MIKRORAČUNALO ORAO 102

PRIRUČNIK

VELEBIT
OOUR INFORMATIKA

PEL
OOUR ELEKTRONIKA

Autori:
PONGRAČIĆ IVAN — ZEBEC BRANKO

MIKRORAČUNALO ORAO 102
PRIRUČNIK

SADRŽAJ

UVOD

U ovom priručniku je opisan postupak povezivanja mikroračunala "ORAO" s pripadajućim perifernim jedinicama (štampač, kazetofon...), te način programiranja u BASIC-u i miniassembleru. Priručnik je organiziran u pet dijelova:

- način početnog korištenja
- programiranje u BASIC-u
- opis i korištenje MONITOR-a
- opis načina adresiranja mikroprocesora 6502
- prikaz mnemoničkih instrukcija 6502

Za sve korisnike koji još nisu imali priliku raditi na računalima, preporuča se pažljivo praćenje uvodnog dijela ovog priručnika, kako bi se izbjegle nepotrebne neugodnosti, koje su posljedica nepravilnog rukovanja.
Mikroračunalo "ORAO" uglavnom je namijenjeno za korištenje u BASIC-u, pa će stoga veći dio priručnika biti usmjeren na objašnjenje BASIC naredbi, uključujući i načine njihovog korištenja. Za one koje će zanimati detaljniji rad mikroračunala i mikroprocesora 6502 predviđen je rad u MONITOR-u s mogućnošću korištenja programiranja u strojnom jeziku.
Zaključno žele autori napomenuti, da ovaj priručnik nije zamišljen kao udžbenik BASIC-a ili ASSEMBLERA, već samo kao pomagalo prilikom savladavanja osnovne problematike rada na mikroračunalu "ORAO". Za sve one kojima je sadržaj priručnika suviše skroman, preporučujemo nabavku specijaliziranih knjiga za učenje BASIC-a i ASSEMBLERA.

POGLAVLJE 1

1.1 PRIKLJUČIVANJE MIKRORAČUNALA "ORAO"

Za vizualno praćenje rada na mikroračunalu "ORAO" predviđeno je korištenje kućnog TV-prijemnika ili video monitora. Ukoliko koristimo TV-prijemnik, potrebno ga je podesiti na područje oko devetog kanala, a zatim ga priloženim TV-kablom vezati na antenski izlaz mikroračunala (Slika 1.1-2). Korisnici sa većim zahtjevima mogu koristiti video monitor, koji se spaja na video izlaz,pri je spomenutim kablom (Slika 1.1-1).

1.priključak za video monitor
2.priključak za TV-prijemnik
3.priključak za kazetofon
4.priključak za štampač (printer)
5.konektor za proširenje
6.RESET tipka
7.prekidač mrežnog napona
8.kabel mrežnog napona

Slika 1.1

Za uključenje računala potrebno je potisnuti prekidač mrežnog napona (Slika 1.1-7) prema gore. Pritiskom na RESET tipku (Slika 1.1-6) pokreće se mikroračunalo. Na TV-prijemniku (ako je upaljen) pojavit će se na vrhu ekrana poruka:

```
                    * * * ORAO * * *

*_
```

Razlog mogućeg izostanka slike može biti u nepodešenosti TV-prijemnika. Zbog toga je obično potrebno naknadno podešavanje slike, dok se ne dobije najpovoljnija oštrina prikazanih znakova. Zvjezdica (*) ispod poruke i zmigajuća crtica (kursor) označuju da se računalo nalazi u MONITOR-u. Kursor pokazuje mjesto na ekranu, gdje će se upisati slijedeći znak unesen s tastature.
Mikroračunalo "ORAO" je uglavnom predviđeno za korištenje u BASIC-u, pa će zbog toga biti potrebno kontrolu računala prebaciti u BASIC. Upišemo li

```
*BC
```

i pritisnemo tipku [CR], na ekranu će se pojaviti:

```
MEMORIJA ? pritisnimo [CR]
DULJINA LINIJE ? pritisnimo [CR]
       7167 LOKACIJA
>
_
```

Dodatna objašnjenja u vezi s ispisanim porukama bit će naknadno opisana. Žmigajući kursor ispod "prompta" (]) označuje da je računalo u BASIC-u. Pritiskom na neku tipku dobit ćemo ispis odgovarajućeg znaka na ekranu. Pritisnemo li tipku A

```
>
A
```

A pojavit ce se znak A na ekranu. Pritiskom na [CR] računalo ispiše:

```
? SN GREŠKA
```

Naime, računalo ne prepoznaje A,niti kao naredbu, niti kao komandu, što rezultira ispisom greške. Unesimo slijedeći program:

```
PRINT"SVAKI POČETAK JE TEŽAK" [CR]
```

Računalo će ispisati

```
SVAKI POČETAK JE TEŽAK
```

Primjećujemo da je svaki znak unutar navodnika pravilno izveden i ispisan bez poruke o grešci. Želimo li da računalo izvede neku računsku operaciju u izravnom izvođenju, upišimo:

```
PRINT 23*124 [CR] 2852
```

Dobiveni broj 2852 predstavlja umnožak između brojeva 23 i 124. Za višekratno ispisivanje istog teksta poslužit ćemo se slijedećim programom;

```
FOR 1=0 TO 10:PRINT"JA SAM U PETLJI":NEXT I
```
pritisnimo [CR]

Slijedi ispis na ekranu:

```
JA SAM U PETLJI
JA SAM U PETLJI
JA SAM U PETLJI
JA SAM u PETLJI
JA SAM u PETLJI
JA SAM u PETLJI
JA SAM u PETLJI
JA SAM u PETLJI
JA SAM u PETLJI
JA SAM u PETLJI
JA SAM u PETLJI
```

U ovom programu možemo uočiti mogućnost pisanja više naredaba u s time da su naredbe odvojene dvotočkom (:).
Beskonačno ispisivanje teksta možemo postići vrlo lako

```
10 PRINT"BBSKONAČMA PETLJA":GOTO 10
```

Ovaj program je upisan u instrukcijskoj liniji 10. Za izvođenje programa takvog oblika moramo poslije pritiska na [CR] unijeti naredbu RUN

```
RUN
```

Ispisivanje poruke "BESKONAČNA PETLJA" ispunit će ekran. Zaustavljanje izvođenja omogućuje [CTL]C ili tipka RESET koja vraća kontrolu u MONITOR. Ponovni ulazak u BASIC postižemo s

```
*BW
```

To je takozvani vrući start, koji ne uništava postojeći BASIC program prethodno unesen.
Mikroračunalo "ORAO" daje korisniku mogućnost rada u grafici 256x256 (mogućih) točaka na ekranu. U tu svrhu postoje naredbe u BASIC-u, Koje na jednostavan način omogućavaju crtanje željene slike. Jednostavnim programom možemo nacrtati dijagonalu na ekranu,

```
MOV 0,0:DRAW 255,255 [CR]
```

Želimo li nacrtati samo jednu točku u donjem desnom kutu, dovoljno je upisati

```
PLOT 255,0 [CR]
```

1.2 SLOVIŠTE (tastatura)

Slovište na mikroračunalu "ORAO" ima standardni raspored tipki uključivši i znakove ć,č,š,ž,đ. Osim uobičajenih znakova, Slovište sačinjava i nekoliko specijalnih tipki koje su uočljive na slici 1.2.

Slika 1.2

Primjećujemo da se radi o slijedećim tipkama:

[CR] Return tipka
[CTL] Kontrol tipka
[ ] Šift tipka

Tipku [CR] smo već upoznali prilikom izvođenja prethodnih programa. Prilikom jednostavnog programiranja (bez brojeva linije BASIC programa), pomoću tipke [CR] izravno izvodimo napisani program.
[CTL] tipka služi za izvođenje nekih specijalnih operacija. Na primjer, istovremeno pritisnimo [CTL] i tipku L. Rezultat će biti brisanje ekrana. Slijedeća tabela daje popis svih [CTL] postojećih operacija na mikroračunalu "ORAO":

[CTL] L	briše ekran
[CTL] G	zvučni signal "beep"
[CTL] F	briše tekst od kursora do kraja ekrana
[CTL] H	pomiče kursor za jedno mjesto u lijevo
[CTL] K	pomiče kursor za jedan redak prema gore
[CTL] I	pomiče kursor za jedno mjesto u desno
[CTL] J	pomiče kursor za jedan redak prema dolje
[CTL] V	zvučni signal "beep" prilikom pritiska na neku tipku
[CTL] E	briše tekst od kursora do kraja linije
[CTL] B	uključuje štampač (printer)
[CTL] U	isključuje štampač
[CTL] D	vraća kursor na početak ekrana
[CTL] M	vraća kursor na početak reda (carriage return)
[CTL] C	prekid izvođenja programa (escape)

Poneke tipke imaju dva znaka. Donji znak ispisuje se izravnim pritiskom na tipku, dok ispis gornjega znaka dobijemo koristeći tipku šift [ ],

1	ili	!
2	ili	"
3	ili	#
4	ili	$
5	ili	%
6	ili	&
7	ili	'
8	ili	(
9	ili	)
-	ili	=
:	ili	*

i tako dalje.

Pomicanje kursora možemo izvoditi izravno, bez korištenja [CTL] kodova. U tu svrhu služe četiri tipke smještene u gornjem desnom kutu slovišta

kursor lijevo	[←][↑]	kursor gore
kursor dolje	[↓][→]	kursor desno

Preostale četiri tipke su tzv. funkcijske tipke.
[PF1] [PF2] [PF3] [PF4]
Pritiskom na [PF1] računalo ispisuje mala slova. Povratak ispisivanja velikih slova postižemo ponovnim pritiskom na [PF1].
[PF2] služi za startanje štampača (printera)
[PF3] prebacuje ispis znakova u invertiranom obliku
[PF4] je kopi (COPY) tipka

Ukoliko držimo neku tipku pritisnutu dulje vrijeme, ispis znakova će se ponavljati samostalno (autorepeat).

