mašinskom kodu. Jedni upravljaju numeričkim operacijama, drugi prazne ekran, treći nalaze slučajni broj, ukratko oni kažu kako se radi sve ono što treba uraditi.	BASIC ROM	&0400	1024
	Operativni sistem ROM	&0000	0

Mapa memorije sadrži ROM i RAM. U ROM-u se nalazi program za prevođenje BASIC-naredbi i operativni sistem, a ostali blokovi su u RAM-u. Početne adrese date u ovoj mapi će se promeniti ako računaru dodate više memorijskih čipova (sve to imate napisano u računarskom priručniku).

Blok za operativni sistem je i sam podeljen na manje blokove koji su rezervisani za određene potrebe:
Korisnička grafika (UDG) — ako generišete sopstvene grafičke simbole, oni su smešteni u tom malom bloku,
Baferi — tu se privremeno odlažu podaci koji dolaze sa tastature ili se šalju na štampač ili u eksternu memorijsku jedinicu,
Mašinski stak — CPU koristi ta mesta u memoriji da na njih odloži potrebne adrese za vreme izvršavanja programa u mašincu,
BASIC-stak — zove se još i GOSUB-stak jer se u taj prostor odlažu brojevi programskih redova upotrebljeni u naredbama GOSUB i GOTO,
Kalkulatorski stak — u njega CPU privremeno odlaže brojeve tokom numeričkih operacija,
Sistemske promenljive — tu je niz adresa u memoriji na koje CPU odlaže informacije o tome šta se događa u računaru. Na primer, tu se beleži pozicija kursora u tom trenutku, ili se tu nalaze adrese na kojima su odložene vrednosti promenljivih koje program koristi.

U ovom delu su nužna još dva objašnjenja. Prvo se odnosi na reč »stak« koju ste videli nekoliko redova ranije. Ta reč je zamena za englesko »stack« (stog, plast) koje se koristi da bi se označila jedna vrlo jednostavna naprava koju ste sigurno imali prilike da vidite u prodavnicama ili restoranima. To je jedna daščica probodena dugim šiljkom sa donje strane i koristi se da se na nju nabodu računi i slični papirići, da se ne bi pomešali ili izgubili. Kod nabadača postoji jedan nepromeljivi zakon, a to je da se prvi nabodeni papir skida poslednji sa šiljka, a poslednji nabodeni skida se prvi. Isti princip primenjuje se i u raučnarima, kod kojih je vrlo važno da se zna kada se koji podatak koristi. Za to se koristi »elektronski nabadač«, a to je niz adresa na koje se odlažu podaci redom tako da se uvek prvo koristi onaj koji je poslednji odložen, a prvi odloženi podatak CPU će koristiti na kraju nekog niza operacija. Prilikom programiranja u mašincu morate voditi računa da se ne ubodete na neki računarov nabadač time što ćete zaboraviti na zakon: poslednji unutra — prvi van.

Druga napomena odnosi se na »stranice memorije«. Da bi se lakše snalazilo u memoriji, ona je podeljena na stranice. Svaka stranica ili strana (kao u knjizi) memorije mikroračunara ima 256 mesta, a četiri strane čine jedan kilobajt ($4 \times 256 = 1024$). Mesta od 0 do 255 najčešće se zovu nulta strana memorije. Pojedini blokovi počinju na početku nove strane. Na prikazanoj mapi korisnička memorija počinje na početku 45. strane, računajući prvu stranu kao nultu.

4. HEKSADECIMALNI BROJEVI

Pre nego što nastavimo da se bavimo mašincem potrebno je da se malo pozabavimo heksa-brojevima, jer se u programima pisanim u mašincu svi brojevi i adrese pišu uvek na ovaj način:

Desetni	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Heksa	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Kao što znamo, desetni brojevi imaju bazu deset. Heksa-brojevi imaju bazu 16. Da biste napisali broj veći od 15 (F) uzimate dve (ili više) cifara, slično principu kod desetnih brojeva većih od 9. Vrednost svake cifre zavisi od mesta u broju. Uzmimo za primer broj 1226:

Desetni				Heksa		
1000	100	10	1	256	16	1
1	2	2	6	4	C	A

Pretvaranje iz heksa u desetni broj obavlja se tako što prvo utvrdite desetni ekvivalent za svaku heksa cifru. Pomnožite te ekvivalente vrednošću njihovog mesta u heksa broju i saberite rezultate. Ništa lakše od toga(!):

	256	16	1	mesto u broju
	4	C	A	heksa broj
&4CA = 1226	4	12	10	desetni ekvivalent
	x256	x16	x1	x mesto u broju
	1024 +	192 +	10 = 1226	zbir rezultata

Pretvaranje desetnog broja u heksa broj obavlja se tako što prvo taj broj podelimo sa 256 da vidimo koliko se puta 256 sadrži u tom broju. Ostatak delimo sa 16, da bismo dobili koliko se puta druga heksa cifra sadrži u ostatku. Ostatak iz ovog delenja daje nam broj jedinica u heksa broju. Na kraju, sva tri rezultata pretvorimo u heksa cifre:

1226:256=4 . . . . . 4 je 4 u heksa
ostatak je 202
202:16=12 . . . . . 12 je C u heksa
ostatak je 10 . . . . . 10 je A u heksa
dakle, 1226 je 4CA u heksa brojevima

Kod adresa u heksa brojevima bitno je da se zna da su dve leve cifre oznaka stranice memorije na kojoj je adresa, a druge dve cifre pokazuju poziciju na stranici. Pretvaranje ide ovako:

Heksa u desetne	Desetne u heksa
adresa &5C64	adresa 23652
stranica=&5C=92	stranica=23652:256=92
pozicija=&64=100	pozicija=100 (ostatak delenja)
92×256=23552+100	92:16=5 i ostatak 12=&5C
=23652	100:16=6 i ostatak 4=&64
&5C64=23652	23652=&5C64

5. PEEK i POKE

Ove dve reči koje spadaju u BASIC veoma su važne, jer prečicom stižu do memorije, jedna da bi se videlo šta je na nekoj adresi, a druga da bi se promenio njen sadržaj. One se kod većine mikroračunara daju sa desetnim vrednostima adrese u memoriji.

PEEK (čita se: PIK) se koristi da bi se reklo računaru da pogleda na datu adresu i vidi njen sadržaj. Ako se koristi sa PRINT, na ekranu će se pojaviti brojka koja predstavlja sadržaj adrese:

```
PRINT PEEK 23457
48
LET A=PEEK 16200: PRINT A
113
```

POKE (čita se: POUK) se koristi da bi se dao nalog računaru da izmeni sadržaj neke adrese. To znači da se POKE može koristiti samo za adrese u RAM, a uz pomoć PEEK može se saznati sadržaj adrese i u ROM i u RAM memoriji. POKE ima oblik:

```
POKE 23609,50
```

što znači nalog računaru da na adresu 23609 smesti brojku 50. Rezultat se može odmah utvrditi ako naredimo:

```
PRINT PEEK 23609
50
```

Brojevi koje unosite kao novi sadržaj adrese moraju da budu između 0 i 255, pošto je to najveći broj koji se može izraziti binarnim brojem sa osam cifara (jedan bajt).

Eksperimentišite malo sa PEEK i POKE. Ako vam se desi da promenite sadržaj neke adrese na koju je odložena vrednost neke sistem-

ske promenljive — nemojte se uplašiti. Samo isključite računar i ponovo ga uključite: sve je na svom mestu! Proverite sa PEEK, ako ne verujete.

Na prethodnim stranama bilo je reči o heksa brojevima, sad smo upoznali dve naredbe, a sve se to vrti oko brojeva. Zašto samo brojevi?

Kad naredite računaru da na ekranu prikaže sadržaj neke adrese, rezultat je uvek broj između 0 i 255. To je zbog toga što jedna adresa sadrži samo jedan bajt. Postoji, dakle, samo 256 različitih bajta u računarskom kodu i svaki od njih može imati različito značenje za računar, zavisno od upotrebe. Na primer, binarni broj 00110000 (decimalni: 48) može biti kod neke naredbe, ili kod nekog znaka ili slova, a može biti i deo koda neke adrese u memoriji (svaka adresa sastoji se od dva bajta).

Većina računara koristi ASCII kod (izgovara se: Aski) u kojem je svaki grafički znak i simbol obeležen brojem. Da biste videli kojem znaku pripada koji broj pogledajte u priručnik svog računara ili napišite mali program koji će na ekranu prikazati broj u aski kodu za svako slovo koje pritisnete na tastaturi.

6. UNUTAR CPU

Kao što smo naglasili ranije, bez obzira na složenost zadatka koji postavite računaru, on će morati da ga razloži na najsitnije operacije i tako će ga izvršavati. Sve što se radi u računaru svodi se na pozivanje bajta sa instrukcijama i podacima iz memorije i, potom, izvršavanje instrukcija u CPU.

Unutar CPU postoje tri glavna područja: registri — u kojima se drže bajti sa podacima tokom obrade; ALU — aritmetičko-logička jedinica gde se bajti sabiraju, oduzimaju ili upoređuju; kontrolna jedinica koja organizuje sve te aktivnosti.

Naredbe koje CPU može da izvrši veoma su jednostavne. CPU može da pozove bajte iz memorije i smesti u registre, da pomera bajte iz jednog registra u drugi, obrađuje ih u ALU i odlaže rezultate u memoriju. Čak i najjednostavniji zadatak kao sabiranje dva broja i prikazivanje rezultata na ekranu uključuje više od stotinu jednostavnih koraka, a CPU može da izvrši više od pola miliona takvih operacija u jednoj sekundi.

Za svaku od tih operacija logička jedinica poziva jedan instrukcioni bajt iz ROM ili RAM, puni registar bajtom podataka, a onda izvršava operaciju koja je naložena instrukcijom. Uz pomoć mašinca vi možete da naredite CPU šta da radi sa bajtima u registrima, ali ALU

i kontrolna jedinica su van vašeg domašaja i obavljaju svoje zadatke automatski, po nalozima koje im je odredio proizvođač procesora.

Pre nego što se upoznate sa registrima naša dva procesora, pomenimo samo da je jedina razlika između čipova Z80 i 6502 u tome što Z80 ima više registara. To znači da se bajti mogu privremeno odložiti u CPU, a kod 6502 moraju biti vraćeni u memoriju.

Z80 Registri:

A — akumulator, najvažniji registar u CPU i u njega se odlažu bajti na njihovom putu u/iz aritmetičko-logičke jedinice. On može da drži samo po jedan bajt;

B, C, D, E, H i L — registri opšte namene u koje se odlažu bajti na putu iz/u memoriju. Svaki može da drži samo po jedan bajt, ali se mogu koristiti u parovima, na primer: BC, DE, HL i tada drže po dva bajta;

F — signalni registar. On drži osam bita, od kojih se koristi šest. Svaki bit služi kao indikator. Na primer, prenosni signal se postavlja na 1 ako je odgovor veći od 255 i ne može da stane u jedan bajt. Ili, signal znaka koji pokazuje da li je broj pozitivan ili negativan;

SP — brojač staka. To je 16-bitni registar u koji se odlaže adresa poslednje stavke na mašinskom staku — mestu na koje CPU privremeno odlaže podatke;

IX i **IY** — indeksni registri i svaki od njih drži po 16 bita koji se koriste u određenim instrukcijama da daju adresu bajta u memoriji;

PC — programski brojač. To je 16-bitni registar i on drži adresu bajta koji će biti sledeći pozvan iz memorije. Broj u programskom brojaču povećava se za jedan svaki put kad se izvrši neka instrukcija.

6502 Registri

A — akumulator, služi za odlaganje bajta na njihovom putu u/iz ALU. Isti je kao kod Z80 i može da drži jedan bajt;

X i **Y** — indeksni registri koji mogu da budu upotrebljeni i kao registri opšte namene, za privremeno odlaganje bajta;

P — registar statusa procesora, ima istu funkciju kao signalni registar kod Z80. Sadrži osam bita, od kojih se koristi sedam i svaki od njih se postavlja na 1 da označi određeni uslov, kao što je da li je broj pozitivan ili negativan;

PC — programski brojač. On radi na isti način kao PC registar kod Z80;

S — brojač staka. On sadrži adresu poslednje stavke na staku. Kod 6502 to je 8-bitni registar. Deveti bit se drži uvek na 1, jer je kod 6502 stak uvek na prvoj stranici memorije. Broj u brojaču staka pokazuje poziciju na stranici.

7. DAVANJE INSTRUKCIJA CPU

Instrukcije CPU se mogu dati samo iz liste instrukcija za taj procesor. Znači, za sve računare koji imaju Z80 ili Z80A procesor važe samo instrukcije sa njihove liste, a računari sa procesorom 6502, 6502A ili 6510 moraju da koriste svoju listu instrukcija. Najveći deo mašinskih instrukcija sastoji se iz dva dela: a) opkod (jedna od instrukcija sa liste i b) operand (ova reč označuje objekt operacije).

Opkodovi se mogu pisati kao mnemonici. Na primer, LD A za Z80 i LDA za 6502 znači »unesi bajt u akumulator«. Isti opkodovi u heksa brojevima za Z80 su 3E, a za 6502 A9. Mnemonici su lakši za razumevanje ali se ne mogu unositi u računar bez asemblera (programa koji ih prevodi u mašinski kod). Pogledajmo na primeru programa za sabiranje dva broja kako to izgleda. Prvo se prvi broj unosi u akumulator. Potom se drugi broj dodaje prvom u akumulatoru, da bi se, na kraju, rezultat odložio u memoriju:

Z80 mnemonici	Značenje
LD A, broj	Unesi broj u A (akumulator)
ADD A, broj	Dodaj drugi broj u A
LD (adresa), A	Unesi sadržaj A na određenu adresu

Kao što vidite, opkod i operand za Z80 uvek se odvajaju zarezom, a adresa se uvek daje u zagradi. Program izgleda ovako:

mnemonici	heksa
LD A, &02	3E,02
ADD A, &04	C6,04
LD(&7F57), A	32,577F

6502 mnemonici	Značenje
LDA broj	Unesi broj u A (akumulator)
ADC broj	ADC je mnemonik za instrukciju »dodaj pažljivo«
STA adresa	

Za 6502 program bi izgledao ovako:

mnemonici	heksa
LDA # &02	A9 02
ADC # &04	69 04
STA &7F57	8D 577F

Zapamtite, vi koji koristite 6502, da se znak # koristi uvek kad su u pitanju podaci. Nešto kasnije objasnićemo inverzan oblik heksa-broja koji označuje adresu na koju se odlaže rezultat.

U priručniku vašeg računara data je lista instrukcija za procesor koji računar koristi. Uporedo su dati mnemonici i heksa-kodovi i morate biti veoma pažljivi da nešto ne pomešate, jer postoje različiti heksa-kodovi jedne iste instrukcije zavisno od toga da li je operand podatak, adresa ili ime registra. Kad je operand podatak to se zove »direktno adresiranje«. Ako je operand adresa na kojoj je podatak odložen, zove se »apsolutno adresiranje«. Analogno tome, koriste se pojmovi »indirektno adresiranje« i »implicitno adresiranje«.

Još jedna napomena: pošto se programi u mašincu pišu korišćenjem heksa brojeva, kod pisanja adresa koriste se dva para heksa-cifara (jer adrese zauzimaju dva bajta). Treba uvek imati na umu da se cifre pišu obrnutim redom od očekivanog. Normalno bi bilo da par koji obeležava stranicu memorije na kojoj je adresa bude prvi, ali to pravilo ovde ne važi. Prvo se piše par heksa-cifara koji označava mesto na stranici memorije i on se zove »bajt nižeg reda«. Potom dolazi par heksa-cifara koji znači stranicu memorije i naziv mu je »bajt višeg reda«. Znači, prvo se piše drugo a drugo se piše prvo.

8. SLOBODAN PROSTOR U RAM

Pre nego što unesete i pokrenete mali mašinski program za sabiranje, predstoji još nekoliko stvari. Prvo, morate znati da pri unošenju programa u BASIC-u poseban program za prevođenje smešta vaš program u korisnički deo RAM. Kad unosite mašinski progrram on mimoilazi program za prevođenje, pa morate sami da odredite mesto na koje će biti smešten. Dakle, morate odrediti prostor u RAM u kojem se vaš program neće pomešati sa postojećim informacijama ili podacima.