1.3 PRIMJER BASIC PROGRAMA

Pokušajmo napisati program, koji će na ekranu nacrtati funkciju sinus. Za razliku od prethodnih programa, kod ovoga ćemo svaku instrukcijsku liniju označiti odgovarajućim brojem.

```
10 FOR 1=0 TO 255
20 Y=100*SIN(I/40)+128
30 PLOT I,Y
40 NEXT
```

Program ćemo pokrenuti naredbom RUN

```
>
_
```

`RUN` [CR]

Na ekranu će se iscrtati funkcija sinus. Detaljniji opis korištenih BASIC naredbi slijedi kasnije. Dobivenu sliku i program možemo izbrisati pomoću [CTL]L, međutim, naš program se i dalje nalazi u memoriji računala. U to ćemo se uvjeriti koristeći komandu LIST.

`LIST` [CR]

Prije uneseni program (bez slike) bit će ispisan na ekranu. Za one koji žele imati nacrtanu sinusnu funkciju više frekvencije, upišimo

```
LIST 20
```

pojavit će se

```
20 Y=100*SIN(I/40)+128
```

Tipkom [↑] dovedemo kursor u liniju 20 i pritisnemo [PF4] (kopi tipku), što će rezultirati pomicanjem kursora u desno s istim učinkom, kao da smo posebno pritisnuli tipku slovišta. Pomoću kopi tipke pomičemo kursor dok ne dođemo do broja 40, koji zamijenimo sa 10, a ostatak linije iskopiramo sa [PF4]. Poslije LIST 20 na ekranu se mora pojaviti:

```
20 Y=100*SIN(I/10)+128
```

Poslije RUN dobit ćemo sliku sinusne funkcije više frekvencije.

Prilikom pisanja programa, moguće je unijeti neki pogrešan znak,npr.:

```
PRINT"ORAOR"
```

Vidimo da je jedno slovo R suvišno. Problem rješavamo tipkom za pomicanje kursora u lijevo [←] ili tzv. delit (DELETE) tipkom. Ispod svakog suvišnog znaka dovedemo kursor (u našem slučaju ispod suvišnog R), i znak u BASIC instrukciji više ne postoji.

```
PRINT"ORAOR"
 _
```

Na mjestu bivšeg "R" upišemo navodnik i ispravka je gotova.

Dakle, koristeći tipke delit (DELETE), kopi (COPY=[PF4]), te tipke za pomicanje kursora, moguće je popravljati tekst na ekranu,, dok ne dobijemo željeni oblik. Opisana mogućnost popravljanja (editiranja) teksta na ekranu od izuzetne je važnosti, jer omogućava korisniku da uz jednostavne zahvate popravlja ili mijenja sadržaj bilo koje instrukcijske linije. Prilikom korištenja grafičkih mogućnosti na mikroračunalu "ORAO", može se dogoditi slučaj miješanja teksta i grafike. Tako dobivenu "mješavinu" kopi tipka ne prepoznaje, pa na tome mjestu ispiše "space" (prazno mjesto).

1.4 RAD S KAZETOFONOM

Za trajno pohranjivanje nekog programa koristit ćemo kazetofon (vanjska memorija), naravno uz pretpostavku, da je povezanost kazetofona sa računalom pravilno izvedena. Korištenjem kazetofona riješit ćemo problem "skladištenja" velikog broja različitih programa koje ćemo kasnije lako pozvati i ponovno koristiti. Potrebno je naglasiti, da korištene trake moraju bez bilo kakvih oštećenja, jer će svaka greška na traci biti uzrokom kasnijih teškoća u radu s pozivanjem prije pospremljenih programa.

Na slici 1.3 je prikazano kako se kazetofon priključuje na mikroračunalo ORAO".

1. na ulaz kazetofona (MIC)
2. masa
3. na izlaz kazetofona (PHONO ili EAR

Slika 1.3

1.4.1 Snimanje programa na traku (SAVE)

Poslije pravilnog povezivanja računala sa kazetofonom, možemo početi sa probnim prijenosom podataka iz računala na traku. Ukoliko se program za crtanje sinusa još uvijek nalazi u računalu, iskoristit ćemo ga za "probni rad", prilikom kojega ćemo provjeriti pravilnost spajanja i veličinu nivoa koji je potreban za korektno snimanje i reprodukciju.

Unesimo:

```
SAVE"SINUS" [CR]
SNIMANJE ?_ [CR]
```

Poslije prvog pritiska na [CR] pojavit će se poruka SNIMANJE ?, koja obavještava korisnika neka uključi kazetofon na SNIMANJE (RECORD TAPE). Slijedeći [CR] prouzrokuje zvuk iz mikroračunala koji označava da je unos podataka iz računala na traku u toku. Istovremeno se kursor privremeno gubi i pojavljuje se tek kada je prijenos podataka završen. Sada smo naš program trajno pohranili i moći ćemo ga pozvati s trake kada to zaželimo u potpuno pravilnom obliku.

1.4.2 Učitavanje programa s trake (LOAD)

Proces pozivanja programa s kazete izvodi se na slijedeći način:

```
LOAD"SINUS" [CR]
REPRODUKCIJA ? [CR]     ,
```

Naredba LOAD omogućava čitanje programa iz kazetofona u memoriju računala. Poslije te naredbe moramo upisati ime programa pod navodnim znakovima. To je vrlo važno, jer će tijekom vremena biti na traku snimljeno mnogo različitih programa pod različitim imenima. Dakle, upisano ime programa kojega želimo učitati, omogućava računalu da nađe upravo onaj program koji želimo. Poslije prvog [CR] pojavit će se poruka REPRODUKCIJA ? (PLAY TAPE), koja nam govori da moramo pokrenuti kazetofon. Prilikom traženja željenog programa računalo će nailaziti na imena nekih drugih programa i njihova imena ispisivati na ekranu, ali ih neće učitavati. Drugim pritiskom na [CR] gubi se kursor i računalo počinje tražiti zadani program. Pojava prompta i kursora označava, da je program učitan i spreman za korištenje. Dovoljno je upisati LIST i program će biti ispisan na ekranu, spreman za izvođenje.
Zaustavljanje procesa "loadanja" moguće je postići pritiskom na tipku [CTL].

1.4.3 Katalogiziranje trake (LOADC)

Tijekom vremena na traku ćemo pospremiti mnogo programa s različitim imenima. Da bi imali pregled o kojim programima se radi, koristimo naredbu LOADC:

```
LOADC [CR]
REPRODUKCIJA ? [CR]

TEST                  0400 07FE 03FE
SLIKA                 6000 7FFH 1FFF
FUNKCIJA SINUS        0400 082A 042A
PETLJA. ASS           1000 102A 002A
POTPROGRAM.ASS        1100 1234 0134
POKUS.BAS             0400 148A 108A
SLIKA.DAT             6000 8000 2000
[CTL]

>
_
```

Ovdje se vidi primjer kataloga trake. U slučaju da želimo zaustaviti daljnje katalogiziranje, dovoljno je pritisnuti tipku [CTL] koja nas vraća u početno stanje računala.
Upotrebljavajući opisane mogućnosti rada s kazetofonom, korisnici mogu vrlo lako i brzo rukovati sa mnogo različitih programa. Dodatne mogućnosti korištenja trake bit će naknadno opisane.

1.5 POVEZIVANJE MIKRORAČUNALA SA ŠTAMPAČEM

Na mikroračunalu "ORAO" predviđen je rad sa štampačem (printerom). Na slici 1.4 prikazan je način spajanja štampača s mikroračunalom.

1.usklađivanje (handshake)
2.ne spaja se
3.masa
4.ne spaja se
5.vrući kraj printera (linija podataka)

Slika 1.4

1.6 NAČIN KORIŠTENJA ŠTAMPAČA

Mikroračunalo "ORAO" je oblikovano tako, da može raditi sa štampačem serijskog prijenosa podataka (riječ je o serijskom standardu tzv, RS 232 ili V24). Zbog toga je važno poznavati na koju je brzinu prijenosa podešen priključeni štampač. Uobičajene brzine nogu biti:

300 [bauda] (data frame 8 + 2)
600 [bauda]
1200 [bauda]
2400 [bauda]
4800 [bauda] itd.