Prethodno morate tačno da odredite koliko vam prostora treba za smeštanje programa u mašincu. Do toga se dolazi lako — brojanjem parova heksa cifara. Svaki par heksa cifara zauzima jedan bajt. Na primer, program za sabiranje ima sedam bajta. Najveći broj mašinskih programa je kratak tako da će rezervacija sto bajta prostora u memoriji biti sasvim dovoljna za vaše prve programe.

Uobičajeno mesto za odlaganje mašinskih programa je vrh korisničke memorije, mesto gde se odlažu BASIC-programi. Naravno, morate biti sigurni da se dva programa neće pomešati (što, ako se desi, može da se razreši jedino isključivanjem i ponovnim uključivanjem računara). To se postiže snižavanjem vrha korisničke memorije. Ova tačka memorije zove se RAMTOP ili HIMEM ili samo vrh memorije.

Računar ima zapisanu adresu za RAMTOP (ili HIMEM) među sistemskim promenljivima. Način promene te adrese je različit kod pojedinih računara tako da to morate da saznate iz priručnika. Među-

tim, postoji nekoliko opštih pravila. Adresa RAMTOP zauzima dva uzastopna mesta među sistemskim promenljivim. Prvi bajt znači poziciju na određenoj stranici memorije, a drugi bajt označuje stranicu memorije (kao što rekosmo, prvo dolazi drugo, a drugo dolazi prvo — kad je reč o adresi). Da ne biste razmišljali o pretvaranju desetnih u heksa brojeve, nađite adresu vrha memorije u priručniku i primenite postupak koji je predviđen za promenu sadržaja. Na primer:

CLEAR adresa vrha memorije — 100 (Spectrum)
HIMEM adresa vrha memorije — 100 (Orik)

Znači, posle naredbe pišemo u desetnom obliku adresu vrha memorije (koju smo našli u priručniku) umanjenu za 100 (to je broj mesta u memoriji koja želimo da rezervišemo za mašinski program). Izvršenjem te naredbe snižen je vrh memorije za 100 mesta i time rezervisan prostor od 99 bajta za mašinski program koji počinje od adrese iznad vrha korisničke memorije. Brojku 100 zamenićemo nekom drugom, ako to dužina mašinskog programa zahteva.

Postoje i drugi prostori u memoriji koji se mogu koristiti za odlaganje mašinskih programa, pod uslovom da se ne koriste za osnovnu namenu. Na primer, ako program nećete snimati, može se odložiti u kasetni bafer. Drugi mogući prostor je onaj namenjen generisanju grafičkih simbola od strane korisnika (takozvani UDG). U priručniku su adrese tih prostora ili drugih koje proizvođač preporučuje, a mnogo korisnih saveta ima u časopisima koji objavljuju korisničke programe.

9. UNOŠENJE I POKRETANJE PROGRAMA

Pre nego što razmotrimo način unošenja i pokretanja mašinskog programa potrebno je obaviti još dve pripremne radnje:

Izbor adrese za rezultat — podaci koji se dobiju izvršenjem mašinskog programa nazivaju se »bajti podataka«. Mora se voditi računa da se bajti podataka odlože u deo memorije u kojem neće doći do mešanja sa samim programom. Najbolje mesto je na samom početku prostora koji ste rezervisali za mašinac. Na primer, ako ste vrh memorije pomerili na adresu 16000 onda će prva adresa rezervisana za mašinac da bude 16001. Tu ćete odložiti rezultat, a program će početi od adrese 16002. Nužno je da se adresa iz desetnog prebaci u heksa oblik i takva unese u program;

Naredba RETURN — na kraju svakog programa u mašincu mora da stoji naredba za povratak u glavni program i to RET (za Z80) ili RTS (za 6502).

Heksa-ubacivač je naziv za program koji se koristi kod većine mikroračunara, jer je sadržaj adrese u desetnom obliku. Taj program

pretvara svaki bajt mašinskog koda u desetni broj i, potom, smešta ga (POKE) u memoriju. Daćemo primer takvog programa sa objašnjenjima:

```
10 PRINT "POČETNA ADRESA?"
20 INPUT A ———————————————— A je prva adresa mašinskog programa
30 LET B=0 ———————————————— B je brojač
40 READ H$ ———————————————— uzima prvi par heksa cifara za H$
50 IF H$="KRAJ" THEN GOTO 180 — provera da li je kraj podataka
60 IF LEN(H$)<>2 THEN GOTO 170 — provera da li H$ ima dve cifre, ako nema, ide na 170

70 LET X=(ASC(H$)—48)*16
80 IF ASC (H$)>57 THEN LET      — pretvara prvu heksa cifru u desetnu i odlaže je u X
   X=(ASC(H$)—55)*16
90 LET Y=ASC(RIGHT$(H$,1))
100 IF Y<58 THEN LET X=X+Y—48   — pretvara drugu heksa cifru u desetnu Y i sabira sa X
110 IF Y>57 THEN LET X=X+Y—55

120 IF X<0 OR X>255 THEN GOTO 170 - provera da li je desetni broj u X između 0 i 255
130 POKE A+C,X ———————————— pri C=0 odlaže X na adresu A
140 LET C=C+1 ———————————— povećava C za 1, pa desetni oblik narednog heksa koda odlaže na adresu A+1
150 GOTO 40 ————————
155 REM PRIMER PODATAKA
160 DATA EF,F6,E2,A9,KRAJ       — Stavite tu svoje heksa kodove i signalnu reč KRAJ
170 PRINT "NE VALJA PODATAK" —  štampa rečenicu ako nađe neodgovarajuće podatke u 60,120

180 STOP
```

Ako imate Spectrum, zamenite naredbu ASC sa CODE i stavite svaki par heksa cifara u navodnike. Takođe promenite red 90 da glasi:

90 LET Y=CODE(H$(2 TO)).

Da biste proverili ovaj heksa ubacivač pokušajte sa programom za sabiranje. On nije posebno zanimljiv i služi samo kao primer. U redu 160 zamenite podatke sledećim (zavisno od procesora):

Z80: 160 DATA 3E,02,C6,04,32,nb,vb,C9,KRAJ
6502: 160 DATA A9,02,69,04,8D,nb,vb,60,KRAJ

Skraćenicu nb (bajt nižeg reda) zamenite bajtom koji označuje položaj adrese na stranici memorije, a skraćenicu vb (bajt višeg reda) zamenite bajtom koji označuje stranicu memorije na kojoj je adresa rezultata.

Kada unesete heksa ubacivač u memoriju, kucajte RUN. Na ekranu će se pojaviti natpis POČETNA ADRESA i prompt koji zahteva da unesete prvu adresu vašeg mašinskog programa (prvu posle one koju ste rezervisali za rezultat). Adresu unesite kao desetni broj. Posle toga će program sam uzeti podatke iz reda 160 i odložiti ih na naredne adrese.

Sad dolazi trenutak pokretanja mašinskog programa. Naredba za pokretanje se razlikuje od računara do računara. Neki koriste CALL, drugi PRINT USR ili SYS, a posle toga kod svih sledi desetna adresa prvog bajta programa. Kad računar primi tu naredbu on ide na zadatu adresu i počinje izvršavanje mašinskog koda. Kad budu izvršene sve operacije, rezultat biva odložen na datu adresu.

Da biste proverili rezultat koristite naredbu PRINT PEEK uz koju kucate desetni oblik adrese koju ste rezervisali za rezultat. Na primer: PRINT PEEK (16001). Rezultat se dobija u desetnom obliku.

Pokušajte sad sami da napravite neki program za sabiranje malih brojeva (32 i 12 ili 77 i 18 i 37). Pri tome koristite ovu listu za proveru:

Lista koraka
u mašinskom programiranju

1. *Napišite svoj program u asembleru i pretvorite sve podatke u heksa brojeve*
2. *Utvrdite heksa kod za svaki mnemonik (iz liste u priručniku)*
3. *Stavite na kraj naredbu za povratak na glavni program*
4. *Izbrojte koliko program ima bajta i rezervišite prostor u RAM*
5. *Zabeležite na papiru adrese za bajte podataka i početnu adresu programa*
6. *Pretvorite adrese za bajte podataka u heksa oblik i unesite ih u program (ne zaboravite na obrnuti redosled heksa cifara)*
7. *Unesite heksa ubacivač kucanjem ili sa trake i dopunite red 160 svojim podacima (ne zaboravite signalnu reč KRAJ)*
8. *Pokrenite heksa ubacivač sa RUN i unesite decimalnu adresu za početak programa*
9. *Pokrenite mašinski program (koristeći odgovarajuću naredbu iz priručnika i decimalnu adresu početka mašinskog programa).*
10. *Proverite rezultat sa PRINT PEEK (i adresa koju ste rezervisali za rezultat).*

10. KORIŠĆENJE BAJTA IZ MEMORIJE

Ponekad ćete za mašinski program koristiti podatke koji se već nalaze u memoriji. U tom slučaju operand instrukcije biće adresa na kojoj će računar naći podatak koji će koristiti u izvršenju programa. Kada se instrukcija sastoji od opkoda i podatka to se naziva neposredno adresiranje. Kada je, pak, operand adresa onda se to naziva apsolutno (ili direktno, ili produžno) adresiranje. To su samo dva od mogućih načina adresiranja i svaka instrukcija ima različit heksa kod zavisno od primenjenog načina adresiranja. Da se vratimo na korišćenje podataka (bajta) iz memorije. Uporedite naš program za sabiranje sa prethodnih strana (koristio je neposredno adresiranje) sa narednim koji koristi apsolutno adresiranje:

Z80

Mnemonici	Heksa kod	Značenje
LD A,(1.adresa)	3A,1.adresa	Unesi broj sa 1. adrese u reg. A
LD B,A	47	Unesi broj iz reg. A u reg. B
LD A,(2.adresa)	3A,2.adresa	Unesi broj sa 2. adrese u reg. A
ADD A,B	80	Dodaj broj iz reg. B akumulatoru
LD(3. adresa), A	32,3.adresa	Odloži sadržaj reg. A na 3. adresu
RET	C9	Vrati se

Da biste sabrali dva broja iz memorije prvo ih unesite u registre. Za to možete koristiti akumulator (A) i registar B. Podaci ne mogu da se unesu direktno iz memorije u registar B nego ih prvo unesete u akumulator, pa iz njega prenesete u registar B.

6502

Mnemonici	Heksa kod	Značenje
LDA 1. adresa	AD 1.adresa	Unesi broj sa 1. adrese u reg. A
ADC 2.adresa	6D 2.adresa	Dodaj broj sa 2. adrese u reg. A
STA 3.adresa	8D 3.adresa	Odloži sadržaj reg. A u 3. adresu
RTS	60	Vrati se

Pre pokretanja ovog programa (sledite listu sa prethodne strane) treba sa POKE da odložite u memoriju dva broja koja ćete sabirati. Koristite adrese na početku rezervisanog prostora da biste odvojili bajte podataka od programa. Te adrese pretvorite u heksa oblik i unesite u program (to se odnosi i na adresu za rezultat). Rezultat ćete dobiti sa PRINT PEEK (3. adresa).

11. RAD SA VELIKIM BROJEVIMA

Prethodni brojevi za sabiranje ne mogu u zbiru da budu veći od 255, jer to je najveći broj koji se može predstaviti sa osam bita. Pre nego što savladamo i taj korak u mašinskom programiranju moraćemo da se pozabavimo binarnim brojevima i prenosnim signalom.

1	1	1	1	1	1	1	1	binarni oblik
x128	x64	x32	x16	x8	x4	x2	x1	
128	+ 64	+ 32	+ 16	+ 8	+ 4	+ 2	+ 1 = 255	desetni oblik

U binarnom broju svaka cifra ima vrednost koja je dvostruko veća od vrednosti cifre sa njene desne strane. Prva cifra pokazuje koliko jedinica ima broj; druga cifra pokazuje koliko ima dvojki; treća cifra — koliko ima četvorki (i tako redom: 8, 16, 32, 64, 128). Da bi se binarni broj pretvorio u decimalni treba pomnožiti svaku cifru sa vrednošću njenog mesta i sabrati rezultate. Na primer, 00101110=46 ili 10000111=135. Proverite.

U računaru, brojevi veći od 255 odloženi su u dva bajta, bajt višeg reda i bajt nižeg reda, kao što se radi sa adresama. Bajt višeg reda pokazuje koliko puta se 256 sadrži u tom broju, a bajt nižeg reda je ostatak delenja. Kao sa adresama, računar prvo uzima bajt nižeg reda, pa bajt višeg reda i tako ih i odlaže u memoriju.

17840 : 256 = 69, a ostatak je 176
176 je bajt nižeg reda
69 je bajt višeg reda

Kao što vidite, da biste dali računaru broj veći od 255, morate dobiti vrednost svakog bajta. Prvo, broj podelite sa 256 i tako dobijate vrednost bajta višeg reda. Ostatak uzimate kao bajt nižeg reda. Ako takav broj hoćete da koristite u mašinskom programu onda svaki bajt pretvorite u heksa oblik (o čemu je bilo reči ranije).

12. PRENOSNI SIGNAL

Prenosnim signalom nazivamo jedan bit u signalnom registru koji se koristi da bi naznačio da je rezultat neke operacije veći od 255 i ne može da stane u jedan bajt. Kada se to dogodi računar automatski stavlja prenosni signal na 1 i to se naziva postavljanje prenosnog signala. Kad mu računar dodeli vrednost 0 to se naziva čišćenje prenosnog signala. Prenosni signal možete zamisliti kao deveti bit u binarnom broju koji je rezultat sabiranja.

Z80

Za Z80 postoje dve instrukcije za sabiranje: ADD i ADC. ADD daje nalog procesoru da sabere dva broja ali da ne uzima u obzir prenosni signal zaostao iz prethodnih operacija. Ako je rezultat veći od 255 procesor će postaviti prenosni signal (1), a ako nije, dodeliće mu vrednost 0. Instrukcija ADC daje nalog procesoru da sabere dva broja i uzme u obzir i stanje prenosnog signala iz prethodne operacije, da bi mu, zavisno od rezultata nove operacije dodelio 1 ili 0. Ako se obavlja niz operacija bolje je koristiti ADD za prvo sabiranje, da se izbegne uticaj nekog ranijeg programa, a za sve naredne operacije treba koristiti ADC jer postoji mogućnost da je u nekoj od njih već postavljen prenosni signal.

6502

Pošto 6502 ima samo jednu instrukciju za sabiranje i to ADC, on uvek u operaciju uključuje sadržaj prenosnog signala. Zbog toga je neophodno u program uneti instrukciju CLC (očisti prenosni signal) pre bilo kakvog sabiranja.

13. PROGRAM SA VELIKIM BROJEVIMA

Recimo da smo namerili da saberemo (opet ta dosadna sabiranja!) dva veća broja: 347 i 812. Prvo treba da dobijemo bajt nižeg i višeg reda za svaki od brojeva i da ih odložimo u memoriju uz pomoć POKE:

347:256=1 ostatak je 91	91 — bajt nižeg reda
	1 — bajt višeg reda
812:256=3 ostatak je 44	44 — bajt nižeg reda
	3 — bajt višeg reda

Odložimo to na rezervisane adrese u memoriji:

adrese:	W	W1	X	X1	Y	Y1	Z
sadržaj:	91	1	44	3			

Ne zaboravite da za svaki broj bajt nižeg reda ide prvi, a za njim ide bajt višeg reda. Tri adrese Y, Y1 i Z su rezervisane za rezultat (jedna za bajt nižeg reda, jedna za bajt višeg reda i Z za mogući prenosni signal).

Z80

Sabiranje kod Z80 je jednostavno jer se registri mogu koristiti u parovima, gde svaki par drži oba bajta jednog broja: registri H i L (u paru HL), a B i C (u paru BC). Pošto ne koristimo akumulator (registar A), koristićemo HL za sabiranje.

Mnemonici	Heksa kod
LD HL,(adresa W)	2A,adresa W
LD BC,(adresa X)	ED 4B,adresa X
ADD HL,BC	09
LD(adresa Y),HL	22,adresa Y
LD A,&0	3E,0
ADC A,&0	CE,0
LD(adresa Z),A	32,adresa Z
RET	C9

Da bi program mogao da se izvrši potrebno je uneti heksa oblik adresa W, X, Y i Z (ne zaboravite da obrnete cifre). Inače, kad registre koristite u parovima, dovoljno je navesti prvu adresu. Računar će automatski bajt sa naredne adrese uneti u drugi registar u paru.
Peti, šest i sedmi red gornjeg programa služe za proveru prenosnog signala. Pošto sadržaj prenosnog indikatora ne može neposredno da se unese ni u registar ni u memoriju, jedini način provere je da se izvede kontrolna operacija. Da bi se to obavilo prvo se unese 0 u akumulator, a potom se doda 0 tako da se uzme u obzir stanje prenosnog signala. Ako je on bio postavljen prethodnom operacijom, akumulator će sada sadržati 1, a rezultat se odlaže na adresu Z (sedmi red).