Mikroračunalo "ORAO" može raditi s brzinama od 300 do 2400 [bauda]. Određena brzina odabire se programski,

P0 300 [bauda]
P1 600 [bauda]
P2 1200 [bauda]
P2 2400 [bauda]

Nepravilno unesen podatak o brzini, npr. P6, generirat će:

```
*NEPRAVILAN UNOS
```

1.7 PRIMJER PROGRAMIRANJA U STROJNOM JEZIKU

Programiranje a strojnom jeziku izvodivo je pomoću miniassemblera, koji se nalazi u dijelu programske podrške (SOFTWARE) pod imenom MONITOR. Mogućnost rada u MONITORU dobivamo odmah poslije uključivanja

mikroračunala. Ponekad se može dogoditi, da se poslije uključenja računala ne nalazimo u MONITORU. U tom slučaju dovoljno je pritisnuti tipku RESET (Slika 1.1-6) i računalo će ispisati:

```
                                        * * * O R A O * * *
*_
```

uključujući i žmigajući kursor. Sada je sve spremno za unos našeg jednostavnog programa u strojnom kodu, koji će ispisati 16 uzastopnih znakova A. Unesimo

```
*A1000 [CR]
1000 LDA #0A
1002 JSR FFF1
1005 LDX #10
1007 LDA #41
1009 JSR FFF1
100C DEX
100D BNE 1007
100F RTS
1010Q [CR]
```

Unošenjem znaka Q izlazimo iz miniassemblera ponovo u MONITOR. Međutim, naš strojni program se nalazi u memoriji računala. Da bi se uvjerili u točnost unosa koristimo se monitorskom naredbom X (disassembler), koja strojni kod pretvara u lakše razumljiv mnemonički
oblik.

```
*X 1000 100F [CR]
1000 A9 CA            LDA #0A
1002 20 F1 FF  JSR FFF1
1005 A2 10            LDX #10
1007 A9 41            LDA #41
1009 20 F1 FF  JSR FFF1
100C CA                       DEX
100F 60                       RTS
```

Lijeva strana disassemblerskog listinga prikazuje strojni kod, kojeg ustvari mikroprocesor 6502 jedino razumije. Ostaje nam još samo da prijašnji strojni program izvršimo:

```
*U1000 [CR]
AAAAAAAAAAAAAAAA
*_
```

Poslije ovog uvodnog dijela, koji je prije svega namijenjen lakšem savladavanju osnovnih problema rada na računalu, preći ćemo na rješavanje kompleksnijih problema,, koristeći pri tom dvije izuzetne prednosti računala:

- brzinu
- točnost

Možda će prvi koraci u radu s mikroračunalom "ORAO" (ili nekim drugim) biti malo teži, međutim, svakog dana naučit ćemo nešto novo, ulazeći tako u jedan izuzetan svijet, svijet računala.

POGLAVLJE 2

2.1 OPĆE KARAKTERISTIKE

Mikroračunalo ORAO, osim strojnog koda za mikroprocesor 6502, razumije i program napisan u BASIC-u.
BASIC spada u grupu viših programskih jezika, tj. za njegovo razumijevanje nije potrebno detaljno poznavanje hardware-ske strukture računala, kao ni znanje programiranja u strojnom jeziku.

Karakteristike:

- aritmetika kliznog zareza
- točnost na 6 znamenki
- opseg između 10E-38 i 10E38
- 72 znaka u jednoj liniji
- Dužina stringova maksimalno 255 znakova
- Pohranjivanje korisničkih programa na audio kazetu
- Mogućnost pozivanja strojnih programa iz BASIC-a
- Grafika 256*256
- Broj programske linije između 1 i/ili 63999

2.1.1. Računanje u BASIC-u

Vrlo često je potrebno saznati rezultat neke računske operacije, a da pri tom ne pišemo program u BASIC-u nego direktno unesemo željenu računsku operaciju ili izraz.

Primjer 1: Želimo izračunati vrijednost izraza

(7+11)*3.5

Upotrebom naredbe PRINT direktno u liniji napišemo:

```
PRINT (7+11)*3.5
```

Nakon pritiska na tipku [CR] na ekranu imamo:

```
63
>
```

Primjer 2: Želimo u jednoj liniji izračunati više izraza odjednom

```
PRINT 3*3,3.14159/2,7*7.2+4/5,SQR(2)
```

U liniji imamo četiri različita izraza odvojena zarezom. Nakon pritiska na tipku [CR], na ekranu će se pojaviti četiri rezultata:

```
9       1.5708  51.2            1.41421
>
```

Zbog neformatiranog ispisa rezultati nisu pregledno smješteni na ekranu. O formatiranju ispisa bit će govora u odjeljku 2.7.2.

```
3*3=9                   ;množenje
3.14159/2=1             ;dijeljenje
7*7.2+4/5=51.2          ;aritmetički izraz
SQR(2)=1.41421          ;korjenovanje
```

Primjer 3: Ispisivanje rezultata i teksta

```
PRINT"REZULTAT=",3^2 [CR]
REZULTAT= 9
>
```

što je rezultat operacije kvadriranja broja 3.

U primjeru 3 smo pokazali da je moguće naredbom PRINT ispisati uz rezultat željene operacije i popratni tekst. Tekst koji u BASIC-u napišemo pod navodnicima ima posebno ime STRING.

Primjer 4:Želimo na ekranu imati ispisan izraz i rezultat tog izraza

22*3=66

```
PRINT"22*3=",22*3 [CR]
 22*3= 66
>
```

U ovom primjeru pod navodnicima imamo isključivo tekst, a iza zareza tek dolazi izraz čiji rezultat tražimo.

Kao zaključak ovog odjeljka upamtimo slijedeće:

- naredbu PRINT direktno koristimo za ispis rezultata računske operacije
- ukoliko imamo više izraza čije rezultate tražimo, izrazi moraju biti odvojeni zarezom
- sve što se u naredbi PRINT nalazi pod navodnicima ispiše se u nepromijenjenom obliku i zove se STRING
- na kraju unosa komande obavezno pritisnuti [CR]

2.1.2 Varijable

Numeričke varijable mogu biti tipa

XY

X - prvi znak obavezno iz ASCII seta (isključena mala slova te velika Č,C,Đ,Š,Ž)
Y - slovo ili broj ASCII seta

Za STRING varijable važe ista ograničenja uz dodatak znaka $ (dolar) na kraju.

Primjer 1:Unesite u liniji X=7 i pritisnite tipku [CR]

```
X=7 [CR]

>
```

Sada smo simboličkoj varijabli X priredili numeričku vrijednost 7. Da se u to uvjerimo Unesimo ponovo u liniji:

```
PRINT X [CR]
7 >
```

čime smo ispisali numerički iznos simboličke varijable X.

Operacija "=" nema u potpunosti isto značenje kao i znak jednakosti u algebarskim izrazima, već ona ovdje isključivo služi za to da varijabli na lijevoj strani priredi vrijednost varijable ili izraza desne strane.

Pokušajmo sada
`X=X+1` [CR] te nakon toga
`PRINT X` [CR]
`8`

što znači,da smo numeričku vrijednost simboličke varijable X uvećali za jedan i ponovo je priredili varijabli X .

Primjer 2: unesite PRINT "X" [CR]

```
X

>
```

Na ekranu se pojavilo X !?

ZAKLJUČAK: "X" je STRING varijabla, a X je numerička varijabla, kao što smo već rekli u odjeljku 2.1.1

Primjer 3:

Unesite

```
X$="X="
```

te nakon toga

```
PRINT X$,X
```

Na ekranu sada imamo:

```
X= 8
```

jer je posljednja vrijednost varijable X ostala 8.

Istim postupkom, za vježbu, izvedimo primjer 4 iz prethodnog odjeljka:

`X$="22*3="` [CR]
`X=22*3` [CR]
`PRINT X$,X` [CR]

2.1.3 Planiranje programa:

Prvi korak prilikom pisanja programa, koji će eventualno biti kodiran u BASIC-u ili ASSEMBLBR-u, je apstrakcija problema koji želimo riješiti.
Drugim riječima, problem treba tako pripremiti da ga računalo može obraditi nizom svojih osnovnih naredbi.
Apstrakcija problema obično se izvodi na dva načina:

- opisno
- dijagramom toka (flow chart)

Primjer 1: Načinite apstrakciju problema koju ćemo koristiti za ispis prvih 100 brojeva od 0-99.

OPISNO

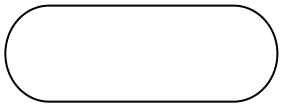
KORAK1: postavi varijablu N na nulu
KORAK2: ispiši N
KORAK3: uvećaj varijablu N za jedan
KORAK4: da li je varijabla N veća od 99?

- ako nije, idi na KORAK2:
- ako je, idi na KORAK5:

KORAK5: kraj (prekid izvođenja)

DIJAGRAM TOKA

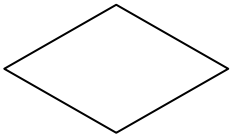
Dijagram toka nam omogućava grafičku apstrakciju željenog problema,a pri tom se koriste standardni simboli koji odgovaraju pojedinoj operaciji sl.1.2
Simbole povezujemo strelicama, čiji smjer označava tok pojedinih informacija iz bloka u blok.



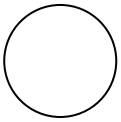
- definiranje početka i kraja problema (dolazi na početku i na kraju)

- ulazno / izlazne operacije

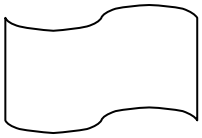
- definiranje procesa u problemu



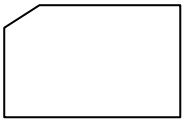
- simbol odluke



- veznik



- bušena traka

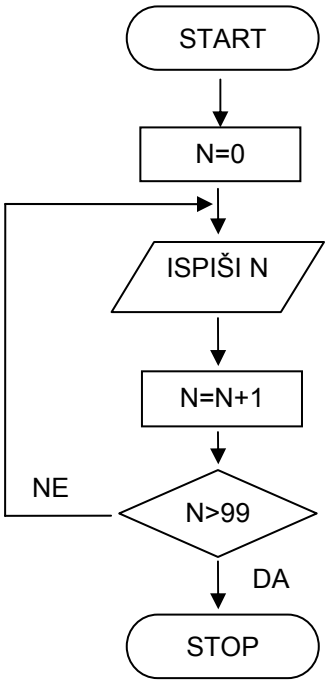


- bušena kartica

Slika 1.2

Grafički simboli za crtanje blok dijagrama

Nacrtajmo sada dijagram toka za problem ispisa 100 brojeva



Slika 2.2
Dijagram toka za ispis 100 brojeva

2.1.4 Pisanje izvornog programa u BASIC-u

Pisanje programa u jeziku koje računalo razumije najčešće se u literaturi naziva PROGRAMIRANJE, što nije pravilno, jer se pojam PROGRAMIRANJE odnosi isključivo na apstrakciju problema koji želimo riješiti računalom.
Problem svodimo na kreiranje ALGORITMA, tj. postupka ili ključa koji će nas najkraćim putem dovesti do željenog rješenja. Stoga pisanje programa u BASIC-u i/ili nekom drugom programskom jeziku nije programiranje već KODIRANJE programa koji je izražen u hipotetičkom obliku (opisno, dijagram toka itd.) .

Primjer 1: Kodirajmo u BASIC-u program iz prethodnog odjeljka za ispis 100 brojeva.

Program kodiran u BASIC-u sastoji se od programskih linija koje sadrže osnovne naredbe.
Svaka programska linija mora obavezno biti numerirana brojem između 1 i 63999, dok svaka slijedeća programska linija mora imati redni broj veći od prethodne. Program kodiran u BASIC-u MIKRORAČUNALO "ORAO" interpretira sekvencijalno od linije s manjim rednim brojem prema liniji s većim rednim brojem, ukoliko programska linija ne sadržava instrukcije grananja koje skreću izvođenje programa.

Prvo unesimo direktno u liniji (bez rednog broja):

`LIST` [CR]

Ukoliko se na ekranu pojavi znak > u slijedećoj liniji, znači da u tekstualnom prostoru računala nema nikakvog programa.
Ukoliko dođe do ispisivanja (listanja) teksta na ekran, pa tek nakon toga znak > tada ponovo direktno u liniji unesite

`NEW` [CR]

Čime ćete obrisati tekstualni prostor predviđen za BASIC programe i na taj način omogućiti pisanje novog programa.

Vratimo se konačno na naš primjer:

```
10 N=0
20 PRINT N
30 N=N+1
40 IF N>99 THEN END
50 GOTO 20
```

Napisani program će računalo početi izvoditi kad direktno u liniji napišemo:

`RUN` [CR]

što znači IZVRŠI,

Rješenje našeg problema je na ekranu, tj. imat ćemo slijedeći ispis:

```
0
1
2
3
.
.
.
98
99
```

2.1.5 Mijenjanje izvornog programa

Izvorni program je potrebno često mijenjati bilo zato, jer smo prilikom pisanja napravili grešku, tj. pogrešno unesli neku od naredbi, ili želimo isti program koristiti za rješenje još nekog srodnog problema, ali prethodno moramo dodati neke izmjene.
Kako ne bismo zbog toga ponovo pisali novi program, koristeći neke od tipki sa tastature, modificirat ćemo postojeći program.Takav postupak naziva se uređivanje programa na ekranu ("screen editing").
Koristimo tipke za pomak kursora: [←], [↑], [↓], [→] i tipku [PF4]. Smjer strelice na tipki označuje smjer pomaka kursora. Tipka [PF4] ima funkciju kopiranja teksta ispod kursora, Potpuni opis navedenih tipki je dan u odjeljku koji opisuje tastaturu.

Navedimo na kraju primjer korištenja "editorskih" tipki.

Primjer 1: Unesimo prvo program za crtanje sinusoide

```
10 PI=3.14 159
20 K=2*PI/256
30 FOR X=0 TO 255
40 Y=127+l 00* SIN(K*X)
50 PLOTX,Y
60 NBXT X
70 END
>
```

Program ponovo izvršimo tako da direktno u liniji unesemo

`RUN` [CR]

Na ekranu će se pojaviti jedna perioda sinusoide amplitude 100, a ujedno će se na ekranu nalaziti još i tekstualni dio programa.
Sada bismo željeli da na ekranu bude ista sinusoida, ali da nema tekstualnog djela, odnosno našeg programa. Drugim riječima, moramo u programu navesti prethodno brisanje ekrana, a tek onda crtanje.
To postižemo tako da između 20 i 30 programske linije umetnemo (insertiramo) naredbu za brisanje ekrana.

Unesimo:

`25 PRINT CHR$(12)` [CR] na kraju

i sada ponovo izvršimo program. Na ekranu sada imamo samo graf funkcije.
Sljedeća promjena koju želimo je da nam se nacrta ista funkcija, ali dvije periode, te amplitudom 50.
Unesimo prvo direktno (bez programa), u liniji [CTL] i [L] istovremeno čim obrišemo ekran te nakon toga

`LIST` [CR]

Za dvije periode moramo mijenjati sadržaj linije 20 tj. umjesto

```
20 K=2*PI/256
```

mora biti

```
20 K=2*PI/128
```

Postupak izmjena je slijedeći :

- tipkom [↑] pomaknemo kursor do linije 20
- tipkom [PF4] iskopiramo kompletan sadržaj linije do 256
- sada se kursor nalazi ispod brojke 2
- unesemo umjesto 256, 128 i pritisnemo na tipku [CR]
- za promjenu amplitude moramo u liniji 40 faktor 100 promijeniti u 50
- tipkom [↓] pomaknemo kursor do linije 40
- iskopirajmo kompletan sadržaj linije do 100
- kursor se sada nalazi ispod brojke 1
- umjesto 1 Unesimo [SP] (razmaknica)
- umjesto 0 Unesimo 5
- tipkom [PF4] iskopiramo sve do kraja linije i pritisnemo tipku [CR]

Sada ponovo obrišemo ekran [CTL] i [L] istovremeno, te nakon toga

`LIST` [CR]

Opažamo da naš program sadrži sve izmjene.
Nakon izvršenja programa na ekranu imamo novi graf funkcije sa traženim izmjenama.

2.2 ULAZNO/IZLAZNE naredbe

ULAZNO/IZLAZNE naredbe služe za kontrolu toka podataka između mikroračunala ORAO i ulazno/izlaznih jedinica.
U osnovnom obliku mikroračunalo ORAO ima za ulaznu jedinicu tastaturu, a kao izlaznu jedinicu prikaz na TV ekranu ili TV monitoru.

2.2.1 INPUT naredba

INPUT naredba omogućava unos podataka sa tastature pod programskom kontrolom.

Primjer 1:

```
10 INPUT X
```

znači unesi jednu numeričku varijablu i priredi je simboličkoj varijabli X

```
20 INPUT X$
```

znači unesi jednu string varijablu i priredi je simboličkoj varijabli X$

Nakon što je računalo interpretiralo naredbu INPUT, na ekranu se pojavi znak ? što znači da korisnik može unijeti podatak (na kraju obavezno pritisnuti tipku [CR]).
Ukoliko se prilikom upisa pojavila greška mikroračunalo odgovara sljedećim ispisom na ekranu:

```
? PONOVI UPIS
```

te u novu liniju postavlja znak ?.

Primjer 2 pokazuje istovremeni unos numeričkih i string varijabli.
Svaka varijabla u INPUT naredbi mora biti od druge odvojena zarezom.

Primjer 2:

```
10 INPUT X,X,Z,A$
```

višestruki unos podataka

```
20 INPUT"UNESI VRJEDNOST";X
```
ispis poruke prije unosa

2.2.2 DATA i READ naredbe

Naredbe DATA i READ se koriste uvijek zajedno, da bi se izvršilo prirеđivanje vrijednosti pojedinim varijablama (znak "=" služi za istu svrhu).
Naredba READ prireduje sekvencijalno varijablama iz polja argumenta vrijednosti specificirane u polju argumenta DATA naredbe.
Podaci u polju argumenta DATA naredbe moraju biti odvojeni zarezom.

Primjer 1:

```
10 DATA 1,2,3,4,5
20 READ A,B,C,D,E
```

ekvivalentan niz operacija je

```
10 A=1
20 B=2
30 C=3
40 D=4
50 E=5
```

Primjer 2:

```
10 DATA 1,2,ORAO,MIKRORAČUNALO,7
20 READ A,B,C$,D$,C
```

ekvivalentan niz operacija je

```
10 A=1
20 S=2
30 C$="ORAO"
40 D$="MIKRORAĆUNALO"
50 C=7
```

2.2.3 RESTORE naredba

Vraća podatkovnu kazaljku na prvi argument DATA naredbe

Primjer 1:

```
10 DATA 1,2,3,4,5,6
20 READ A,B,C
30 RESTORE
40 READ D,E,F
50 RESTORE
60 READ G,H
```

Ekvivalentan niz operacija je

```
10 A=1
20 B=2
30 C=3
40 D=1
50 E=2
60 F=3
70 G=1
80 H=2
```

2.2.4 PRINT naredba

PRINT naredba omogućava korisniku da pod kontrolom BASIC programa i/ili direktno u liniji ispiše vrijednost željene varijable na ekran.
O naredbi PRINT je već bilo govora u odjeljcima 2.1.1 i 2.1.2, pa sada više nećemo navoditi primjere.
Kao specifičnost ovog BASIC-a navedimo da umjesto naredbe PRINT možemo koristiti znak ?.

Primjer 1:

```
10?"UPiTNIK ZAMJENJUJE PRINT NAREDBU"
```

```
>
RUN [CR]

UPITNIK ZAMJENJUJE NAREDBU PRINT
> LIST [CR]

10 PRINT"UPITNIK ZAMJENJUJE NAREDBU PRINT"
```

Primjer 2: Razmotrimo još slijedeća dva programa

```
10?"MIKRORAĆUNALO";
20?"ORAO"
```

odnosno

```
10?"MIKRORAĆUNALO"
20?"ORAO"
```

Nakon izvođenja prvog primjera imamo na ekranu ispis:

```
MIKRORAČUNALO ORAO
```

a nakon drugog

```
MIKRORAČUNALO
ORAO
```

Znak ";" na kraju PRINT naredbe nam nastavlja slijedeći ispis u istoj liniji, dok PRINT bez ";" na kraju automatski generira ispis u novom retku.