6502

Mnemonici	Heksa kod
CLC	18
LDA adresa W	AD adresa W
ADC adresa X	6D adresa X
STA adresa Y	8D adresa Y
LDA adresa W1	AD adresa W1
ADC adresa X1	6D adresa X1
STA adresa Y1	8D adresa Y1
LDA#&0	A9 00
ADC#&0	69 00
STA adresa Z	8D adresa Z
RTS	60

Rezultat je odložen kao tri bajta. Bajt nižeg reda (adresa Y) pokazuje broj jedinica. Bajt višeg reda (adresa Y1) pokazuje broj 256-ica. Ovoga puta prenosni signal (adresa Z) pokazuje broj 65536-ica. Rezultat se može dobiti na ekranu sa:

```
PRINT PEEK(adresa Y)+((PEEK(adresa Y1)*256)+
((PEEK(adresa Z)*65536))
```

14. SKOKOVI I GRANANJE

Nalog računaru da pređe na instrukciju koja je u drugom delu mašinskog programa naziva se grananje. Postoje tri tipa:

— skok, nalog procesoru da ide na određenu adresu,
— potprogram, slično kao u BASIC-u,
— uslovno grananje, izvršenje skreće ako je zadovoljen uslov.

Važan učesnik u grananju je programski brojač. To je posebni 16-bitni registar koji drži adresu instrukcije koju procesor treba sledeću da izvrši. Procesor čita broj u programskom brojaču i ide na tu adresu da bi uzeo sledeću instrukciju. Tada se programski brojač uvećava za jedan i pokazuje narednu adresu za rad. Kada naložite procesoru da skoči ili skrene na određenu adresu, ta adresa biva stavljena u programski brojač i računar dalje uzima sadržaj adresa počev od nje. Opkod za skok u Z80 je JP, a za 6502 je JMP. Na primer, JP &7F57 i JMP &7F57.

Do potprograma u mašinskom programu računar ide na osnovu instrukcije »CALL adresa« za Z80 (USR u ZX Spektrumu) ili »JSR adresa« za 6502. To je principijelno identično kao u BASIC-u i na kraju je nužna instrukcija za povratak na glavni mašinski program (RET kod Z80 i RTS kod 6502). Kad naložite procesoru da ide na potprogram adresa početka potprograma biva stavljena u programski brojač. Sadržaj brojača (adresa koja se nalazi uz CALL ili JSR) se odlaže ili »nabada« na stak. Kad procesor izvršavajući instrukcije iz potprograma stigne do RET (ili RTS) on uzima poslednju stavku sa staka i šalje je u programski brojač. To je adresa posle koje je prešao na potprogram i tako se izvršavanje nastavlja u glavnom toku programa.

U uslovnom grananju računar proverava jedan od bita u signalnom registru i onda, zavisno od rezultata, skreće tok izvršenja ili nastavlja dotadašnjim tokom. Svaki od tih bita ima svoj naziv i određenu funkciju: bit znaka (S za Z80, N za 6502), bit jednakosti (V ili P/V), nulti bit (Z) i prenosni signal (C). U vašem priručniku su instrukcije za uslovno grananje koje procesor vašeg računara koristi.

Kada date računaru instrukciju za premeštanje (grananje) programa uz koju stoji broj, računar ga sabira ili oduzima od programskog brojača i tako dobija adresu na koju ide. Broj za premeštanje stavlja se u program. Evo dva kratka primera za 6502 (princip je isti za Z80):

```
LDA adresa
CMP #&FF
BNE          | LDA| CMP| FF|BNE| 03| STA| nb| vb| RTS|
STA adresa                      |_____________________↑
RTS
```

Broj za premeštanje je 3, a u narednom primeru taj broj je —6.

```
LDA #&00
ADC #&01
CMP #&FF | LDA| 00| ADC| 01| CMP| FF| BNE|—6| RTS|
BNE                  ↑_____________________________|
RTS
```

Skokovi mogu biti unapred i unazad. Kod skokova unapred broj za premeštanje se pretvori u heksa oblik i unese u program. Nešto je složenije kod skoka unazad (kao i sa ljudima!) jer je u tom slučaju broj za premeštanje negativan, a nema načina da se to pokaže u osmobitnom binarnom broju. Umesto toga koristi se poseban sistem obeležavanja — komplementarnim brojem. Kod komplementarnog broja levi bit se koristi kao bit znaka. Ako je taj bit 1, broj za premeštanje je negativan. Ako je 0, broj je pozitivan. Način dobijanja komplementarnog binarnog broja možete naći u malo većim knjigama od ove, ali princip je u tome da se svaka 0 pretvara u 1, a svaka jednnica u nulu, da bi se krajnjoj desnoj cifri dodao 1. Potom se to pretvara u heksa oblik i unosi u program.

15. KUDA DALJE

Ako vam se čitanjem ovog poglavlja nije ogadilo programiranje u mašinskom kodu, onda ste stvarno zainteresovani i pravo je pitanje kako i otkuda saznati još više. Kao što ste primetili ovo i nije bio kurs nego prikaz nekih, najosnovnijih elemenata za mašinsko programiranje. Neke od tih radnji i nije neophodno da pamtite, čak i ako se upustite dublje u mašinac.

Mnogo jednostavniji pristup je da nabavite dobar asembler. Mnogo firmi nudi bolje i lošije asemblere, pa se raspitajte kod prijatelja ili prelistajte neki od časopisa o kućnim računarima pre nego što se opredelite za nabavku. Sa asemblerom možete čak da kucate i napomene uz mnemonike da biste se podsetili čemu koja upotrebljena instrukcija služi. Pošto završite unošenje svog programa, asembler će ga prikazati na ekranu u heksa obliku i u mnemonicima, sa adresama na kojima su instrukcije odložene i sa vašim napomenama. Da vam olakša, asembler će automatski menjati red cifara kod adresa i izračunati broj za premeštanje kod skoka. Neki asembleri dozvoljavaju i korišćenje simboličnih imena za podatke, kao kod promenljivih u BASIC-u. Dobar asembler ima još dve funkcije — on će pronaći gde ste napravili grešku, a editorski deo će vam poslužiti da je ispravite.

Do tad predstoji vam dalje učenje. Literature ima najviše na engleskom jeziku. Nešto od dobrih knjiga pomenuto je i u bibliografiji na kraju PZP, a nemojte propustiti da pročitate prikaze novih knjiga u časopisima. U međuvremenu prostudirajte priručnik vašeg računara i pokušajte da odgonetnete neki program u mašincu kakvih na desetine ima u svim časopisima iz ove oblasti.

VII/ REČNIK DIJALEKATA BASIC-A

Uporedni pregled naredbi: standardni Microsoft i Applesoft, Atari, BBC, CBM 64, Galaksija, TRS 80, ZX Spectrum

Verujemo da će vam ovo poglavlje biti najduže od koristi u PZP. Zamišljeno je tako da bude podsetnik standardnog Microsoft BASIC-a, a firma MICROSOFT napravila je do sada najčešće korišćeni dijalekt ovog programskog jezika. Njega, sa podvarijantama, koriste svi popularni mikroračunari u svetu.

Podsetnik se sastoji od tabela u kojima su u gornjem redu date ključne reči u standardnom obliku sa kratkim objašnjenjem čemu služe. Ispod njih nalaze se te iste reči u varijantama za one mikroračunare kojih u vreme pripreme PZP ima u najvećem broju u Evropi. Znači, ne samo da dobijate kratki pregled naredbi i funkcija standardnog BASIC-a nego i veliku mogućnost da prerađujete programe jednog računara za drugi.

Sve ključne reči su date abecednim redom da biste mogli da ih koristite lako, tokom rada na nekom programu.

U tabelama su korišćene skraćenice i simboli:

iz=izraz pr=promenljiva ik=iskaz in=indeks
ad=adresa (u memoriji) br=broj reda kt=konstanta
kr=korak [ ] u formatu=nije obavezan deo naredbe

Microsoft	**ABS** apsolutna vrednost izraza	**ASC** ASCII kod prvog znaka stringa	**ATN** arkustan-gens izraza	**AUTO** automatski numerator p. redova
Uređaj	ABS(iz)	ASC(string)	ATN(iz)	AUTO[br, kr]
APPLE II	ABS(iz)	ASC(string)	ATN(iz)	AUTO[br, kr]
ATARI	ABS(iz)	ASC(string)	ATN(iz)	
BBC Micro	ABS(iz)	ASC(string)	ATN(iz)	AUTO[br, kr]
COMMODORE 64	ABS(iz)	ASC(string)	ATN(iz)	
GALAKSIJA				
TRS 80 II	ABS(iz)	ASC(string)	ATN(iz)	AUTO[br, kr]
ZX SPECTRUM	ABS(iz)	CODE (string)	ATN(iz)	

Microsoft	**CALL** poziva asembler potprogram	**CHAIN** novi prog. sa starim promenljivim	**CHR$** daje jednoznačni kod stringa	**CLEAR** briše odabrane promenljive
Uređaj	CALL pr	CHAIN "ime"	CHR$(iz)	CLEAR (iz,iz)
APPLE II	CALL ad	CHAIN "ime"	CHR$(iz)	CLR
ATARI BBC Micro	CALL ad	CHAIN "ime"	CHR$(iz) CHR$(iz)	CLEAR
COMMODORE 64			CHR$(iz)	CLR
GALAKSIJA			CHR$(iz)	
TRS 80 II			CHR$(iz)	CLEAR (iz,iz)
ZX SPECTRUM	USR(ad)		CHR$(iz)	CLEAR

Microsoft	**CLOSE** zatvara disk-fajle	**CONT** nastavlja izvršenje programa	**COS** kosinus izraza	**DATA** lista podatke za READ
Uređaj		CONT	COS(iz)	DATA kt
APPLE II		CONT	COS(iz)	DATA kt
ATARI	CLOSE [#]ime	CONT	COS(iz)	DATA kt
BBC Micro	CLOSE #ime		COS(iz)	DATA kt
COMMODORE 64	CLOSE ime	CONT	COS(iz)	DATA kt
GALAKSIJA				#
TRS 80 II	CLOSE ime	CONT	COS(iz)	DATA kt
ZX SPECTRUM	CLOSE#	CONTINUE br	COS(iz)	DATA kt

Microsoft	**DEF** definiše aritmetičku string f-ju	**DELETE** briše programski red	**DIM** rezerviše prostor za niz	**EDIT** pokazuje programski red
Uređaj	DEF FNpr	DELETE br	DIM pr(in)	EDIT br
APPLE II	DEF FNpr	DELETE br	DIM pr(in)	
ATARI			DIM pr(in) isto COM	
BBC Micro	DEF FNpr	DEL br	DIM pr(in)	
COMMODORE 64	DEF FNpr		DIM pr(in)	
GALAKSIJA			ARR$(in)	EDIT br
TRS 80 II		DELETE br	DIM pr(in)	EDIT br
ZX SPECTRUM	DEF FNpr	DELETE	DIM pr(in)	EDIT

Microsoft	**END** kraj prog. i vraća na BASIC	**EXP** stepenuje izraz	**FOR** koristi se sa NEXT za ponavljanje	**FRE** daje ostatak memorije
Uređaj	END	EXP(iz)	FOR pr=iz TO iz	FRE(iz)
APPLE II	END	EXP(iz)	FOR pr=iz TO iz	FRE(iz)
ATARI	END	EXP(iz)	FOR pr=iz TO iz	FRE(iz)
BBC Micro	END	EXP(iz)	FOR pr=iz TO iz	
COMMODORE 64	END	EXP(iz)	FOR pr=iz TO iz	FRE(iz)
GALAKSIJA			FOR pr=iz TO iz	MEM
TRS 80 II	END	EXP(iz)	FOR pr=iz TO iz	FRE(iz)
ZX SPECTRUM		EXP(iz)	FOR pr=iz TO iz	

Microsoft	**GET** čita zapis sa diska ili trake	**GOSUB** skreće na potprogram	**GOTO** skreće na dati broj reda	**IF/THEN** uslovno izvršava iskaz
Uređaj	GET[#] broj fajle	GOSUB br	GOTO br	IFizTHENik [ELSEik]
APPLE II		GOSUB br	GOTO br	IFizTHENik
ATARI	GET#iz	GOSUB br	GOTO br/iz	IFizTHENik [ELSEik]
BBC Micro	INPUT#	GOSUB br	GOTO ad/br/ iz	IFizTHENik [ELSEik]
COMMODORE 64	GET#br. fl., pr	GOSUB br	GOTO br	
GALAKSIJA		CALL br/iz	GOTO br/iz	IF iz
TRS 80 II	INPUT #—1	GOSUB br	GOTO br	IFizTHENik [ELSEik]
ZX SPECTRUM	INPUT	GOSUB br/iz	GOTO br/iz	IFizTHENik (bez ELSE)

Microsoft	**INKEY$** daje kucani znak ili 0	**INPUT** čita podatke sa terminala	**INT** izračunava najveće celo	**LEFT$** daje dati deo stringa od početka
Uređaj	INKEY$	INPUT	INT(iz)	LEFT$ (string ,dužina)
APPLE II		INPUT	INT(iz)	LEFT$ (string ,dužina)
ATARI		INPUT	INT(iz)	string (1,br znakova)
BBC Micro	INKEY$	INPUT	INTiz	LEFT$ (string ,dužina)
COMMODORE 64		INPUT#	INT(iz)	LEFT$ (string ,dužina)
GALAKSIJA	KEY(kod dirke)	INPUT	INT(iz)	
TRS 80 II	INKEY$	INPUT	INTiz	LEFT$ (string ,dužina)
ZX SPECTRUM	INKEY$	INPUT	INTiz	string(TO)

Microsoft	**LEN** daje dužinu stringa	**LET** daje vrednost promen-ljivoj	**LIST** daje listu programa	**LLIST** daje listu programa na štampač
Uređaj	LEN(string)	[LET]pr=iz	LIST [br, br]	LLIST [br]
APPLE II	LEN(string)	[LET]pr=iz	LIST [br, br]	LIST"P
ATARI	LEN(string)	[LET]pr=iz	LIST [br, br]	LIST "P;"
BBC Micro	LEN(string)	[LET]pr=iz	LIST [br, br]	CTRL B LIST [br]
COMMODORE 64	LEN(string)	[LET]pr=iz	LIST [br-br]	
GALAKSIJA		pr=iz	LIST [br]	
TRS 80 II	LEN(string)	[LET]pr=iz	LIST [br-br]	LIST (u PRO modu)
ZX SPECTRUM	LEN(string)	LET pr=iz	LIST [br]	LLIST [br]

Microsoft	**LOAD** puni prog. u memoriju	**LOG** daje prirodni logaritam iz	**MID$** daje znakove udesno od datog mesta	**NAME** daje novo ime fajli
Uređaj	LOAD "ime"	LOG(iz)	MID$ (string, mesto,duž.)	NAME "ime" AS "ime"
APPLE II	LOAD	LOG(iz)	MID$ (string, mesto,duž.)	RENAME "ime" "ime"
ATARI	LOAD(ime) CLOAD (ime)	LOG(iz)	string(m,n)	
BBC Micro	LOAD ["ime"]	LN(iz)	MID$ (string, mesto,duž.)	
COMMODORE 64	LOAD ["ime"]	LOG(iz)	MID$ (string, mesto,duž.)	
GALAKSIJA	OLD			
TRS 80 II	CLOAD ["ime"]	LOG(iz)	MID$ (string, mesto,duž.)	
ZX SPECTRUM	LOAD "ime"	LN(iz)	string (m TO n)	

Microsoft	**NEW** briše program i podatke	**NEXT** kraj FOR/NEXT petlje	**ON ERROR** potprogram za traženje greške	**ON/GOSUB** ide na br. reda dat izrazom
Uređaj	NEW	NEXT pr	ON ERROR GOTO br	ON iz GOSUB br
APPLE II	NEW	NEXT pr	ONERR GOTO br	ON iz GOSUB br
ATARI	NEW	NEXT pr	TRAPiz	ON iz GOSUB br
BBC Micro	NEW	NEXT [pr]	ON ERROR ik	ON iz GOSUB br
COMMODORE 64	NEW	NEXT pr		
GALAKSIJA	NEW	NEXT pr		
TRS 80 II	NEW	NEXT [pr]	ON ERROR GOTO br	ON iz GOSUB br
ZX SPECTRUM	NEW	NEXT [pr]		

Microsoft	**ON/GOTO** ide na red određen izrazom	**OPEN** otvara novu disk-fajlu	**OUT** šalje bajt na izlazni kanal	**PEEK** čita bajt sa adrese
Uređaj	ON iz GOTO br	OPEN mod br.f."ime"	OUT kanal, bajt	PEEK(ad)
APPLE II	ON iz GOTO br	OPEN "ime" [,mod]		PEEK(ad)
ATARI	ON iz GOTO br	OPEN#iz,iz kod fajle	PUT#iz,iz	PEEK(ad)
BBC Micro	ON iz GOTO br	ROPEN ili WOPEN „ime"		bajt=?ad
COMMODORE 64	ON iz GOTO br	OPEN		PEEK(ad)
GALAKSIJA				PRINT BYTE
TRS 80 II	ON iz GOTO br		OUT kanal, bajt	PEEK(ad)
ZX SPECTRUM		OPEN#	OUT kanal, bajt	PEEK(ad)

Microsoft	**POKE** smešta dati bajt na adresu	**PRINT** upisuje podatak u disk-fajl	**RANDO-MIZE** na početak niza slučaj. brojeva	**READ** dodeljuje vrednosti iz DATA iskaza
Uređaj	POKE ad,bajt	PRINT[[#] broj fajle]	RANDO-MIZE [iz]	READ pr[,pr. . .]
APPLE II	POKE ad,bajt	PRINT[iz] [;][iz]		READ pr[,pr. . .]
ATARI	POKE ad,bajt	PRINT[#] broj fajle		READ pr[,pr. . .]
BBC Micro	?ad=bajt	PRINT [#"ime"]		READ pr[,pr. . .]
COMMODORE 64	POKE ad,bajt	PRINT# [broj fajl]		READ pr[,pr. . .]
GALAKSIJA	BYTE ad,bajt			TAKE pr[,pr. . .]
TRS 80 II	POKE ad,bajt	PRINT[iz] [,iz . . .]	RANDOM	READ pr[,pr. . .]
ZX SPECTRUM	POKE ad,bajt		RAND	READ pr[,pr. . .]

Microsoft	**REM** napomene koje program ignoriše	**RENUM** menja brojeve redova	**RESTORE** vraća poenter za DATA	**RESUME** vraća na izraz sa greškom
Uređaj	REM tekst	RENUM [br,kr]	RESTORE	RESUME
APPLE II	REM tekst		RESTORE	RESUME
ATARI	REM tekst		RESTORE	
BBC Micro	REM tekst	RENUM [br,kr]	RESTORE iz	RESUME
COMMODORE 64	REM tekst		RESTORE	
GALAKSIJA	! tekst			
TRS 80 II	REM tekst		RESTORE	RESUME
ZX SPECTRUM	REM tekst		RESTORE	

Microsoft	**RETURN** vraća iz pot-programa na iza GOSUB	**RIGHT$** daje dati deo stringa do kraja	**RND** generiše slučajni broj	**RUN** izvršava program
Uređaj	RETURN	RIGHT$ (string,duž)	RND[iz]	RUN[br]
APPLE II	RETURN	RIGHT$ (string,duž)	RND[iz]	RUN[br]
ATARI	RETURN	string(m)	RND[iz]	RUN[br]
BBC Micro	RETURN	RIGHT$ (string,duž)	RND[iz]	RUN
COMMODORE 64	RETURN	RIGHT$ (string,duž)	RND(iz)	RUN[iz]
GALAKSIJA	RETURN		RND[iz]	RUN[br]
TRS 80 II	RETURN	RIGHT$ (string,duž)	RND[iz]	RUN[br]
ZX SPECTRUM	RETURN	string(TO)	RND[iz]	RUN

Microsoft	**SAVE** snima prog. na traku ili disk	**SGN** daje 1(iz>0) 0(iz=0) ili —1(iz<0)	**SIN** daje sinus izraza (u radijan.)	**SQR** daje kvadratni koren izraza
Uređaj	SAVE "ime"	SGN(iz)	SIN(iz)	SQR(iz)
APPLE II	SAVE	SGN(iz)	SIN(iz)	SQR(iz)
ATARI	SAVE(ime) CSAVE (ime)	SGN(iz)	SIN(iz)	SQR(iz)
BBC Micro	SAVE ime	SGN(iz)	SIN(iz)	SQR(iz)
COMMODORE 64	SAVE ["ime"]	SGN(iz)	SIN(iz)	SQR(iz)
GALAKSIJA	SAVE			
TRS 80 II	CSAVE ["ime"]	SGN(iz)	SIN(iz)	SQR(iz)
ZX SPECTRUM	SAVE "ime"	SGN(iz)	SIN(iz)	SQR(iz)

Microsoft	**STOP** zaustavlja program	**STR$** pretvara brojčani iz. u string	**TAN** daje tangens izraza (u radijan.)	**USR** poziva asem. potprogram za neki broj
Uređaj	STOP	STR$(iz)	TAN(iz)	USR (parametar)
APPLE II	STOP	STR$(iz)	TAN(iz)	USR (parametar)
ATARI	STOP	STR$(iz)	TAN(iz)	USR (parametar)
BBC Micro	STOP	STR$(iz)	TAN(iz)	USR (parametar)
COMMODORE 64	STOP	STR$(iz)	TAN(iz)	USR (parametar)
GALAKSIJA	STOP			USR (parametar)
TRS 80 II	STOP	STR$(iz)	TAN(iz)	USR (parametar)
ZX SPECTRUM	STOP	STR$(iz)	TAN(iz)	USR (parametar)

Microsoft	**VAL** daje string u obliku broja	**WAIT** zaustavlja program za dato vreme	**WIDTH** postavlja parametre za štampač	**WRITE** zapisuje podatke na traku/disk
Uređaj	VAL(string)	WAIT port, mark	WIDTH (parametar)	WRITE [#ime fajle]
APPLE II	VAL(string)	WAIT ad,iz	POKE 82,vr. POKE 83,vr.	WRITE "ime"
ATARI	VAL(string)			PRINT# (ime)
BBC Micro	VAL(string)		WIDTH (parametar)	PRINT# ("ime")
COMMODORE 64	VAL(string)	WAIT ad,iz		
GALAKSIJA	VAL(string)			
TRS 80 II	VAL(string)			PRINT#
ZX SPECTRUM	VAL(string)	PAUSE broj(50/sec)		PRINT (Micro-drive)

VIII/ ZA VAŠU BIBLIOTEKU

Na kraju PZP pripremili smo vam po nekoliko programa za najmasovnije kućne računare sredinom osamdesetih godina u Jugoslaviji, a to su ZX SPECTRUM i COMMODORE 64. Ako i ne pripadate ni jednoj ni drugoj porodici biće vam korisne liste programa, ako se potrudite oko nalaženja odgovarajućih naredbi u uporednim tabelama dijalekata BASIC-u koje imate u PZP. Prvenstvo ima stariji:

Programi za ZX SPECTRUM

POTAPANJE

Klasična igra koju ste igrali uz pomoć olovke i papira, a sada je možete igrati i uz pomoć računara. Vaš zadatak je da potopite flotu koju je postavio računar pre nego što on potopi vašu flotu. Najpre morate postaviti vaših šest brodova u koordinatno polje na levoj strani ekrana. Ovo ćete lako učiniti sledeći uputstva koja će vam davati računar. Svaki od brodova mora biti postavljen po horizontali, vertikali ili dijagonali. Gledaćete kako se vaša flota pojavljuje u levom kvadratu na ekvranu.

Zatim morate pronaći i potopiti flotu koju je postavio računar, tako što ćete ispaljivati po tri granate u kvadrant koji odredite na desnoj strani ekrana. Postupak je isti kao i postavljanje flote. Rezultat će se videti grafički u desnom kvadratu i tekstualno ispod vašeg dela ekrana. Posle vas računar ispaljuje tri granate na vašu flotu, i tako naizmenično dok jedna flota ne bude potopljena u celosti.

Kada unesete program u računar pažljivo proverite da li ima grešaka, i ako nema snimite ga sa: SAVE "potapanje" LINE 1. Program će po učitavanju sam startovati.

NAPOMENA: znak koji se nalazi između dva ovakva " " znaka, na primer " A ", predstavlja grafički simbol i unosi se u modu za grafiku (G modu).

```
 10 REM *** POTAPANJE ***
 15 REM Startuje program i ucitava grafiku
 20 GO SUB 1000
 21 BORDER 1: PAPER 5: CLS : INK 7
 22 POKE 23658,8: POKE 23609,50
 25 LET h=0: LET i=0: LET j=0
 30 DIM a(2,10,10): DIM n(2,6): DIM n$(6,11)
 60 RANDOMIZE
 65 RESTORE 70: FOR i=1 TO 6: READ x$: LET n$(i)=x$: NEXT i
 70 DATA "Bojni brod","Krstarica","Razarac1","Razarac2","Podmor
nica1","Podmornica2"
 75 FOR s=1 TO 2: RESTORE 77: FOR i=1 TO 6: READ x: LET n(s,i)=
x: NEXT i: NEXT s
 77 DATA 5,4,3,3,2,2
 80 GO TO 300
160 REM Unosi flotu igraca
165 INPUT a$: IF a$="" THEN  GO TO 165
170 LET x=(CODE a$(1))-64: IF x>32 AND x<43 THEN  LET x=x-32
175 IF x<1  OR x>10 THEN  GO TO 165
180 LET a$=a$(2 TO ): IF CODE a$<48 OR CODE a$>57 THEN  GO TO 1
65
185 LET y=INT (VAL a$)+1: IF y<1 OR y>10 THEN  GO TO 165
187 IF a(2,x,y)<0 THEN  GO TO 165
190 RETURN
200 REM Izracunava gde da uputi kompjuterove granate
202 IF h<>0 AND j<>0 THEN  GO TO 222
205 IF h<>0 THEN  GO TO 230
210 LET x=INT (RND*10)+1: LET y=INT (RND*10)+1: IF a(1,x,y)<0 T
HEN  GO TO 210
220 RETURN
222 LET p=h: LET q=i
230 FOR d=-1 TO 1: FOR u=-1 TO 1: LET x=u+p: IF x<1 OR x>10 THE
N  GO TO 263
235 LET y=d+q: IF y<1 OR y>10 THEN  GO TO 265
261 IF a(1,x,y)<0 THEN  GO TO 263
262 RETURN
263 NEXT u
265 NEXT d
270 LET p=h1: LET q=i1: GO TO 230
275 GO TO 210
300 REM GLAVNI PROGRAM
301 PAPER 7: INK 0
305 PRINT  PAPER 5;"  "; PAPER 7;"Tvoja flota "; PAPER 5;"     "
; PAPER 7;" Moja flota ": PRINT
310 LET b$=" ABCDEFGHIJ ": PRINT  PAPER 5;"  "; PAPER 7;b$; PAP
ER 5;"     "; PAPER 7;b$
320 FOR s=3 TO 12: RESTORE 330: FOR i=1 TO 4: READ x: PRINT AT
s,x; INVERSE 1;CHR$ (60-s): NEXT i: NEXT s
330 DATA 2,13,18,29
340 PRINT  PAPER 5;"  "; PAPER 7;b$; PAPER 5;"     "; PAPER 7;b$
: PRINT AT 15,2; INK 2; FLASH 1;"Postavljam svoju flotu"
```

```
 400 FOR s=1 TO 6
 410 LET d=INT (RND*3)-1: LET u=INT (RND*3)+1: IF d=0 AND u=0 TH
EN  GO TO 410
 420 LET p=INT (RND*10)+1: LET q=INT (RND*10)+1
 430 LET m=n(1,s)-1: LET t1=p+d*m: LET t2=q+u*m
 440 IF t1>10 OR t1<1 OR t2>10 OR t2<1 THEN  GO TO 410
 450 FOR l=0 TO m: IF a(2,p+d*l,q+u*l)<>0 THEN  GO TO 410
 460 NEXT l
 490 FOR l=0 TO m: LET a(2,p+d*l,q+u*l)=s: NEXT l
 500 NEXT s
 550 PRINT AT 15,2;"Sada ti postavi svoju flotu"
 570 FOR s=1 TO 6: PRINT AT 17,2;n$(s)
 590 FOR l=1 TO n(2,s): PRINT AT 19,2;"Kvadrat";l: GO SUB 160
 595 IF a(1,x,y)<>0 THEN  GO SUB 160: GO TO 595
 600 LET a(1,x,y)=s: PRINT AT 13-y,x+2;n$(s,1): NEXT l: NEXT s
 630 PRINT AT 15,2; PAPER 5;"                              "
 640 PRINT AT 17,2; PAPER 5;"                "
 650 PRINT AT 19,2; PAPER 5;"                    "
 660 LET k=2
 665 REM Rutina za Paljbu
 670 IF k=1 THEN  PRINT AT 15,2;"Ja sam na potezu                  "
 680 IF k=2 THEN  PRINT AT 15,2;"Ti si na potezu                   "
 690 FOR g=1 TO 3
 700 FOR s=1 TO 20: NEXT s
 710 PRINT AT 17,2; PAPER 5;"                              "
 720 PRINT AT 16,2;"PALJBA";g
 730 IF k=1 THEN  GO SUB 200
 740 IF k=2 THEN  GO SUB 160
 750 LET z=a(k,x,y): LET a(k,x,y)=-1: IF z=0 THEN  GO TO 900
 755 REM Pogodak
 760 PRINT AT 13-y,x+(k*16)-14; FLASH 1; PAPER 2; INK 6;"@B@"
 770 PRINT AT 17,2;"BUUM"; PAPER 5;"                    ": BEEP 1,1
 780 IF k=1 THEN  IF h=0 THEN  LET h1=x: LET i1=y
 785 IF k=1 THEN  LET h=x: LET i=y: LET j=1
 790 LET n(k,z)=n(k,z)-1
 800 IF n(k,z)<>0 THEN  GO TO 875
 805 PRINT AT 17,2;n$(z);" tone"
 810 IF k=1 THEN  LET h=0
 815 FOR s=1 TO 50: NEXT s
 820 FOR s=1 TO 6: IF n(k,s)>0 THEN  GO TO 875
 825 NEXT s
 830 IF k=2 THEN  PRINT AT 19,2; INK 6; PAPER 2; FLASH 1;"Ti si
";
 835 IF k=1 THEN  PRINT AT 19,2; INK 6; PAPER 2; FLASH 1;"Ja sam
";
 840 PRINT  INK 6; PAPER 2; FLASH 1;"Pobedio."
 850 FOR k=1 TO 2: FOR x=1 TO 10: FOR y=1 TO 10
 855 IF a(k,x,y)=0 THEN  PRINT AT 13-y,x+(k*16)-14;"@A@": BEEP
1,1
 860 IF a(k,x,y)>0 THEN  PRINT AT 13-y,x+(k*16)-14; FLASH 1; PAP
ER 2; INK 6;"@B@": BEEP 1,1
 865 NEXT y: NEXT x: NEXT k
 870 GO TO 9990
 875 IF k=1 THEN  PAUSE 50
 880 NEXT g
 890 LET k=1+ABS (k-2): GO TO 670
 900 REM Promasaj
 905 PRINT AT 13-y,x+(k*16)-14;"@A@": BEEP .1,1
 910 PRINT AT 17,2;"PLJAS"; PAPER 5;"                    "
 915 IF k=1 THEN  LET j=0
```

```
 [illegible] GO TO 875
 999 GO TO 9990
1000 REM Podprogram koji ucitava graficke simbole
1010 RESTORE 1020: FOR n=USR "a" TO USR "b"+7: READ nn: POKE n,n
n: NEXT n: RETURN
1020 DATA 85,170,85,170,85,170,85,170
1030 DATA 153,90,90,255,255,90,90,153
9990 PRINT AT 21,0;"Bilo koja dirka za sledecu igru"
9991 GO TO 20
```

REGISTAR

Ovaj program će vam prikazati abecedni redosled podataka koje ste uneli u računar. Može vam poslužiti kao registar knjiga, kaseta, ploča itd. Najpre unosite naziv (vrstu) registra kao i ime autora (korisnika). Zatim formirate veličinu stranice unetog teksta, određivanjem broja unetih znakova po stranici. Kada završite sa unosom podataka, računar će sortirati stranice po abecednom redu i ponudiće vam opcije za štampanje registra na štampaču, snimanje na kasetu, početak novog registra ili završetak programa.