2.3 PETLJE I POTPROGRAMI

Primjer petlje smo upoznali već u odjeljku 2.1.3, tj. 100 puta smo ponavljali operaciju /ISPIŠI N/, /UVEĆAJ N ZA JEDAN/, te testirali konačni uvjet /DA LI JE N>99/.
U primjeru 1 iz odjeljka 2.1.4 problem iste petlje kodirali smo u BASIC-u.
BASIC mikroračunala ORAO omogućava kodiranje petlje uz automatsko ispitivanje konačnog uvjeta.

2.3.1 FOR...NEXT naredbe

Naredbe FOR i NEXT koriste se zajedno da oi se osnovala programska petlja.
Petlja omogućava izvršenje jedne ili više naredbi određeni broj puta.

FOR naredba sastoji se iz tri dijela: FOR,TO i STEP

- FOR dio inicijalno postavlja početni iznos varijable s kojim će se petlja početi izvoditi
- TO dio specificira konačni iznos varijable s kojim će petlja biti završena
- STEP dio definira korak povećavanja varijable prilikom opetovanja (iteracije)

Sve naredbe koje se nalaze unutar petlje izvršavaju se toliko puta kolike i petlja.
Ako ne definiramo vrijednost za STEP, korak iteracije će biti automatski jedan.
NEXT naredba dolazi na kraj petlje i njome se petlja zatvara.
Dozvoljena je upotreba petlje unutar petlje, ali se one ne smiju preklapati (Primjer 1).

Petlje koje završavaju na istom mjestu mogu biti zatvorene jednom NEXT naredbom npr.

```
NEXT I,J
```

Primjer 1: Ispis brojeva od 0 do 99

```
10 FOR N=0 TO 99
20 PRINT N
30 NEXT N
```

Primjer 2: Nedozvoljena upotreba FOR...NEXT petlje:

```
100 FOR X=7 TO 20
200 PRINT X
210 FOR Y=0.25 TO 2.5
220 NEXT X
230 PRINT Y
240 NEXT Y
```

Primjer 3: Petlje koje završavaju na istom mjestu:

```
10 FOR I=0 TO 10 STEP 2
20 FOR J=0 TO 20 STEP 4
30 PRINT I,J
40 NEXT J,I
```

liniju 40 možemo još napisati:

```
40 NEXT J:NEXT I
```

ili samo

```
40 NEXT:NEXT
```

Primjer 4: Početni iznos varijable, konačni iznos varijable i korak mogu, također, biti varijable čiji će iznos biti izračunat unutar programa ili definiran naredbom INPUT:

```
10 INPUT X, Y,Z
20 FOR I=X TO Y STEP Z
30 PRINT 1, 1^2
40 NEXT

RUN

?10,100,2
10      100
12      144
.       .
.       .
.       .
100 10000
```

Primjer 5: Izračunavanje faktorijela

```
10      INPUT"N=";N
15      FA=1
20      IF N=0 THEN 60
30      FOR X=N TO 1 STEP -1
40      FA=FA*X
50      NEXT X
60      PRINT"N!=";FA
70      END

RUN


N = ? 3 [CR]
N!= 6
>
```

2.3.2 GOTO naredba

Naredba kojom se izvođenje programa bezuvjetno skreće na specificiranu liniju .
Nakon toga program se dalje ponovo izvodi sekvencijalno od linije na koju je skrenut GOTO naredbom.

Primjer 1:

```
100?"mikroračunalo";
200 GOTO 300
250 A=2:B=3:C=A+B
300?"ORAO"
310?"PEL VARAŽDIN"
320?"LINIJA 250 NEĆE BITI IZVRŠENA"
RUN
```

```
mikroračunalo ORAO
PEL VARAŽDIN
LINIJA 250 NEĆE BITI IZVRŠENA
```

2.3.3 GOSUB i RETURN naredbe

Unutar programa često se pojavljuje zahtjev za izvršenje istog zadatka u različitim dijelovima programa.
Svaki takav zadatak tada definiramo kao neovisnu cjelinu programa koja može biti testirana i izvođena odvojeno od tzv. GLAVNOG DJELA, a nazivamo je SUBRUTINA ili POTPROGRAM.
Svaki POTPROGRAM mora biti zaključen naredbom RETURN, kako bi se kontrola izvođenja ponovo vratila GLAVNOM DIJELU.

Primjer korištenja GOSUB...RETURN naredbi je opisan u odjeljku 2.3.4

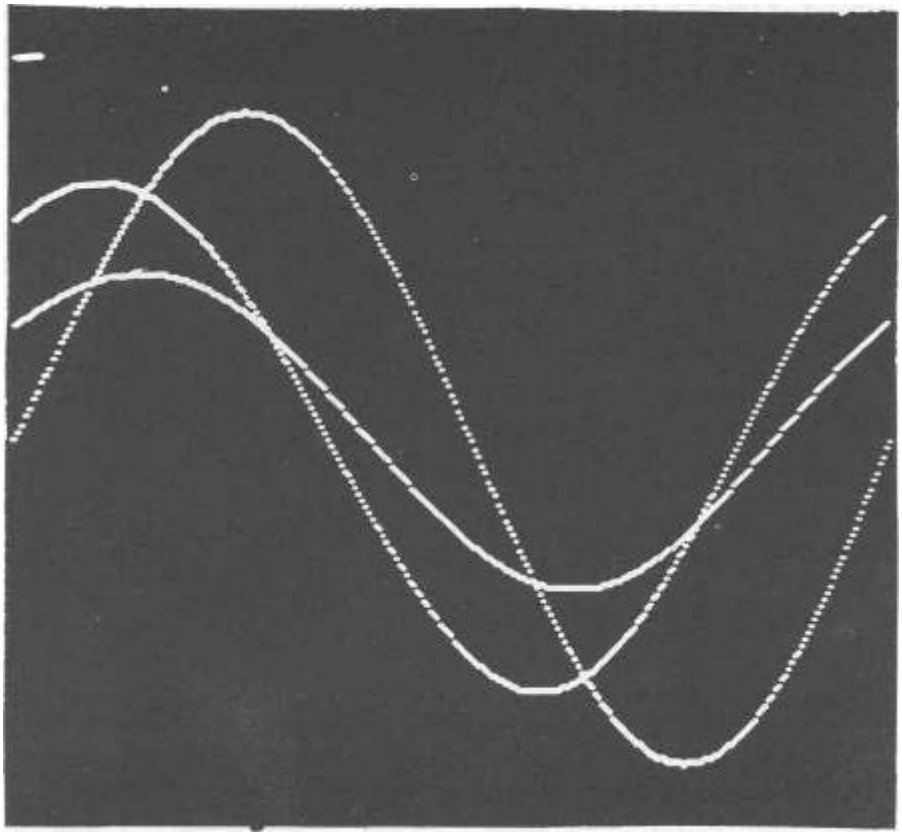
2.3.4 REM naredba

REM naredba omogućava pisanje korisnikovog komentara u program. Preporučuje se što češće korištenje naredbe REM unutar programa, zbog veće preglednosti i razumljivosti programa.

Primjer 1: Korištenje potprograma

Želimo nacrtati tri sinus funkcije:
1. amplituda 100 bez faznog pomaka s obzirom na ishodište
2. amplituta 50 i fazni pomak pi/4 radijana
3. amplituda 80 i fazni pomak pi/3 radijana

```
10 REM*
20 REM* GLAVNO TJELO PROGRAMA
30 REM*
35 REM
40 PI=3.141593
50 K=2*PI/256
60 FOR X=0 TO 255
65 REM
66 REM prva funkcija
67 REM
80 A=100:FI=0:GOSUB 1000
90 REM
92 REM druga funkcija
95 REM
100 A=50:FI=PI/4:GOSUB 1000
130 REM
135 REM treča funkcija
138 REM
140 A=80:FI=PI/3:GOSUB 1000
150 NEXT X
160 END:REM* KRAJ GLAVNOG TJELA
1000 REM *
1100 REM * potprogram za računanje
1200 REM * i crtanje točke
1300 REM *
1400 Y=128+A*SIN(K*X+FI)
1500 PLOTX,Y
1600 RETURN:REM* kraj potprograma
RUN
```



Slika 3.2

Kao zaključak navedimo dvije važne značajke SUBRUTINA:

1. Dijeljenje programa u module koji mogu biti testirani i izvođeni odvojeno od glavnog dijela, čime se povećava razumljivost i preglednost programa
2. Omogućavaju da se isti dio programa koristi za slične i/ili srodne probleme, a da se za svaki problem ne piše poseban program

2.3.5 ON naredba

Koristi se u kombinaciji sa naredbama GOTO i GOSUB, čime se omogućava razgranjivanje programa iz jednog mjesta na više.

Sintaksa ON naredbe je:

```
ON I GOTO 100,200,300
```

ako je 1=1 izvodi se GOTO100
ako je 1=2 izvodi se GOTO200
ako je 1=3 izvodi se GOTO300

Primjer 1: Upotreba ON naredbe

```
10 INPUT L
20 ON I GOTO 50,60,70
30 END
50 PRINT"PRVI SKOK":END
60 PRINT"DRUGI SKOK":END
70 PRINT"TREĆI SKOK":END
```

Ako je I veći od broja navedenih skokova, izvršava se prva slijedeća linija nakon ON...GOTO/GOSUB naredbe. Ako I ima negativnu vrijednost, korisnik će primiti od računala poruku da se pojavila greška.

2.3.6 IF...THEN naredba

Naredba IF se koristi da bi se kontrola izvršavanja programa prebacila na drugi dio iz glavnog toka, ako je ispunjen određeni uvjet.