Posle unošenja programa u računar i provere listinga (da nema grešaka), program ćete snimiti sa: SAVE "registar" LINE 1.

```
  10 REM *** REGISTAR ***
  30 REM Postavlja Podatke u centar ekrana Pomocu funkcije
  40 CLEAR
  50 DEF FN t(x$)=INT ((32-LEN x$)/2)
  60 REM unosi naziv registra u ime korisnika (autora)
  70 PRINT TAB 8; INK 7; PAPER 2;"R E G I S T A R";AT 2,0; PAPER
 1;"Unesite naziv registra": INPUT a$
 120 PRINT AT 2,0; INK 7; PAPER 1;"Unesite ime korisnika ": INPU
T b$
 140 REM forsira registar i kolicinu raspolozive memorije
 150 PRINT AT 2,0; INK 7; PAPER 1;"Unesite broj stranica registr
a  koji zelite da napravite"
 160 PRINT AT 15,0; INK 1; PAPER 7;"Vodite racuna o broju strani
ca, naknadno se ne moze menjati"
 170 INPUT n
 190 PRINT AT 2,0; INK 7; PAPER 1;"Unesite maksimalan broj znako
va po stranici"; PAPER 7;"
                    ": INPUT m
 210 LET Pramt=PEEK 23732+256*PEEK 23733: LET ext=Pramt-32767
 220 LET d=5000+ext-3*n
 230 IF d<m*n THEN  PRINT AT 2,0; INK 7; PAPER 2;"Unesite ponovo
 broj stranica za ";m;" znakova, najveci broj": PRINT  INK 7; PA
PER 2;" stranica je ";INT (d/m): GO TO 170
 235 REM Priprema registar za unos Podataka
 240 DIM l$(n+1,m+3)
 260 CLS : PRINT  INK 1;"Spremite se da unesete Podatke maksimal
ne duzine od "; INK 2;"**";m;"**"; INK 1;" znakova Po Jednoj str
anici"
 265 PRINT '"Za Pocetak unosa-bilo koja dirka"
 269 REM unosi Podatke utvrdjene duzine i broj stranice
 270 PAUSE 0: PRINT : CLS
 300 FOR i=1 TO n
 310 PRINT  INK 7; PAPER 1;"UNOS ";i;AT 15,7;"Sadrzaj?"
 320 IF i=n-10 THEN  PRINT AT 15,0; INK 7; PAPER 2;"PAZNJA - nem
a vise memorije"
 330 INPUT x$: IF LEN x$<=m THEN  GO TO 390
 340 PRINT AT 8,5; INK 7; PAPER 2;"Skratite unos"
 350 PRINT AT 4,5; INK 1;x$;"                                    ": F
OR f=1 TO m: PRINT AT 5,4+f;"-": NEXT f: GO TO 330
 390 LET l$(i)(1 TO m)=x$: LET x$=" ": IF CODE l$(i)=226 THEN LE
T n=i-1: GO TO 500
```

```
 430 CLS : PRINT  INK 1;"UNOS ";i;TAB 12;l$(i)( TO m);AT 15,0; I
NK 7; PAPER 1;"Broj strane?"
 440 INPUT p$: LET c=CODE p$: LET s=1
 450 IF c=32 THEN  LET s=s+1: LET c=CODE p$(s): GO TO 450
 460 IF c<48 OR c>57 THEN  PRINT AT 18,5; PAPER 2;"GRESKA": GO T
O 440
 470 LET l$(i)(m+1 TO )=p$
 480 CLS : NEXT i
 500 REM sortira stranice po abecedi
 510 CLS : PRINT AT 10,5; PAPER 6; INK 2; FLASH 1;"PODACI SE SOR
TIRAJU"
 520 FOR k=1 TO n-1: FOR j=1 TO n-k: IF l$(j)<=l$(j+1) THEN  GO
TO 580
 550 LET t$=l$(j): LET l$(j)=l$(j+1): LET l$(j+1)=t$
 580 NEXT j: NEXT k
 610 REM stampa sortiran registar na ekran
 630 CLS : PRINT TAB FN t(a$+"  "); INK 1;"""";a$;"""";TAB FN t(
"napisao");"napisao"'TAB FN t(b$);b$: PRINT
 640 PAUSE 100: PRINT
 650 FOR i=1 TO n: PRINT  INK 1;l$(i)( TO m);TAB m+7;l$(i)(m+1 T
O ): NEXT i
 700 PRINT ''TAB FN t("KRAJ REGISTRA"); INK 7; PAPER 1;"KRAJ REG
ISTRA"
 705 REM nudi opcije za stampanje, snimanje, brisanje ili izlaz
 710 INK 7: PAPER 1: PRINT ''"Zelite li...                      "
 712 PRINT " S - da snimite registar          "
 714 PRINT " P - da ga stampate na printeru "
 716 PRINT " N - da pocnete novi registar    "
 718 PRINT " Z - da zavrsite sa radom?        "
 720 POKE 23658,8: INK 0: PAPER 7
 730 INPUT q$: IF q$<>"S" AND q$<>"P" AND q$<>"N" AND q$<>"Z" TH
EN  GO TO 730
 735 POKE 23658,0
 739 REM deo programa odabran opcijom
 740 IF q$="S" THEN  GO TO 780
 750 IF q$="P" THEN  GO TO 810
 760 IF q$="N" THEN  GO TO 30
 770 GO TO 9999
 775 REM opcija za snimanje programa
 780 SAVE "registar" LINE 600: CLS : GO TO 710
 800 REM opcija za stampanje registra na stampacu (Printeru)
 810 CLS
 820 LPRINT TAB FN t(a$+"  ");"""";a$;"""";TAB FN t("napisao");"
napisao"'TAB FN t(b$);b$: LPRINT
 850 LPRINT TAB 8;"R E G I S T A R": LPRINT : LPRINT
 880 FOR i=1 TO n: LPRINT l$(i)( TO m);TAB m+7;l$(i)(m+1 TO ): N
EXT i
 910 LPRINT : LPRINT : LPRINT TAB FN t("KRAJ REGISTRA");"KRAJ RE
GISTRA": GO TO 710
```

KO ĆE PRE

Ovaj program trebalo bi da pobudi vaš takmičarski duh. Cilj je da se što brže na tastaturi otkuca cela abeceda, od A do Ž. Što je više takmičara igra je zabavnija jer najbolji rezultat računar »pamti« i posle svakog kruga ga prikazuje na ekranu. Program unesite sa listinga, proverite i snimite sa SAVE "ko će pre" LINE 1.

```
  1 REM *** KO CE PRE ***
  5 POKE 23658,8
 10 LET NV=4E4
 20 LET V=PI-PI
 23 CLS
 25 PRINT "              START !
 30 FOR N=65 TO 90
 40 IF INKEY$<>CHR$ N THEN  GO TO 130
 50 PRINT CHR$ N;
 60 NEXT N
 65 PRINT AT 0,12;"        "
 70 PRINT AT 6,0;"VREME=";V;" VREMENSKIH JEDINICA"
 80 IF V<NV THEN  LET NV=V
 90 PRINT ;;"NAJBOLJE VREME=";NV
100 PRINT ;;;;"ZA PONOVNI START-BILO KOJA TIRKA"
110 IF INKEY$="" THEN  GO TO 110
120 GO TO 20
130 LET V=V+1
140 GO TO 40
```

SLAGALICA

Verovatno vam je bar jedanput bila u rukama pločica u obliku kvadrata u kojoj se nalazilo još petnaest malih kvadrata na kojima su bili, ili brojevi, ili delovi neke sličice i jedno prazno mesto. U oba slučaja zadatak je bio isti: trebalo je ili složiti sličicu ili brojeve po određenom redu, pomerajuć pločice tako što bi se u prazno mesto postavila željena pločica, a zatim na njeno mesto sledeća i tako dalje, sve dok se ne bi dospelo do željenog rezultata (ili ne). Ovoga puta je pred vama "ispreturao" računar, a na vama je da ih složite po abecedi po pravilu koje vam je već poznato: pritisnućete taster sa slovom koje treba da popuni prazno mesto i tako sve do uspeha.

Posle unošenja i provere programa otkucajte: SAVE "slagalica" LINE 1 i snimite program na traku.

```
  1 REM  *** SLAGALICA ***
  5 POKE 23658,8
 10 DIM a$(4,4)
 20 LET a$(1)="ABCD"
 30 LET A$(2)="EFGH"
 40 LET A$(3)="IJKL"
 45 LET A$(4)="MNO "
 46 GO SUB 600
 61 PRINT AT 0,0;"*********": PRINT ;"*       *"
 62 FOR N=1 TO 4
 63 PRINT ;"* ";A$(N);" *"
 64 NEXT N
 65 PRINT ;"*       *": PRINT ;"*********"
 70 PRINT "KOJE SLOVO?"
 80 LET B$=INKEY$
 90 FOR N=1 TO 4
100 FOR M=1 TO 4
110 IF A$(N,M)=B$ THEN  GO TO 150
120 NEXT M
130 NEXT N
140 GO TO 80
150 IF (N=X OR N=X+1 OR N=X-1) AND (M=Y OR M=Y+1 OR M=Y-1) THEN
 GO TO 170
160 GO TO 80
170 LET A$(X,Y)=A$(N,M)
180 LET A$(N,M)=" "
190 LET X=N
195 LET Y=M
200 GO TO 61
610 FOR N=1 TO 20
620 LET Q=INT (RND*4)+1
630 LET T=INT (RND*4)+1
631 LET P=INT (RND*4)+1
632 LET L=INT (RND*4)+1
640 LET T$=A$(Q,T)
645 LET A$(Q,T)=A$(P,L)
650 LET A$(P,L)=T$
660 NEXT N
661 FOR N=1 TO 4
662 FOR M=1 TO 4
663 IF A$(N,M)=" " THEN  GO TO 671
664 NEXT M
665 NEXT N
671 LET X=N
672 LET Y=M
680 RETURN
```

CRTANKA

Evo jednog programa čije vam ime sve govori: pomoću njega možete da se zabavite crtajući po ekranu. Ovde ćemo vam umesto uvoda dati spisak naredbi koje se koriste u programu, a ostatak prepuštamo vašoj mašti. Naredbe se sastoje iz dva slova i pozitivnog ili negativnog broja (u zagradi).

rl (n) crta sa desna u levo
rl (-n) crta sa leva u desno
ud (n) crta vertikalno gore
ud(-n) crta vertikalno dole
du (n) crta dijagonalno gore i desno
du(-n) crta dijalgonalno gore i levo
dd (n) crta dijagonalno dole i desno
dd(-n) crta dijagonalno dole i levo
ca (n) pomera kursor desno
ca (-n) pomera kursor levo
cu (n) pomera kursor gore
cu(-n) pomera kursor dole
st zaustavlja program ili ponavljanje
hm vraća kursor na početnu poziciju
cs briše ekran ali ga zadržava u memoriji
rs briše ekran iz memorije
rp omogućava ponavljanje
ru poništava poslednju naredbu

Program unesite u računar, pažljivo proverite da nema grešaka i snimite ga na kasetu sa: SAVE "crtanka" LINE 1.

```
  1 REM *** CRTANKA ***
  5 LET ff=0: LET m=0
  6 BORDER 5
 10 LET x=128: LET y=88
 12 LET g$=""
 15 LET w=0: LET v=0
 20 DIM a$(6,2): DIM d$(6,2)
 40 FOR j=1 TO 6
 50 READ a$(j)
 60 LET d$(j)=a$(j)
 70 LET a$(j)=" "+STR$ j
 80 NEXT j
 90 DATA "ud","rl","dd","du","ca","cu"
120 PRINT AT 8,12;"CRTANKA"
125 PRINT AT 21,2;"ZA START - BILO KOJA DIRKA"
130 IF INKEY$="" THEN  GO TO 130
135 CLS
140 PLOT x,y
142 INPUT "NAREDBA? ",i$
145 PRINT AT 21,0;"
146 PRINT AT 21,0;i$
147 LET fg=0
150 IF i$="rs" THEN  CLS : LET x=128: LET y=88: LET g$="": GO T
O 140
```

```
 152 IF i$="cs" THEN  CLS : GO TO 140
 153 IF i$="st" THEN  STOP
 154 IF i$="ru" AND LEN g$<10 THEN  PRINT AT 21,0;"NE MOZETE SA
A OBRISATI": GO TO 140
 155 IF i$="ru" THEN  LET g$=g$( TO (LEN g$-8))+"  ": LET fg=1:
LET fh=1: LET fj=1: CLS : GO TO 200
 157 LET fj=0
 160 IF i$="rp" THEN  LET fg=1: LET fh=0: GO TO 200
 165 IF i$="hm" THEN  LET x=128: LET y=88: GO TO 140
 170 GO SUB 999
 180 GO TO 140
 200 LET l=LEN g$
 210 IF l<5 THEN  PRINT AT 21,0;"TO NECE RADITI": GO TO 140
 215 IF fh=0 THEN  GO TO 440
 220 DIM x$(l,4)
 230 FOR j=1 TO l
 240 IF g$(j)=" " THEN  GO TO 280
 245 IF x$(j-1)=" " AND x$(j+1)=" " THEN  GO TO 280
 250 IF g$(j-1)=" " THEN  LET x$(j)=g$(j TO (j+3))
 280 NEXT j
 290 FOR j=1 TO l
 300 IF x$(j,1)=" " THEN  GO TO 370
 310 IF x$(j,1)="1" THEN  GO SUB 2000
 320 IF x$(j,1)="2" THEN  GO SUB 3000
 330 IF x$(j,1)="3" THEN  GO SUB 4000
 340 IF x$(j,1)="4" THEN  GO SUB 5000
 350 IF x$(j,1)="5" THEN  GO SUB 6000
 360 IF x$(j,1)="6" THEN  GO SUB 7000
 370 NEXT j
 380 IF fj=1 THEN  GO TO 140
 440 PRINT AT 21,0;"KORISTITE ca  ILI cu ZA KURZOR "
 444 IF ff=1 THEN  LET g$=" "+g$(m TO )
 445 LET ff=1: LET fh=1
 450 INPUT "NAREDBA? ",i$
 455 IF i$="st" THEN  GO TO 950
 457 LET m=LEN i$+1
 460 PRINT AT 21,0;"                                          "
 461 LET l=LEN i$
 462 IF l<3 THEN  PRINT AT 21,0;"MORATE UNETI NAREDBU I BROJ
": GO TO 450
 463 IF i$(1)=CHR$ 45 OR (l=5 AND i$(l-1)=CHR$ 45) THEN  PRINT
T 21,0;"TO SE NE MOZE": GO TO 450
 465 IF i$(1 TO 2)<>"ca" AND i$(1 TO 2)<>"cu" THEN  PRINT AT 21
0;"SAMO NAREDBE ZA KURZOR, MOLIM  ": GO TO 450
 470 IF l>5 THEN  PRINT AT 21,0;"UNESITE KRACU NAREDBU": GO TO
50
475 LET fl=0
480 FOR j=3 TO l
490 IF (i$(j)>CHR$ 47 AND i$(j)<CHR$ 58) OR i$(j)=CHR$ 45 THEN
LET fl=fl+1
500 NEXT j
510 IF fl<l-2 THEN  PRINT AT 21,0;"MORATE UNETI NAREDBU I BROJ
  ": GO TO 450
520 IF i$(1 TO 2)="ca" THEN  LET h$=" 5"+i$(3 TO )
530 IF i$(1 TO 2)="cu" THEN  LET h$=" 6"+i$(3 TO )
540 LET g$=h$+"  "+g$
550 GO TO 200
950 PRINT AT 21,0;"""C"" ZA NASTAVAK, ""F"" ZA KRAJ "
952 IF INKEY$<>"c" AND INKEY$<>"f" THEN  GO TO 952
```

```
60 IF INKEY$="f" THEN  STOP
62 IF ff=1 THEN  LET 9$="  "+9$(6 TO )
65 LET ff=0
70 GO TO 140
99 LET l=LEN i$: LET fl=0
00 IF l<3 THEN  PRINT AT 21,0;"MORATE UNETI NAREDBU I BROJ
 RETURN
01 IF l>5 THEN  PRINT AT 21,0;"UNESITE KRACU NAREDBU": RETURN
02 FOR j=3 TO l
03 IF (i$(j)>CHR$ 47 AND i$(j)<CHR$ 58) OR i$(j)=CHR$ 45 THEN
ET fl=fl+1: NEXT j
04 IF fl<LEN i$-2 THEN  PRINT AT 21,0;"MORATE UNETI NAREDBU I
OJ    ": RETURN
05 LET e$=i$(1 TO 2): LET fl=0
07 IF i$(l)=CHR$ 45 OR (l=5 AND i$(l-1)=CHR$ 45) THEN  PRINT A
21,0;"TO NECE RADITI": RETURN
10 FOR j=1 TO 6
20 IF e$=d$(j) THEN  LET e$=a$(j): LET fl=1
40 NEXT j
50 IF fl=0 THEN  PRINT AT 21,0;"KOMANDA NE POSTOJI": RETURN
55 GO SUB 9500
60 IF l=3 THEN  LET 9$=9$+e$+i$(3 TO )+"   "
65 IF l=4 THEN  LET 9$=9$+e$+i$(3 TO )+"   "
70 IF l>4 THEN  LET 9$=9$+e$+i$(3 TO )+"   "
80 RETURN
00 GO SUB 8000
10 LET w=e: LET v=0
12 GO SUB 9000
15 PLOT x,y
17 BEEP .4,22
20 DRAW v,w
25 LET y=y+w
30 RETURN
00 GO SUB 8000
10 LET v=e: LET w=0
12 GO SUB 9000
15 PLOT x,y
17 BEEP .4,20
20 DRAW v,w
25 LET x=x+v
30 RETURN
00 GO SUB 8000
20 IF SGN e=1 THEN  LET v=e: LET w=-e
30 IF SGN e=-1 THEN  LET w=e: LET v=e
32 GO SUB 9000
35 PLOT x,y
37 BEEP .4,19
40 DRAW v,w
45 LET x=x+v: LET y=y+w
50 RETURN
00 GO SUB 8000
20 IF SGN e=1 THEN  LET v=e: LET w=e
30 IF SGN e=-1 THEN  LET w=-e: LET v=e
32 GO SUB 9000
35 PLOT x,y
37 BEEP .4,21
40 DRAW v,w

45 LET x=x+v: LET y=y+w
50 RETURN
00 GO SUB 8000
10 LET x=x+e
```

```
6012 IF x<0 THEN  LET x=0
6013 IF x>255 THEN  LET x=255
6015 PLOT x,y
6020 RETURN
7000 GO SUB 8000
7010 LET y=y+e
7013 IF y<0 THEN  LET y=0
7014 IF y>175 THEN  LET y=175
7015 PLOT x,y
7020 RETURN
8000 IF fg=0 THEN  LET e=VAL i$(3 TO ): RETURN
8005 IF x$(j,3)<>" " THEN  LET e=VAL x$(j)(2 TO 3)
8010 IF x$(j,4)<>" " THEN  LET e=VAL x$(j)(2 TO 4)
8020 IF x$(j)(3)=" " THEN  LET e=VAL x$(j)(2)
8050 RETURN
9000 IF x+v<0 THEN  LET v=-x
9040 IF x+v>255 THEN  LET v=255-x
9050 IF y+w<0 THEN  LET w=-y
9060 IF y+w>175 THEN  LET w=175-y
9080 RETURN
9500 IF e$(2)="1" THEN  GO SUB 2000
9520 IF e$(2)="2" THEN  GO SUB 3000
9530 IF e$(2)="3" THEN  GO SUB 4000
9540 IF e$(2)="4" THEN  GO SUB 5000
9550 IF e$(2)="5" THEN  GO SUB 6000
9560 IF e$(2)="6" THEN  GO SUB 7000
9570 RETURN
```

KNJIGOVODSTVO

Ovaj relativno jednostavan, ali efikasan program će vam omogućiti da vodite sopstveno knjigovodstvo. Kada unesete program i startujete ga biće vam ponuđeno pet opcija — unošenje stavke, prikaz stavke, saldiranje, snimanje stavki na traku i brisanje pojedinih stavki. Unete stavke računar sortira po datumima unosa i po želji daje saldo za određeni datum. Program se može korisno upotrebiti za kontrolu čekovne knjižice, kućnih troškova i slično.

Program snimite sa: SAVE "finansije" LINE 1.

```
  5 REM *** KNJIGOVODSTVO ***
 10 LET n=0
 12 DIM x(150): DIM y(150): DIM z(150)
 15 DIM g$(1,8)
 20 DIM a$(150,8)
 30 DIM a(150)
 35 BORDER 0: PAPER 0: INK 7: CLS
 40 PRINT "     LICNO KNJIGOVODSTVO       "
 55 PRINT "MOZETE UNETI JOS ";150-n;" STAVKI"
 60 PRINT : PRINT " OPCIJE:": PRINT "      "
 70 PRINT
 80 PRINT
 90 PRINT "  1:Unos stavke": PRINT : PRINT "  2:Prikaz stavki":
PRINT : PRINT "  3:Svodjenje salda": PRINT
 95 PRINT "  4:Snimanje stavki"
 97 PRINT : PRINT "  5:Brisanje stavke"
100 PRINT AT 21,0;"Unesite opciju"
110 INPUT i
120 IF i>5 OR i<1 THEN  GO TO 110
125 CLS
130 GO TO i*1000
1000 REM
1001 REM UNOS PODATAKA
1002 REM
1010 PRINT "FORMIRANJE STAVKE"
1015 PRINT
1020 LET n=n+1
1030 IF n>=101 THEN  PRINT "MEMORIJA POPUNJENA": GO TO 35
1035 LET k=n
1040 PRINT "UNESITE IME STAVKE                (NE VISE OD 8 ZNAKOV
A)"
1050 INPUT a$(k)
1060 PRINT : PRINT a$(k)
1070 PRINT : PRINT "Unesite iznos"
1080 INPUT a(k)
1090 PRINT : PRINT "-";a(k)
1092 PRINT : PRINT "Unesite dan/mesec/godinu": INPUT x(k): PRINT
 x(k);"/"; INPUT y(k): PRINT y(k);"/"; INPUT z(k): PRINT z(k)
1100 PRINT : PRINT "Pritisnite ~M~ za OPCIJE": PRINT "Bilo koji
taster za Promenu ove stavke"
1110 INPUT k$
1120 IF k$="m" OR k$="M" THEN  GO TO 9000
1130 CLS : GO TO 1040
2000 REM
2001 REM PRIKAZ STAVKI
2002 REM
2010 CLS
2020 PRINT "PRIKAZ STAVKI"
2024 PRINT : PRINT "Pritisnite ~0~ za start/stop"
2030 PAUSE 0
```

```
2035 CLS
2060 FOR d=1 TO n
2070 PRINT a$(d);" ";x(d);"/";y(d);"/";z(d);" ";a(d)
2072 PRINT
2075 IF INKEY$="0" THEN  PAUSE 0
2080 NEXT d
2090 PRINT : PRINT "Bilo koja dirka za OPCIJE"
2100 PAUSE 0
2110 GO TO 35
3000 REM
3001 REM SVODJENJE SALDA
3002 REM
3005 LET to=0
3010 PRINT "SVODJENJE SALDA"
3020 PRINT : PRINT "Posledniji unos je bio ";
3030 PRINT x(n);"/";y(n);"/";z(n)
3040 PRINT : PRINT "Do kog datuma zelite saldo"
3050 INPUT vx: PRINT vx;"/";: INPUT vy: PRINT vy;"/";: INPUT vz:
 PRINT vz
3055 PRINT

3060 LET vd=(10000*vz)+(100*vy3060 LET vd=(10000*vz)+(100*vy)+v
x
3070 FOR s=1 TO n
3080 LET vn=(10000*z(s))+(100*y(s))+x(s)
3090 IF vn>vd THEN  GO TO 3190
3100 NEXT s
3190 LET s=s-1
3200 FOR h=1 TO s
3210 PRINT a$(h);" ";x(h);"/";y(h);"/";z(h);" ";a(h)
3220 LET to=to+a(h)
3230 NEXT h
3240 PRINT : PRINT "Saldo na dan ";vx;"/";vy;"/";vz;" = ";to
3250 PRINT : PRINT "Bilo koja dirka za OPCIJE"
3260 PAUSE 0
3270 GO TO 35
3999 STOP
4000 REM
4001 REM SNIMANJE STAVKI
4002 REM
4010 PRINT "Unesite ime pod kojim ce stavke biti snimljene"
4015 INPUT e$: PRINT : PRINT e$
4020 SAVE e$ LINE 35
4030 GO TO 35
5000 REM
5001 REM BRISANJE STAVKE
5002 REM
5005 PRINT "BRISANJE STAVKE": PRINT
5010 PRINT "Unesite ime stavke koju zelite  obrisati"
5020 INPUT 9$(1)
5030 PRINT : PRINT 9$(1)
5040 FOR 9=1 TO n
5050 IF 9$(1)=a$(9) THEN  GO TO 5500
5060 NEXT 9
5070 PRINT : PRINT "STAVKA NE POSTOJI"
5080 PRINT AT 21,0;"Bilo koja dirka za OPCIJE"
5090 PAUSE 0
5100 GO TO 35
```

```
5500 CLS
5510 PRINT "STAVKA PRONADJENA"
5513 PAUSE 100
5515 PRINT : PRINT "STAVKA IZBRISANA"
5518 PAUSE 100
5520 FOR u=9 TO n-1
5530 LET bi=u+1
5540 LET a$(u)=a$(bi): LET a(u)=a(bi): LET x(u)=x(bi): LET y(u)=
y(bi): LET y(u)=y(bi)
5550 NEXT u
5560 LET n=n-1
5570 GO TO 35
9000 REM
9001 REM SORTIRANJE STAVKI PO DATUMIMA
9002 REM
9005 IF n<2 THEN  GO TO 35
9007 LET t=(10000*z(n))+(100*y(n))+(x(n))
9010 FOR s=n TO 2 STEP -1
9030 LET r=(10000*z(s-1))+(100*y(s-1))+(x(s-1))
9040 IF t<r THEN  GO TO 9500
9050 GO TO 35
9060 NEXT s
9070 GO TO 35
9500 LET j=s-1
9510 LET q$=a$(s): LET q=a(s): LET qx=x(s): LET qy=y(s): LET qz=
z(s)
9520 LET a$(s)=a$(j): LET a(s)=a(j): LET x(s)=x(j): LET y(s)=y(j
): LET z(s)=z(j)
9530 LET a$(j)=q$: LET a(j)=q: LET x(j)=qx: LET y(j)=qy: LET z(j
)=qz
9540 GO TO 9060
```

LOTO

Ovaj kratak program će rešiti umesto vas koje ćete brojeve upisati u listić za loto. Računar će koristiti generator slučajnih brojeva i pomoću njega vam "predložiti" osam kolona sa po pet od trideset šest brojeva. Kada bude počeo da se primenjuje novi sistem izmenićete linije 15, 20 i 30:

```
15 DIM a(x)                          x= koliko se brojeva izvlači
20 FOR h=1 TO x
30 LET b=INT (y*RND)+1               y= od koliko se brojeva izvlači
```

Program snimite sa: SAVE "loto" LINE 1. Puno sreće!

```
 5 REM *** LoTo ***
10 BORDER 1: PAPER 1: INK 7: CLS
11 PRINT  INK 5;AT 6,0;"LoToLoToLoToLoToLoToLoToLoToLoToLoTo"
12 FOR f=1 TO 31 STEP 4: LET li=8
15 DIM a(5)
20 FOR h=1 TO 5
30 LET b=INT (36*RND)+1
50 FOR q=1 TO h: IF b=a(q) THEN  GO TO 30
55 NEXT q
60 LET a(h)=b: PRINT  BRIGHT 1;AT li,f;a(h): LET li=li+1: NEXT
h: NEXT f
```

KONVERTOR KODOVA

Onima koji se malo ozbiljnije bave računarom, sigurno će dobro doći ovaj program. On radi sledeće: unesite u računar broj u decimalnom, heksadecimalnom ili binarnom obliku i računar će vam ga predstaviti u sva tri oblika. Ovo je naročito korisno kada se radi sa nekim nižim programskim jezikom od bejzika. Ako ste zaboravili PZP sadrži i poglavlje MAŠINE GOVORE MAŠINSKI!

Program, po unošenju, snimite sa: SAVE "konvertkod" LINE 1.

```
9000 REM *** KONVERTOR KODOVA ***
9010 PRINT AT 10,7; FLASH 1;"ZAUSTAVITE TRAKU";AT 12,2;"ZA START
 - BILO KOJA DIRKA": PAUSE 0
9020 POKE 23658,8
9050 CLS : PRINT AT 3,0;"KOJI KOD ZELITE DA PRERACUNATE";''';TAB
 9;"1 - HEKSADECIMALNI";TAB 9;"2 - DECIMALNI";TAB 9;"3 - BINARNI
";''';TAB 9;"4 - ZAVRSETAK": PAUSE 0: IF CODE INKEY$<49 OR CODE I
NKEY$>52 THEN  BEEP .5,16: GO TO 9050
9060 LET W$=INKEY$: GO SUB (VAL W$*100)+9000: GO TO 9050
9100 CLS : PRINT AT 2,2;"UNESITE HEKSADECIMALNI BROJ   KOJI ZELI
TE DA PRERACUNATE I     PRITISNITE";TAB 2;"'ENTER'": INPUT  LINE
W$: FOR I=1 TO LEN W$: IF CODE W$(I)<48 OR CODE W$(I)>57 AND COD
E W$(I)<65 OR CODE W$(I)>70 THEN  BEEP .5,16: CLS : PRINT AT 3,5
;W$;" JE POGRESAN UNOS";''';TAB 5;"HEKSADECIMALNI ZNACI";TAB 5;"SU
 OD 1 DO 9 I OD A DO F.": GO SUB 9600: GO TO 9100
9110 NEXT I: LET D=0: LET W=LEN W$-1: FOR I=1 TO LEN W$: IF CODE
 W$(I)>64 AND CODE W$(I)<71 THEN  LET A=CODE W$(I)-55: GO TO 912
0
9115 LET A=VAL W$(I)
9120 LET P=A*16^W: LET D=D+P: LET W=W-1: LET A=0: NEXT I: LET Y$
=STR$ (D): LET B=2: LET P$="": GO SUB 9180: LET X$=J$: CLS : PRI
NT AT 2,1;"VAS HEKSADECIMALNI BROJ :-";''';TAB 2;W$;''';: GO SUB
9500: GO SUB 9510: GO SUB 9600: RETURN
9180 LET E=INT (D/B): LET C=(D/B)-INT (D/B): LET G=C*B: IF G>=10
 THEN  LET G$=CHR$ (G+55)
9182 IF G<10 THEN  LET G$=STR$ (G)
9184 LET P$=P$+G$: IF E<B THEN  GO TO 9190
9186 LET D=E: GO TO 9180
9190 LET J$="": FOR I=LEN P$ TO 1 STEP -1: LET J$=J$+P$(I): NEXT
 I: RETURN
9200 CLS : PRINT AT 2,2;"UNESITE DECIMALNI BROJ KOJI   ZELITE DA
 PRERACUNATE I ZATIM   PRITISNITE";TAB 2;"'ENTER'": INPUT  LINE
W$: FOR I=1 TO LEN W$: IF CODE W$(I)<48 OR CODE W$(I)>57 THEN  B
EEP .5,16: CLS : PRINT AT 3,5;W$;" JE POGRESAN UNOS"';TAB 4;"DEC
IMALNI ZNACI SU OD 0 DO 9": GO SUB 9600: GO TO 9200
9210 NEXT I: LET D=VAL W$: LET B=16: LET P$="": GO SUB 9180: LET
 Z$=J$: LET D=VAL W$: LET B=2: LET P$="": GO SUB 9180: LET X$=J$
: CLS : PRINT AT 2,1;"DECIMALNI BROJ JE :-";''';TAB 2;W$;''';: GO
 SUB 9500: GO SUB 9520: GO SUB 9600: RETURN
9300 CLS : PRINT AT 2,0;"UNESITE BINARNI BROJ KOJI ZELITEDA PRER
ACUNATE I PRITISNITE";TAB 2;"'ENTER'": INPUT  LINE W$: FOR I=1 T
O LEN W$: IF W$(I)<>"1" AND W$(I)<>"0" THEN  BEEP .5,16: CLS : P
RINT AT 3,5;W$;" JE POGRESAN UNOS";TAB 5;"BINARNI ZNACI SU 0 I 1
.": GO SUB 9600: GO TO 9300
9310 NEXT I: LET D=0: LET W=LEN W$-1: FOR I=1 TO LEN W$: LET E=V
AL W$(I): LET P=E*2^W: LET D=D+P: LET W=W-1: NEXT I: LET Y$=STR$
```

```
(D): LET B=16: LET P$="": GO SUB 9180: LET Z$=J$: CLS : PRINT A
T 2,1;"BINARNI BROJ JE :-"';TAB 2;W$;'''';: GO SUB 9510: GO SUB
 9520: GO SUB 9600: RETURN
9400 CLS : PRINT AT 10,8;"PROGRAM ZAVRSEN": STOP
9500 PRINT " BINARNI EKVIVALENT JE :-"';TAB 2;X$;'''';: RETURN
9510 PRINT " DECIMALNI EKVIVALENT JE :-"';TAB 2;Y$;'''';: RETUR
N
9520 PRINT " HEKSADECIMALNI EKVIVALENT JE :-"';TAB 2;Z$;'''';:
RETURN
9600 PRINT #1;"  BILO KOJA DIRKA ZA NASTAVAK  ": PAUSE 0: RETURN
```

POKAZATELJ MEMORIJE

Ovaj program vam u svakom momentu kazuje koliko vam je još ostalo slobodne memorije. Veći deo programa je uputstvo, a sama linija 9990 se koristi za očitavanje preostale memorije i možete je uneti u svaki vaš program u BASIC-u.