Naredba IF se koristi u kombinaciji sa slijedećim operatorima:

1. = ; jednako
2. <> ; različito
3. < ; manje
4. > ; veće
5. <= ; manje ili jednako
6. >= ; veće ili jednako

Navedene operatore nazivamo operatori relacije ili odnosa. Sintaksa naredba IF je slijedeća:

```
IF xxx THEN yyy
```

Ukoliko je operacija ili izraz xxx istinit, tada se izvršava naredba yyy, a ukoliko je neistinit, tada se izvršava naredba u prvoj slijedećoj liniji nakon linije s IF naredbom.

Primjer 1: Upotreba IF naredbe

```
10 INPUT A,B
20 IF A>B+4 THEN 50
30 IF A=3 THEN PRINT"TROJKA":END
40 IF A>B THEN PRINTA,B:END
50 PRINT(B-4)
```

Primjer 2: Pogađanje brojeva

```
10 PRINT CHR$(12)
20 PRINT" POGAĐANJE BROJEVA "
40 Y=INT(100*RND(7))
50 FOR 1=1 TO 10
60 INPUT X
70 IF X<Y THEN PRINT"PREMALI JE":GOTO100
80 IF X>Y THEN PRINT"PREVELIKI JE":GOTO100
83 PRINT
85 PRINT"BRAVO POGODILI STE U";I;"-om POKUŠAJU":GOTO190
100 NEXT I
120 PRINT:PRINT"NAŽALOST OVAJ PUT NISTE USPJELI"
190 PRINT
200 INPUT"ŽELITE LI JOŠ (DA/NE)";A$
220 IF A$="DA" THEN 10
230 END
```

2.4 POLJA PODATAKA

Polje podataka u BASIC-u za mikroračunalo ORAO definiramo na slijedeći način:
X(I) ; jednodimenzionalno polje
X(I,J) ; dvodimenzionalno polje
X(I,J,K) ; trodimenzionalno polje

gdje su I,J,K indeksi polja, koji terminiraju određeni element polja za I,J,K=0,I ... (n-1), npr.:
X(3) predstavlja 4-ti element polja
X(I,2) predstavlja element dvodimenzionalnog polja koji je smješten u prvom retku odn. drugom stupcu (vidi Primjer 1 ovog odjeljka)

Opažamo da svako polje ima svoju dimenziju, kojom je ujedno i definiran broj elemenata u polju.

Npr. X(3,4,2) je polje dimenzije 3 koje ima ukupno: (3+1)x(4+1)x(2+1)=60 elemenata.

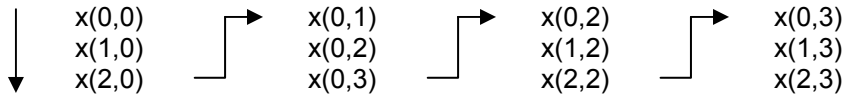
POLJE DAKLE PREDSTAVLJA SKUP INDEKSIRANIH VARIJABLI KOJE NAZIVAMO ELEMENTI POLJA.
ELEMENT polja je predstavljen imenom polja i jednim ili više indeksa u zagradi.
Elementi polja mogu biti numeričke i/ili string varijable, što specificiramo za potonje znakom $ u imenu npr.:

X(i) ; i-ti element polja numeričkih varijabli
X$(i) ; i-ti element polja string varijabli

Primjer 1:
a. Formiranje jednodimenzionalnog polja

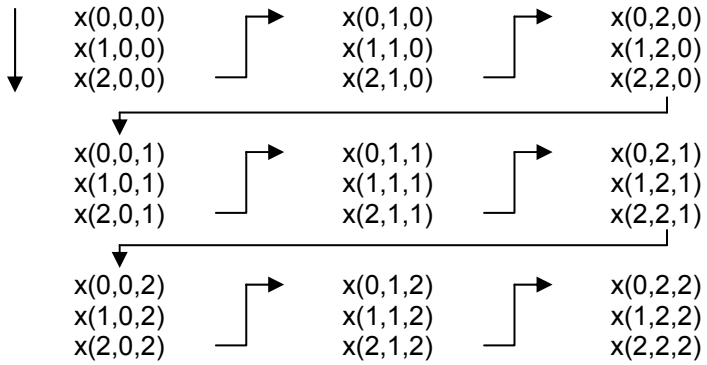
x(i) za i=0,1...9
x(0)
x(1)
x(2)
.
.
.
x(8)
x(9)

b. Formiranje dvodimenzionalnog polja x(i,j) za i=0,1,2 te j=0,1,2,3



Vidimo da dvodimenzionalno polje formiramo prema "brže" rastućem indeksu "i" i "sporije" rastućem indeksu "j".

c. Formiranje trodimenzionalnog polja x(i,j,k) za i=0,1,2 ; j=0,1,2 ; k=0,1,2



2.4.1 DIM naredba

DIM naredbom deklariramo dimenziju polja,odnosno u memoriji rezerviramo prostor za željeno polje, te ujedno sve elemente polja postavimo na nulu.

Primjer 1:

```
10 DIM X(10),X2(23,12),X4$(5)
```

rezerviramo (10+1) elemenata za polje X,(23+1)x(12+1) za polje X2 i (5+1) elemenata za polje X4$

Vratimo se sada na primjer 1 iz odjeljka 2.2.2 Primjer 2:

```
10 DATA 1,2,3,4,5
20 READ A,B,C,D,E
```

umjesto da koristimo pet numeričkih varijabli, formirajmo jednodimenzionalno polje od 5 elemenata i priredimo elementima polja vrijednosti iz polja argumenta DATA naredbe:

```
10 DIM X(4):REM rezerviramo mjesto za 5 elemenata
20 FOR 1=0 TO 4:REM formiramo indeks pomoću petlje
30 READ X(I):REM upisivanje podataka u polje
40 NEXT I
50 DATA 1,2,3,4,5
```

Da bi provjerili pravilnost prireñivanja vrijednosti pojedinim elementima, izvedimo slijedeće:
direktno u liniji napišimo

```
PRINT X(0),X(1),X(2),X(3),X(4)
```
[CR]

dobivamo slijedeći ispis

```
1 2 3 4 5
```

što je potvrda pravilnosti prireĎivanja.

2.4.2 Pisanje podataka u polje

Na prijašnjem primjeru smo naredbama READ i DATA pokazali mogućnost upisivanja podataka u polje. Pokažimo još slijedećim primjerom upisivanje podataka u polja pomoću naraedbe INPUT, te matematičkom funkcijom sinus.

Primjer 1:

```
10 DIM X(5),Y(255),Z(5)
20 PI=3.141593:K=2*PI/256
30 FOR 1=0 TO 255
40 IF I]=6 THEN 90
50 READ X(I):INPUT Z(I)
90 Y(I)=100*SIN(K*I)+128
100 NEXT I
110 DATA 0,1,2,3,4,5
```

Navedenim primjerima smo pokazali kako pojedinom elementu polja priredimo numeričku vrijednost. Potreba za formiranjem polja pojavljuje se čim imamo skup srodnih vrijednosti koje želimo preparirati za daljnju obradu.

2.4.3 Čitanje podataka iz polja

Poljem formiran skup srodnih podataka možemo sada koristiti pri daljnjoj obradi:

- numeričkih analiza
- statističkih obrada itd.
-

Vidimo da se pojavljuje zahtjev, da elementima ulaznog polja priredimo elemente novog polja (izlazno polje),ili da na temelju podataka ulaznog polja odredimo značajke polja (srednja vrijednost elemenata, standardna devijacija, maksimalna vrijednost elemenata itd.) ili, što je često slučaj, odredimo sortiranje elemenata po nekom ključu. Već smo rekli da je element polja jednoznačno definiran:

- imenom polja
- indeksom

Drugim riječima, podatak iz polja pročitamo tako, da formiramo varijablu koja nosi ime polja, a indeksom specificiramo željeni element.

Primjer 1: Suma dvaju elemenata polja:

```
100X1=X(4)+X(7)
110PRINT X1
```

Primjer 2: Srednja vrijednost elemenata polja

Polazimo od matematičke definicije srednje vrijednosti

$$Xsr = (1/n)\sum_{i=0}^{n-1} Xi = \frac{1}{n}(X_0 + X_1 + ...X_{n-1}) = (1/n)*(X_0 + X_1 + ...X_{n-1})$$

U memoriji računala već imamo polje X(I) koje sadrži 100 elemenata.
Program koji izračunava srednju vrijednost je:

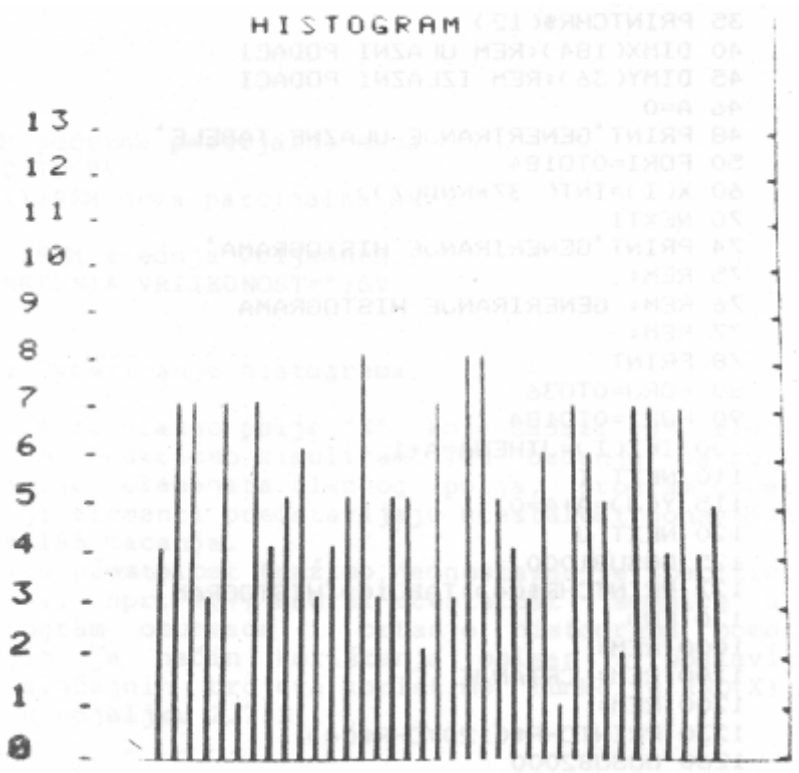
```
90 N=100
100 S=0:REM početna parcijalna suma
110 FOR 1=0 TO 99
120 S=S+X(I):REM nova parcijalna suma
130 NEXT I
140 SV=S/N :REM srednja vrijednost
150 PRINT"SREDNJA VRIJEDNOST=";SV
```