Snima se sa: SAVE "memonitor" LINE 1.

```
  5 REM *** POKAZATELJ MEMORIJE ***
 20 PRINT  BRIGHT 1;"POKAZATELJ SLOBODNE MEMORIJE"
 30 PRINT '''"LINIJA 9990 MOZE BITI SMESTENA  NA KRAJU VASEG PRO
GRAMA KAKO BI VAM POMOGLA DA VIDITE KOLIKO VAMJE PREOSTALO SLOBO
DNE MEMORIJE  DOK UNOSITE NEKI PROGRAM ILI DOKGA PROSIRUJETE."
 40 PRINT '"SAMA LINIJA ZAUZIMA 151 BAJT    DA BI JE KORISTILI
OTKUCAJTE:   GO TO 9990."
 50 PRINT '' BRIGHT 1;"BILO KOJA DIRKA ZA DEMONSTRACIJU": PAUSE
 0
9990 DEF FN z(z$)=VAL ("PEEK "+z$+"+256*PEEK ("+z$+"+1)"): PRINT
 "SLOBODNA MEMORIJA",FN z("23730")-FN z("23641")'"PROGRAM",FN z(
"23627")-FN z("23635")'"VARIJABLE",FN z("23641")-FN z("23627")-S
GN PI
```

TESTIRANJE RAM MEMORIJE

Ovo je kratak program koji će proveriti memoriju vašeg računara i izvestiti vas da li je ispravna ili ne. Program proverava samo ono područje memorije koje se koristi za BASIC programe i zato je poželjno da pre unošenja ovog programa u računar otkucata naredbu PRINT USR 0, koja će obrisati celu memoriju da bi bila proverena.

Program snimite sa: SAVE "ram test" Line 1.

```
  10 REM *** TESTIRANJE RAM MEMORIJE ***
  30 CLS : PRINT AT 3,0; BRIGHT 1;" * PROVERAVANJE RAM MEMORIJE
* "
  40 PRINT '''"OVAJ PROGRAM VAM OMOGUCAVA DA   PROVERITE ISPRAVNO
ST MEMORIJE   KOJU KORISTE VASI PROGRAMI      NAPISANI U BEJZIKU
."
  50 PRINT '' BRIGHT 1;" PRITISNITE BILO KOJU DIRKU ZA       POC
ETAK PROVERAVANJA       ": PAUSE 0: BEEP .1,35
  60 LET f=INT (((PEEK 23730+256*PEEK 23731)-(PEEK 23641+256*PEE
K 23642))/256-3)
  70 DIM a$(f,256): DIM b$(256)
  75 LET o=PEEK 23627+256*PEEK 23628+14
  80 FOR i=1 TO 256: LET b$(i)=CHR$ (i-1): NEXT i: FOR i=1 TO f:
 PRINT ".";: LET a$(i)=b$: NEXT i
  90 LET e=0: PRINT : FOR i=1 TO f: IF a$(i)=b$ THEN  GO TO 95
  91 BEEP .3,12: LET e=1: FOR j=1 TO 256: IF a$(i,j)<>CHR$ (j-1)
 THEN  PRINT "#";(i-1)*256+o+j-1;" JE ";CODE a$(i,j);" - A TREBA
 DA JE ";j-1
  92 NEXT j
  95 NEXT i
 100 IF NOT e THEN  PRINT ' BRIGHT 1;"MEMORIJA JE ISPRAVNA"
 110 IF e THEN  PRINT ' BRIGHT 1; FLASH 1;"MEMORIJA JE NEISPRAVN
A"
```

TUMAČ KODOVA

Program "Tumač kodova" je vodič kroz kodove vašeg računara. Sve što vidite na ekranu prikazano je pomoću takozvanog ASCII koda i ako želite da znate koji kod odgovara kome znaku pomoći će vam ovaj program.

Unesite ga u računar, proverite i snimite sa: SAVE "tumač koda" LINE 1. Dobro je da za ovakav i slične programe koristite kratke trake (do 10 minuta trajanja) koje sami sklapate od starih ispravnih kutija od kaseta i novog sadržaja (trake). Tako od jedne kasete C-45 možete da dobijete četiri manipulativne trake.

```
  1 REM *** TUMAC KODOVA ***
  2 BEEP .06,0: BORDER 1: PAPER 1: INK 7: CLS : POKE 23609,50
  3 PRINT AT 7,4;"K ..... TUMACI KOD";AT 9,4;"D ..... TUMACI DI
RKU";AT 11,4;"Z ..... ZA ZAVRSETAK"
  4 BEEP .1,10
  5 IF INKEY$="" THEN  GO TO 5
  7 IF INKEY$="z" OR INKEY$="Z" THEN  CLS : PRINT AT 12,10;"KRA
J PROGRAMA": STOP
 10 IF INKEY$="k" OR INKEY$="K" THEN  CLS : GO SUB 400
 12 IF INKEY$="d" OR INKEY$="D" THEN  CLS : GO SUB 300
 15 GO TO 5
150 FOR n=1 TO 8
155 LET Y=G/2
165 LET R=G-(INT Y*2)
175 LET A$=CHR$ (48+R)
177 PRINT AT 7+x,12-n;A$
178 IF A$="0" THEN  LET A$=" "
179 IF A$="1" THEN  LET A$="@@@"
180 PRINT AT 7+x,28-n;A$
185 LET G=INT Y
195 NEXT n
200 IF x<>8 THEN  RETURN
210 PRINT AT 21,0;"________________________________"
215 BEEP .4,0
220 IF INKEY$="" THEN  GO TO 220
222 RUN
299 REM INKEY$ INPUT
300 BEEP .5,5
303 PRINT #1;" PRITISNITE NEKU DIRKU"
305 IF INKEY$="" THEN  GO TO 305
307 CLS
310 LET Q$=INKEY$
311 BEEP .03,10
312 PRINT AT 5,0;"ODGOVARAJUCI ZNAK:";" ";Q$;"   (KOD:";CODE Q$
;")";
325 FOR x=1 TO 8
330 LET J=15615+8*(CODE Q$-32)+x: LET G=PEEK J
332 LET G=PEEK J
340 GO SUB 150
350 NEXT x
399 REM UNOSENJE KODA
400 BEEP .5,20
405 INPUT "UNESITE KOD : ";CODE
411 BEEP .01,10
```

```
 414 IF CODE<32 OR CODE>127 THEN  GO TO 500
 420 PRINT AT 5,0;"ODGOVARAJUCI ZNAK:";" ";CHR$ CODE;"   (KOD:";
CODE;")";
 425 FOR x=1 TO 8
 430 LET J=15615+8*(CODE-32)+x: LET G=PEEK J
 440 GO SUB 150
 450 NEXT x
 500 REM NEISKORISCENI KODOVI
 510 IF CODE<6 OR (CODE>23 AND CODE<32) THEN  PRINT AT 5,0;"  KO
D SE NE KORISTI    ";"(KOD:";CODE;")": GO TO 210
 520 IF CODE<24 AND CODE>5 THEN  PRINT AT 5,0;"KONTROLNI KOD  ";
"(KOD:";CODE;")": GO TO 210
 530 IF CODE<144 THEN  GO TO 600
 540 IF CODE<165 THEN  GO TO 700
 550 IF CODE<256 THEN  PRINT AT 5,0;CHR$ CODE;"    (KOD:";CODE;"
)": GO TO 210
 560 PRINT AT 5,0;"KODOVI SU OD 0 DO 255": GO TO 210
 600 REM GRAFICKI SIMBOLI
 610 PRINT AT 5,0;"GRAFICKI SIMBOL: ";CHR$ CODE;"   ";"(KOD :";C
ODE;")": GO TO 210
 700 REM USR GRAFIKA
 710 PRINT AT 4,0;"KORISNICKA GRAFIKA:  ";CHR$ CODE;"
          (KOD :";CODE;")"
 715 FOR x=1 TO 8: LET J=65368+8*(CODE-144)+X-1: LET G=PEEK J
 740 GO SUB 150
 750 NEXT x
```

LISTA PROMENLJIVIH

Ovaj program će vam biti od koristi prilikom programiranja u BASIC-u, pogotovo kada budete pravili duge programe sa mnogo promenljivih. Ovaj program možete uneti kao potprogram i kada ga pozovete, on će vam ispisati na ekranu sve promenljive koje koristite u vašem programu. Koliko je to korisno shvatićete kad ne budete morali da pretražujete ceo program da bi videli da li ste već iskoristili neki naziv za promenljivu ili ne, već vam računar uštedi to vreme izvršavajući to umesto vas.

Program je kratak tako da ne zauzima mnogo mesta u memoriji i možete ga koristiti sa vašim programom.

Snimićete ga sa: SAVE "varijable" LINE 1.

```
  10 REM *** LISTA VARIJABLI ***
8010 LET v0=VAL "PEEK 23627+256*PEEK 23628"
8020 LET v1=PEEK v0: LET v2=VAL "INT (v1/32)": LET v$=CHR$ VAL "
v1-v2*32+96": LET v0=v0+SGN PI: IF VAL "v2=2 OR v2=4 OR v2=6" TH
EN  LET v3=VAL "PEEK v0+256*PEEK (v0+1)+2": GO TO VAL "8070"
8030 IF VAL "v2=3" THEN  LET v3=VAL "5": GO TO VAL "8070"
8040 IF VAL "v2=7" THEN  LET v3=VAL "18": GO TO VAL "8070"
8050 IF PEEK v0<VAL "128" THEN  LET v$=v$+CHR$ PEEK v0: LET v0=v
0+SGN PI: GO TO VAL "8050"
8060 LET v$=v$+CHR$ VAL "PEEK v0-128": LET v3=VAL "6"
8070 IF v$="v0" THEN  STOP
8080 PRINT v$;"$" AND VAL "v2=2 OR v2=6";"%" AND VAL "v2=7";
8090 IF VAL "v2=4 OR v2=6" THEN  PRINT "(": FOR v=VAL "2" TO VA
L "2*PEEK (v0+2)" STEP VAL "2": PRINT VAL "PEEK (v0+v+1)+256*PEE
K (v0+v+2)";",": NEXT v: PRINT CHR$ VAL "8";")";
8095 PRINT : LET v0=v0+v3: GO TO VAL "8020"
```

ALARM

Ovaj program (kao što pretpostavljate) služi da bi vas upozorio na nešto, probudio... Jedino što je potrebno za njegovo korišćenje je da mu upišete trenutno vreme i vreme u koje će se računar oglasiti zvučnim signalima. Naravno, pre toga treba otkucati program i snimiti ga sa: SAVE "ALARM". Startovanje se vrši sa RUN.

```
90 S=54272:FORR=STOS+24:POKER,0:NEXT
95 GOSUB195
100 PRINT"■POSTAVKA VREMENA ZA ALARM"
110 PRINT"■( HHMMSS )"
120 INPUT"■  000000000000000";TI$
130 POKE956,PEEK(160)
140 POKE957,PEEK(161)
150 PRINT"■UNESITE TRENUTNO VREME"
160 PRINT"■( HHMMSS )"
170 INPUT "■  000000000000000";TI$
180 PRINT"■":SYS49152:GOTO180
195 FORG=49152TO49284:READE:POKEG,E:NEXT:RETURN
200 DATA120,173,20,3,141,186,3,173,21,3,141
210 DATA187,3,169,25,141,20,3,169,192,141
220 DATA21,3,88,96,173,160,0,205,188,3
230 DATA208,92,173,161,0,205,189,3,208,84
240 DATA169,145,141,17,4,169,149,141,18,4
250 DATA169,137,141,19,4,169,148,141,20,4
260 DATA169,161,141,21,4,169,15,141,24,212
270 DATA169,9,141,5,212,169,6,141,6,212
280 DATA169,34,141,1,212,169,70,141,0,212
290 DATA169,33,141,4,212,169,255,160,255,136
300 DATA208,253,202,208,248,169,0,141,24,212
310 DATA120,173,186,3,141,20,3,173,187,3
320 DATA141,21,3,88,76,49,234,134,223,32
330 DATA223,0,223,32,223,32,223,32,223,0
```

KOPIRANJE BLOKA

Program za one koji poseduju disk-jedinicu. Radi na taj način što uzima sa diskete količinu memorije od jednog bloka (oko 250 bajta). Dok program radi neprekidno vas obaveštava šta treba da uradite ili vas pita za odluke. Otkucani program snimite sa: SAVE "KOPIRANJE BLOKA". Za start — samo RUN.

```
1 PRINT"■ KOPIRANJE BLOKA 1540/41■■"
5 N$=CHR$(0):B=8192:GOTO10
8 PRINT"■IBACI ORIGINAL !!!"
9 GET#3:IF#$=""THEN9
10 OPEN1,8,15:OPEN2,8,2,"#"
12 INPUT"TRAKA ";TR:INPUT"SEKTOR ";SR
15 PRINT"LISTA  ? I/N"
17 GETAN$:IFAN$=""THEN17
20 PRINT#1,"U1:"2;0;TR;SR:PRINT#1,"B-P:"2;0
100 FORI=0TO255:GET#2,A$:IFA$=""THENA$=N$
105 POKEB+I,ASC(A$)
130 NEXT:CLOSE2:CLOSE1
135 IFAN$="I"THENGOSUB500
140 INPUT"KOPIRANJE I/N ";B$
150 IFB$<>"I"THEN10
160 PRINT"■UNESI DISKETU ZA KOPIRANJE":PRINT"■■UGME"
170 GETB$:IFB$=""THEN170
180 OPEN1,8,15,"I":OPEN2,8,2,"#"
190 PRINT#1,"B-P:"2;0
200 FORI=0TO255:PRINT#2,CHR$(PEEK(B+I));:NEXT
210 PRINT#1,"U2:"2;0;TR;SR
220 CLOSE2:CLOSE1:GOTO8
500 FORI=BTOB+255:A$=CHR$(PEEK(I)):PRINTI-B;
510 PRINTASC(A$);:IFASC(A$)>31ANDASC(A$)<128THENPRINTA$:GOTO520
515 PRINT
520 NE$=CHR$(13):INPUT"■■■■■■■■■■■■■■■■NOV:";NE$
522 IF NE$<>CHR$(13)AND NE$<>"IZLAZ"THEN GOSUB 600
525 IFNE$="EXIT"THENI=B+255:GOTO530
530 NEXT:RETURN
600 IFLEN(NE$)>1THENPOKEI,ASC(NE$):RETURN
610 POKEI,ASC(NE$):RETURN
620 POKEI,ASC(NE$):RETURN
```

FLIST

Kao što će vam i sam program napisati, program sa ove liste služi tome da se kreira nova naredba na vašem računaru. Ako vam se nekad desilo da se slabo snalazite u nepreglednom listingu, evo rešenja. Samo otkucate FLIST i pritisnete RETURN i dobićete listing u kome su sve naredbe u jednoj programskoj liniji napisane jedna ispod druge. Snimanje: SAVE "FLIST". Startovanje: RUN.

```
1 PRINT"**** FLIST ****"
2 FORR=1TO1500:NEXTR
3 PRINT"PROGRAM PROSIRUJE BASIC ZA NAREDBU >FLIST<":PRINT
4 PRINT" ONA FORMATIZUJE LISTING NA EKRANU TAKO"
5 PRINT" DA SVE NAREDBE ODVOJENE SA ':' DOLAZE U NOVI RED"
10 FORI=828TO962::   READJ:    POKEI,J::NEXTI::SYS828::END
100 DATA169,71,141,8
110 DATA3,169,3,141
120 DATA9,3,96,32
130 DATA115,0,201,155
140 DATA240,10,201,70
150 DATA240,25,32,121
160 DATA0,76,231,167
170 DATA169,26,141,6
180 DATA3,169,167,141
190 DATA7,3,32,115
200 DATA0,32,156,166
210 DATA76,174,167,32
220 DATA115,0,201,155
230 DATA240,3,76,8
240 DATA175,169,130,141
250 DATA6,3,169,3
260 DATA141,7,3,76
270 DATA98,3,8,36
280 DATA15,48,4,201
290 DATA58,240,4,40
300 DATA76,26,167,72
310 DATA138,72,152,72
320 DATA160,2,177,95
330 DATA133,99,200,177
340 DATA95,133,98,162
350 DATA144,56,32,73
360 DATA188,32,223,189
370 DATA32,135,180,32
380 DATA166,182,170,169
390 DATA13,32,12,225
400 DATA32,59,171,202
410 DATA208,250,104,168
420 DATA104,170,104,40
430 DATA76,26,167
```

GOLF

Svima nama dobro poznata igra (iz filmova) — sada na računaru. Bez treninga, bez napora, u trci za što bolji rezultat (manje pogrešnih pokušaja). U igri je korišćena grafika sa tastature ali to vas neće omesti u nadmetanju sa prijateljima za prvo mesto. Snimićete ga sa: SAVE "GOLF". Naravno, startovanje je sa RUN.

```
1 PRINT "GOLF"
30 C=0
60 DIMT$(3)
70 FORZ=1TO9
80 SC=0
100 LETJ=INT(RND(1)*12) :
110 LETO=INT(RND(1)*4)+2:
120 PRINTCHR$(147):
130 PRINT
140 PRINT"###BROJ RUPE";Z;"###"
150 PRINT
160 PRINT "TEZINA RUPE >> ";O
170 PRINT
180 PRINT
190 GOSUB 430
200 PRINT
210 PRINT "UDARAC ";:INPUTA
215 PRINT CHR$(147):
220 IFJ>24THENA=-A
230 LETJ=J+INT(A/(RND(1)*O+1)):
240 GOSUB430
250 SC=SC+1
260 PRINT
270 PRINT "POSLE OVOG UDARCA REZULTAT TI JE ";SC
280 PRINT
290 IFJ<>24THEN200
300 GOSUB510
310 C=C+SC
320 PRINT
330 PRINT "REZULTAT ZA";Z;" RUPE  JE ";C
340 FORM=1TO999:NEXTM
350 FORL=1TO20
360 FORY=1TOL
370 PRINT " ";
380 NEXT Y
390 PRINT "PRIPREMA"
400 NEXTL
410 NEXTZ
420 GOTO570
430 IFJ>30THENJ=30
440 FORK=1TOJ-1
450 PRINT " ";
460 NEXTK
470 PRINT "●"
480 PRINT "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ ~~~~~~~~~~~~"
490 PRINT
500 RETURN
510 FORL=1TO10
520 PRINT "UBACIO  SI  U  ";SC;" UDARACA"
530 NEXTL
540 PRINT
550 PRINT "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~●~~~~~~~~~~~~~"
560 RETURN
```

ELEKTRONSKI KLAVIR

Program koji demonstrira muzičke mogućnosti vašeg računara. Uz to, vaša tastatura postaje klavijatura. Mogućnost sviranja u više oktava, izmena talasnog oblika zvuka, jačine... Samo pojačajte zvuk na vašem TV prijemniku (monitoru) i prepustite se eksperimentisanju u računarskom muziciranju. Program se snima sa: SAVE "ELEKTRO-KLAVIR". Startuje se sa RUN.

```
100 REM ELEKTRONSKI KLAVIR
110 PRINT"⌐ ▮ ▮▮ ▮▮ | ▮▮ ▮▮ ▮▮ | ▮▮ ▮▮ | ▮▮ ▮▮ ▮▮ .| "
120 PRINT" ▮ ▮▮ ▮▮ | ▮▮ ▮▮ ▮▮ | ▮▮ ▮▮ | ▮▮ ▮▮ ▮▮ | "
130 PRINT" ▮ ▮▮ ▮▮ | ▮▮ ▮▮ ▮▮ | ▮▮ ▮▮ | ▮▮ ▮▮ ▮▮ | "
140 PRINT" ▮ | | | | | | | | | | | | | | |"
150 PRINT" ▮Q|W|E|R|T|Y|U|I|O|P|@|*|↑|:|=|"
160 PRINT"▮     'SPACE'    ':      PEDALA"
170 PRINT"▮'F1','F3','F5','F7': OKTAVA"
180 PRINT"▮'F2','F4','F6','F8': TALASNI OBLIK"
190 PRINT"****************************************"
200 SI=13*4096+1024:DIMF(26):DIMK(255)
210 FORI=0TO28:POKESI+I,0:NEXT
220 F1=7939:FORI=1TO26:F(27-I)=F1*5.8+30:F1=F1/2↑(1/12):NEXT
230 K$="1Q2W3ER5T6Y7UI9O0P@-*£↑▮:="
240 FORI=1TOLEN(K$):K(ASC(MID$(K$,I)))=I:NEXT
250 PRINT"⌐
260 AN=0:AB=0:HA=15:AU=9:HH=HA*16+AU:AS=AN*16+AB:WF=16:M=1
261 OK=4:HB=256: Z=0
270 FORI=0TO2:POKESI+5+I*7,AS:POKESI+6+I*7,HT
280 POKESI+2+I*7,4000AND255:POKESI+3+I*7,4000/256:NEXT
290 POKESI+24,15+16+64:POKES+23,7
300 GETA$:IFA$=""THEN300
310 FR=F(K(ASC(A$)))/M:FL=SI+V*7:W=FL+4:IFFR=ZTHEN420
320 POKEFL+6,Z
330 POKEFL+5,Z
340 POKEW,8:POKEW,0
350 POKEFL,FR-HB*INT(FR/HB)
360 POKEFL+1,FR/HB
370 POKEFL+6,HH
380 POKEW,WF+1:FORI=1TO50*AN:NEXT
390 POKEW,WF
400 IFP=1THENV=V+1:IFV=3THENV=0
410 GOTO300
420 IFA$="■"THENM=1:OK=4:GOTO300
430 IFA$="■"THENM=2:OK=3:GOTO300
440 IFA$="■"THENM=4:OK=2:GOTO300
450 IFA$="■"THENM=8:OK=1:GOTO300
460 IFA$="■"THENWF=16:GOTO300
470 IFA$="■"THENWF=32:GOTO300
480 IFA$="■"THENWF=64:GOTO300
490 IFA$="■"THENWF=128:GOTO300
500 IFA$=" "THENP=1-P:HH=(HHANDNOT2)OR(NOTHHAND2):GOTO300
510 IFA$="⌐"THEN210
520 GOTO300
```

KALENDAR

Da li vam se desilo da ne možete naći svoj kalendar. Sada ta situacija za vas predstavlja prošlost: sada samo učitajte ovaj program, startujte ga, otkucate željeni mesec i računar će vam ispisati ono što vam treba. Problema sa kalendarima više ne bi trebalo da bude. Kalendar se snima sa: SAVE "KALENDAR". Startovanje kao i kod prethodnih programa — RUN.

```
5 POKE53280,0:POKE53281,0
10 PRINT"■■"TAB(17)"KALENDAR":PRINT
15 DIM M$(12):FORI=1TO12:READM$(I):NEXT
20 DATA JANUAR,FEBRUAR,MART,APRIL,MAJ,JUNI,JULI,AVGUST
21 DATA SEPTEMBAR,OKTOBAR
25 DATA NOVEMBAR,DECEMBAR
30 PRINT"PROGRAM STAMPA KALENDAR BILO KOG MESECA"
40 PRINT"GODINE IZMEDJU 2000 PRE NOVE ERE I"
42 PRINT"2000G NOVE ERE":PRINT
45 PRINT"UNETI - ZNAK ISPRED GODINE PRE NOVE ERE":PRINT
46 PRINT"UNETI BUC NULE ZA KRAT PROGRAMA":PRINT
50 PRINT"GODINU I MESEC UNETI U OBLIKU GGGG,MM":PRINT
100 INPUT"■■■■■■■■■■KOJU■GODINU I MESEC ZELITE ";Y,M
110 IFM<0ORM>12ORY<-2500ORY>2000THEN100
115 IFM=0ORY=0THENEND
120 I=Y:O$="NOVE ERE":IFY<0THENO$="PRE "+O$:I=-I:Y=Y+1
130 PRINT"■MESEC ";M$(M);I;"■■ GODINE "O$:PRINT
150 GOSUB1000:I=I:PRINT
160 PRINTTAB(8);"■N■■ P U O Č P S":PRINT
170 M=M+1:IFM>12THENM=1:Y=Y+1
180 GOSUB1000:N=I-I:I=I-INT(I/7)*7+1
185 IFI=7THENI=0
186 I=I*3+7:K=1
187 IFY<>1752ORM<>10THEN190
188 PRINTTAB(I);"  1  2":K=14:I=19:N=30
190 FORI=KTON:PRINTTAB(I)"
200 IFI>9THENPRINT"■■";
205 IFI=7THENPRINT"■";
210 PRINTI"■";
220 I=I+3:IFI>27THENPRINT:PRINT:I=7
230 NEXT:PRINT:PRINT:GOTO100
1000 K=Y+4712:I=INT(K/4)+365*K
1010 N=30.6*M-32.3
1020 IFM>2THEN1040
1030 N=N+2.3:IFK-INT(K/4)*4=0THENI=I-1
1040 I=I+INT(N+1)
1050 IFI<2261021THENRETURN
1060 K=Y-300
1070 IFM<3THENK=K-1
1080 N=INT(K/100)
1090 I=I-INT(.75*N)-1:RETURN
```

BAR GENERATOR

Ovaj program omogućuje da računar postane jedan od osnovnih aparata svih TV-mehaničara. Taj aparat se zove Bar generator pa je stoga i sam program dobio taj naziv. Veoma je koristan i za one koji nisu TV-mehaničari. Snimak na traci ovog programa pravite sa SAVE "BAR GENERATOR". Za početak kontrolisanja televizora otkucajte RUN.

```
10 PRINT"■□":POKE53280,0:POKE53281,0
15 PRINT"■■■■■■";TAB(5)"*****************************"
16 PRINTTAB(5)"*":PRINTTAB(33)"*"
20 PRINTTAB(5)"* B A R   G E N E R A T O R *"
22 PRINTTAB(5)"*":PRINTTAB(33)"*"
24 PRINTTAB(5)"*****************************"
30 PRINT"
40 FORN=0TO5000:NEXT
100 PRINT"□■■":POKE53280,0:POKE53281,0
105 PRINT"■";
110 PRINTTAB(7)"■ F1 ■        CISTOCA BOJA
120 PRINT:PRINT:PRINT
130 PRINTTAB(7)"■ F3 ■        OSTRINA-FOKUS"
140 PRINT:PRINT:PRINT
150 PRINTTAB(7)"■ F5 ■        B A R"
160 PRINT:PRINT:PRINT
170 PRINTTAB(7)"■ F7 ■        KONVERGENCIJA"
172 PRINT:PRINT:PRINT
173 PRINTTAB(7)"■ *  ■        KONTROLA TONA"
180 PRINT:PRINT:PRINT
190 PRINTTAB(7)"■ SPACE ■     M E N I "
200 GETA$:IFA$=""THENGOTO200
210 IFA$="■"THEN GOTO 1000
220 IF A$="■"THENGOTO2000
230 IF A$="■"THENGOTO3000
240 IFA$="■"THENGOTO4000
245 IFA$="*"THEN GOTO5000
250 GOTO200
1000 PRINT"□":POKE53280,C:POKE53281,C
1010 GETA$:IFA$=""THENGOTO1010
1020 IFA$="■"THENC=C+1
1030 IFC>15THENC=0
1035 IFA$=" "THENGOTO100
1040 IFA$<>"■"THENGOTO1010
1050 GOTO1000
2000 PRINT"□":POKE53280,0:POKE53281,0
2010 FORN=1024TO2023
2020 POKEN,46:NEXT
2030 GETA$:IFA$=""THENGOTO2030
2040 IFA$=" "THENPRINT"□":GOTO100
2050 IFA$<>" "THEN GOTO2030
3000 PRINT"□";:POKE53280,1:POKE53281,1
3010 FOR R=0TO23
3020 PRINT"■■     ■■     ■■     ■■     ■■     ■■     ■■     ■■"
3025 NEXT
3027 PRINT" RED  GRN  BLU  CYN  PUR  YEL  ORG  GRY";
3030 GETA$:IFA$=""THENGOTO3030
3040 IFA$=" "THENPRINT"□":GOTO100
3050 IFA$<>" "THEN GOTO2030
4000 PRINT"□":POKE53280,0:POKE53281,0
4010 FORN=1024TO2023
4020 POKEN,43:NEXT
4030 GETA$:IFA$=""THENGOTO2030
```

IX / BIBLIOGRAFIJA

knjige:

(The) BASIC HANDBOOK, D. A. Lien
Compusoft Publishing, San Diego, USA
BASIC-z Pocket Guide, R. Hunt
Pitman Publishing, London, Eng
BASIC PROGRAMMING, D. Spencer
Charles Scribner's Sons, New York, USA
40 BEST MASHINE CODE ROUTINES FOR THE ZX SPECTRUM, J. Hardman
Hewson Consultants, Wallingford, Eng
MASHINE CODE FOR BEGINNERS (Z80, 6502), L. Watts, M. Wharton
Usborne Publishing Ltd, London, Eng
MASHINE CODE FOR BEGINNERS — 6502, AP Stephenson
Butterworth & Co, Sevenoaks, Eng
PROGRAMMING — Pocket guide, J. Shelley,
Pitman Publishing, London, Eng
20 BEST PROGRAMS FOR THE ZX SPECTRUM, A. Hewson
Hewson Consultants, Wallingford, Eng
THINGS TO DO WITH YOUR COMMODORE 64, J. Willis
The New American Library, New York, USA
UNDERSTANDING BASIC, R. Peddicord
Alfred Publishing Co, Sherman Oaks, USA

časopisi:

BYTE, USA
PERSONAL COMPUTER WORLD, Eng

PROGRAMIRANJE ZA POČETAK

. . . PROGRAMIRANJE ZA POČETAK nesumnjivo je pričrunik koji može korisno da posluži za pripremu i ulazak u tajne programiranja. On to može jer je izlaganju/učenju prišao sa relativno neobičnog pristupa. Uobičajeno je da se u ovakvim priručnicima, to je i u našoj literaturi prisutno, počne od običnih, najčešćih komandi i da se onda ide ka principima programiranja. PROGRAMIRANJE ZA POČETAK je, može se reći, strukturiran na mnogo prihvatljiviji način za početnike. Dajući opšte koncepte na početku, razjašnjavajući opšte funkcije programiranja preko BASIC-a mašinca, algoritma, drugih jezika, preko programa za Commodore i ZX Spectrum-a dolazi se do naredbi za razne vrste mašina (hardvera). Na taj način se poštuje jedno didaktičko pravilo da se preko opštih znanja dolazi do njihove konkretne primene u raznim domenima. Na prvi pogled to izgleda nešto teže nego učenje od komandi do programiranja. Samo na prvi pogled. Onda kada se savladaju opšta znanja, opšta logička pravila, kada se savlada celina koncepta programiranja, ili barem njegovi osnovni elementi, onda programiranje postaje daleko pristupačnije i jednostavnije i može se primeniti na razne mašine. To je upravo put kojim ide PROGRAMIRANJE ZA POČETAK. Zbog toga se čini da je ovaj priručnik dobar i koristan i da zaslužuje preporuke.

Prof. dr Vladimir Štambuk

KORISNE KNJIGE
BEOGRAD