Primjer 3: Generiranje histograma

Program formira ulazno polje "X" koje sadrži 185 slučajnih brojeva između 0 i 36. Praktično simuliramo 185 "bacanja" na ruletu.
Na temelju elemenata ulaznog polja, program generira izlazno polje "Y", čiji elementi predstavljaju učestalost

ponavljanja pojedinih brojeva kroz 185 bacanja.
Broj, čiju učestalost tražimo jednostavno je specificiran indeksom izlaznog polja, npr. Y(7) sadrži učestalost "sedmice" u nizu od 185 bacanja. Program obuhvaća i crtanje histograma pomoću grafičkih naredbi, kojih je način korištenja opisan u poglavlju 2.11. Za generiranje slučajnih brojeva koristimo funkciju RND(X) o kojoj će biti govora u odjeljku 2.7.2 .

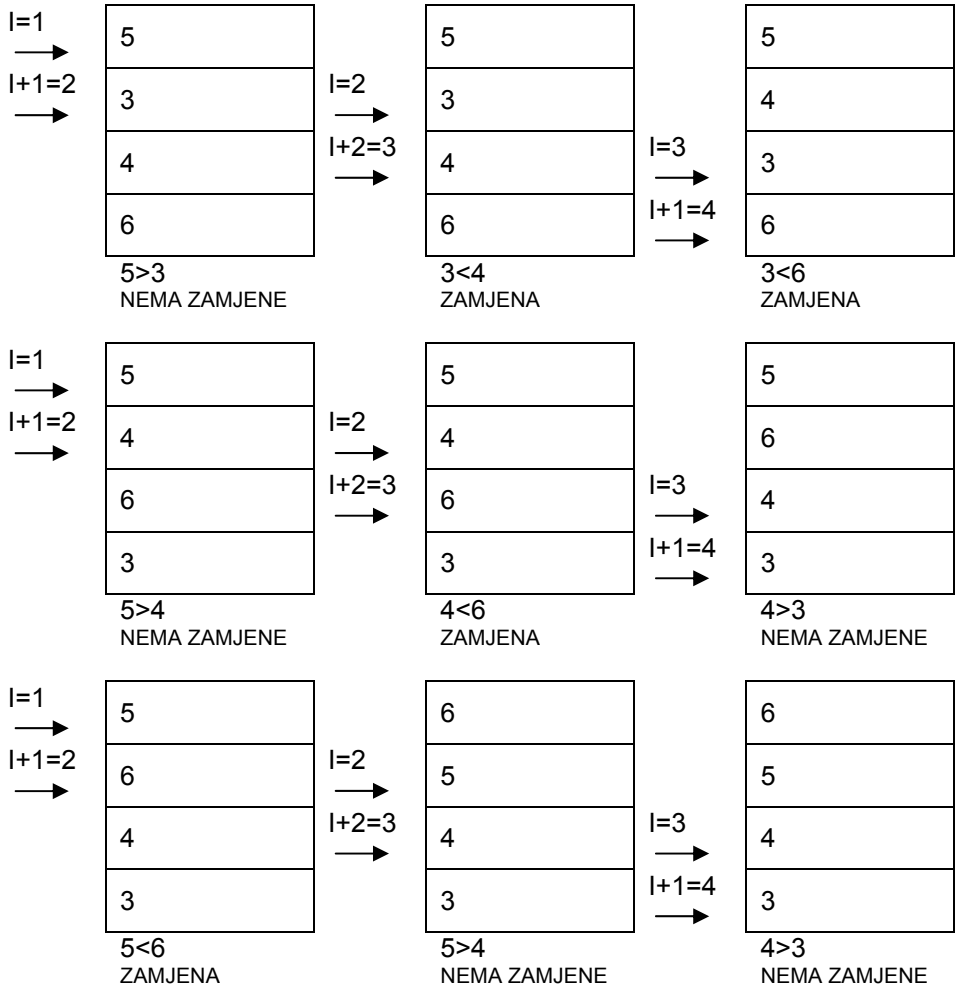
```
LIST

10 REM:
20 REM: HISTOGRAM
30 REM:
35 PRINTCHR$[(12)
40 DIMX(184):REM ULAZNI PODACI
45 DIMY(36):REM IZLAZNI PODACI
46 A=0
48 PRINT"GENERIRANJE ULAZNE TABELE*
50 F0RI=0TO184
60 X(I) = INT(37*RND(7))
70 NEXTI
74 PRINT"GENERIRANJE HISTOGRAMA"
75 REM:
76 REM: GENERIRANJE HISTOGRAMA
77 REM:
78 PRINT
80 F0RJ=0TO36
70 FORI=0TO184
100 IFX(I)=JTHENA=A+1
110 NEXT I
115 Y(J)=A:A=0
120 NEXT J
125 GOSUB1000
127 PRINTCHR$(4);TAB(10)"HIST0GRAM"
130 END
1000 REM:
1100 REM: CRTANJE
1200 REM:
1230 PRINTCHR$(12);CHR*(4)
1260 GOSUB2000
1300 FORI=0TO36
1400 MOV5*I+53,0
1500 DRAW5*I+53,Y(I)*16
1600 NEXT I
1700 RETURN
2000 REM:
2100 REM: SKALA
2200 REM:
2250 GOSUB3000
2300 FORY=0TO210STEP16
2400 MOV30,Y:DRAW32,Y
2450 MOW 253,Y:DRAW255,Y
2500 NEXT Y
2600 RETURN
3000 FORX=0TO31
3100 PRINT:NEXT
3200 FORX=0TO13
3300 PRINTX;CHR$(11);CHR$(11);CHR$(11)
3400 NEXT:RETURN

>
```


HISTOGRAM
13
12
11
10
9
8
7
6
5
4
3
2
1
0

Slika 4.2 HISTOGRAM

Primjer 4: Sortiranje brojeva po veličini

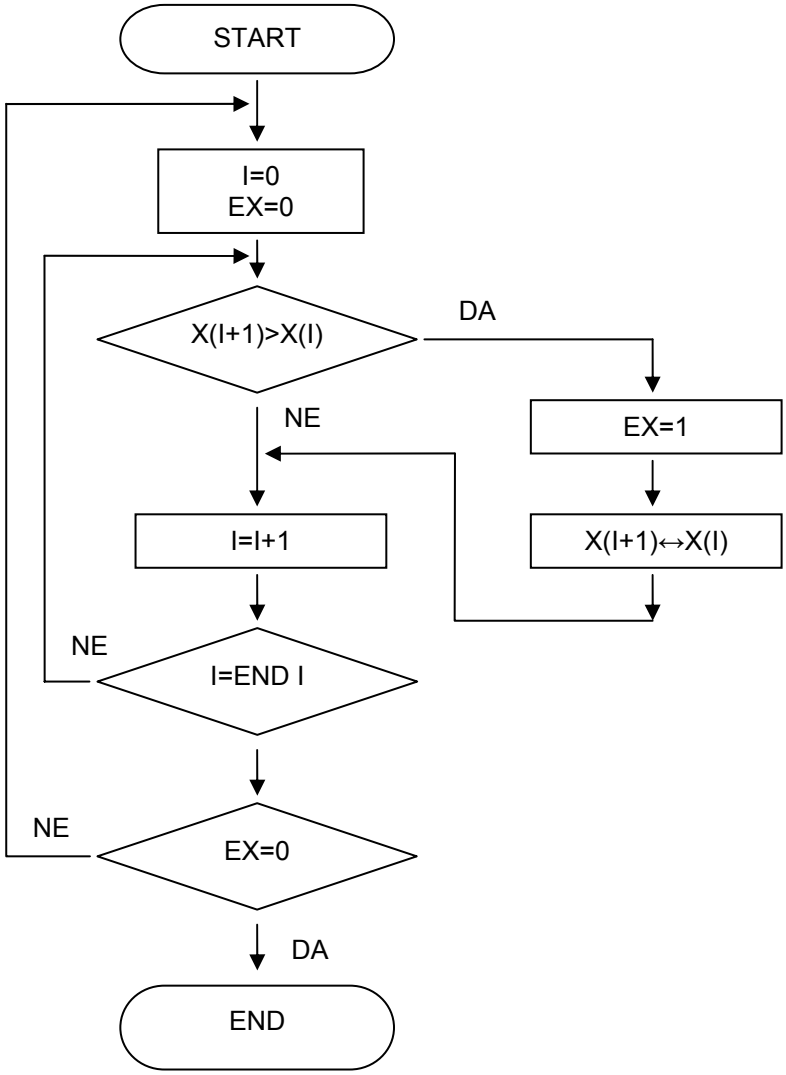
Algoritam, prema kojem se sortiranje provodi, u literaturi je poznat pod imenom "BUBBLE SORT".
Dijagram toka je prikazan na slici 6.2., dok je mehanizam sortiranja po navedenom algoritmu prikazan na slici 5.2..



Slika 5.2. BUBBLE SORT mehanizam

Vidimo daše princip sortiranja bazira na zamjeni mjesta broja sljedbenika s prethodnikom, ukoliko je sljedbenik veći. Drugim riječima, brojevi koji su u polju veći, kao "mjehurići" putuju prema vrhu tabele, zato je i takav način sortiranja dobio ime BUBBLE SORT, jer riječ "bubble" na engleskom znači mjehurić.
Opažamo da, osim indeksa I koji nam služi kao kazaljka elementa polja, imamo još jednu varijablu EX koja nam daje kriterij odluke.
Na početku sortiranja je EX=0 koji odmah pri prvoj zamjeni poprima vrijednost "1" i zadržava je sve dok kazaljka ne dostigne dno tabele. Ukoliko je EX=1, tada postupak ponavljamo tako dugo dok EX ne postane "0" i kazaljka istovremeno pokazuje na dno tabele.
Zamjenu dvaju elemenata u polju provedemo uz pomoć još jedne varijable T.

```
KORAK1: T=X(I-1)      ;X(I-1) spremamo na T
KORAK2: X(I-1)=X(I)   ;X(I) spremamo na X(I-1)
KORAK3: X(I)=T        ;T VRATIMO NA X(I)
KORAK4: kraj
```


START
I=0
EX=0
X(I+1)>X(I)
DA
NE
EX=1
I=I+1
X(I+1)↔X(I)
NE
I=END I
NE
EX=0
DA
END

Slika 6.2 Dijagram toka BUBBLE SORT-a

```
10 REM:
20 REM: BUBBLE SORT
30 REM:
40 PRINTCHR*(12)
50 T=0
60 INPUT"KOLIKO BROJEVA ŽELITE SORTIRATI";N
65 PRINT
68 DIMX(N)
70 REM:
72 REM: UNOS PODATAKA
74 REM:
76 FORI=0TON-1
78 PRINTI+1;:INPUTX(I)
79 NEXT I
80 REM:
90 REM: POČETAK SORTIRANJA
95 REM:
100 EX=0
110 FORI=0TON-1
120 IF X(I + 1)>X(I)THENGOSUB500
130 NEXT i
140 IFEX<>0THEN100
145 GOSUB600
150 END
500 REM:
510 REM: ZAMJENA
520 REM:
540 EX=1
550 T=X(I):X(I)=X(I+1):X(I+1)=T
560 RETURN
600 REM:
610 REM: POTPROGRAM ZA ISPIS
620 REM:
625 PRINT:PRINT
650 FORI=0TON-1
660 PRINTI+1;". ";X(I)
670 NEXT I
680 RETURN
>
RUN KOLIKO BROJEVA ŽELITE SORTIRATI? 8

1 ? 123.456
2 ? 3456,45
3 ? 23
4 ? 678.345
5 ? 12.9090
6 ? 453
7 ? 22234
8 ? .034

1 . 22234
2 . 3456.45
3 . 678.345
4 . 453
5 . 123.456
6 . 23
7 . 12.909
8 . .034
```

2.5 STRINGOVI

Pojam STRING-a smo već upoznali u odjeljku 2.1.1,tj. rekli smo da je STRING niz znakova koji mogu biti:

- slova
- brojevi
- znakovi interpunkcija

a nalaze se pod znakovima navodnika.
Stringovi jedino ne mogu biti kontrolni znakovi npr. "[CTL][L]","[CR]" itd.
String može biti uključen u kontekstu naredbi PRINT ili INPUT dok string varijable moraju obavezno na kraju sadržavati znak "$". Stringovi mogu, također, biti elementi polja i tada ime polja mora također sadržavati znak "$".
Znak "$", dakle, predstavlja identifikacijski znak za stringove, odnosno string operator.
Stringove možemo uspoređivati u smislu određivanja njihove jednakosti.

Primjer 1: Uspoređivanje stringova

```
10 A$ = "LIJEPO":B$ = "LIJEPO"
20 C$ = "RIJETKO":D$ = "ISTI SU"
30 IF A$=B$ THEN PRINT D$
40 IF A$=C$ THEN PRINT D$
50 PRINT"RAZLIČITI SU ! "
```

Primjer 2: Zbrajanje stringova

```
10 READ A$,B$,C$,D$,E$,F$,G$
20 S1$=A$+" "+B$+" "+F$+" "+G$
30 S2$=A$+" "+B$+" "+C$+" "+D$+" "+E$
40 PRINTS1$:PRINTS2$
50 END
90 DATA SVAKI,POČETAK,IMA,SVOJE
95 DATA ČARI,JE,TEŽAK
```

U slijedećim odjeljcima su opisane daljnje moguće operacije sa stringovima pomoću tzv. string funkcija.

2.5.1 ASC funkcija

Funkcijom ASC(X$) priredimo nekoj numeričkoj varijabli decimalnu vrijednost prvog znaka u stringu koja odgovara YU (ASCII) kodu.

Primjer 1:

```
10 D=ASC("?")
20 E=ASC("A")
30 X$="ORAO":X=ASC(X$)
40 PRINT D;E;X

RUN
63 65 79
```

Vrijednosti kodova za pojedine znakove dane su u tablici YU (ASCII) kodova na kraju priručnika.

2.5.2 CHR$ funkcija

Funkcija CHR$(X) pretvara decimalnu vrijednost YU (ASCII) koda u alfanumerički znak, odnosno tvori string od jednog karaktera.
CHR$(X) funkcija ja dakle inverzna funkcija od ASC(X$) funkcije.

Primjer 1:

```
10 PRINT CHR$(63);CHR$(65);CHR$(79)
RUN
? A O
```

2.5.3 LBFT$ funkcija

Funkcijom LEFT$(X$,N) tvorimo podstring od-stringa X$ dužine N znakova, počevši s lijeve strane.

Primjer 1:

```
10 X$="TELEVIZIJA"
20 X4$=LEFT$(X$,4)
30 PRINT X4$

RUN
TELE
>
```

2.5.4 RIGHT$ funkcija

Funkcijom RIGHT$(X$,N) tvorimo podstring od stringa X$ dužine N znakova počevši s desne strane.

Primjer 1:

```
10 X$="TELEVIZIJA"
20 X6$=RIGHT$(X$,6)
30 PRINT X6$

RUN
VIZIJA
>
```

2.5.5 MID$ funkcija

Funkcijom MID$(X$,I,J), tvorimo podstring funkciju od stringa X$ dugačak J znakova, počevši s I-tim znakom.

Primjer 1:

```
10 X$="TELEVIZIJA"
20 A$=MID$(X$,3,4)
30 PRINT A$

RUN
LEVI
>
```

2.5.6 LEN funkcija

Funkcijom LEN(X$) priredimo nekoj numeričkoj varijabli dužinu stringa X$.

Primjer 1;

```
10 A$="RAĆUNALO"
20 X=LEN(A$)
30 PRINT X
RUN

8
>
```

jer string "RAČUNALO" ima 8 znakova.

2.5.7 STR$ funkcija

Funkcijom STR$(X) pretvaramo numeričku vrijednost varijable X u string.

Primjer 1:

```
10 INPUT X: REM X je numerička varijabla
20 X$=STR$(X)
30 PRINT X$
RUN
? 25
25
```

2.5.8 VAL funkcija

Funkcijom VAL(X$) priredimo numeričkoj varijabli numeričku vrijednost string X$.

```
10 A$="3.141593"
20 A=VAL(A$)
30 PRINT A
RUN
3.141593
>
```

2.5.9 Primjeri programa sa STRING funkcijama

Primjer 1: Pretvorba binarnog broja u decimalni

```
LIST
10 REM:
```

```
20 REM: BINARNO-DECIMALNA PRETVORBA
30 REM:
40 INPUT "BINARNI BROJ=";B$
50 L=LEN(B$):D=0
60 FORI=1TOL
70 A=VAL(MID$(B$,I,1))
80 IFA>1THENPRINT"BROJ NIJE BINARNI":END
90 D=D+A*2^(L-I)
92 D=INT(D)
95 NEXT I
100 PRINT"DECIMALNI EKVIVALENT=";D
>
```

```
RUN
BINARNI BR0J=? 1111111111111111
DECIMALNI EKVIVALENT= 65535

>
RUN
BINARNI BR0J=? 1010
DECIMALNI EKVIVALENT= 10

>
RUN
BINARNI BROJ=? 01111111
DECIMALNI EKVIVALENT= 127
```

Primjer 2: Pretvorba decimalnog broja <=65535 u binarni ekvivalent

```
LIST
10 REM:
20 REM: DECIMALNO-BINARNA PRETVORBA
30 REM:
35 B$=""
40 INPUT"DECIMALNI BROJ=";D
50 X=INT(D/256):GOSUB80
60 X=D-256*X;GOSUB80
65 PRINT"BINARNI EKVIVALENT=":PRINT
68 PRINTB$
70 END
80 FORI=7TO0STEP-1
90 B=SGN(XAND2^I)
100 B$=B$+STR$(B)
120 NEXT I
130 RETURN
>

RUN
DECIMALNI BROJ=? 65535
BINARNI EKVIVALENT=
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

>
RUN
DECIMALNI BROJ=? 127
BINARNI EKVIVALENT=
0 0 0 0 0 0 0 0 1 1 1 1 1 1 1

>
RUN
DECIMALNI BR0J=? 40960
BINARNI EKVIVALENT=
1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0

>
RUN
DECIMALNI BR0J=? 10
BINARNI EKVIVALENT=
0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0
```

Primjer 3: Pretvorba heksadecimalnog broja <=FFFF u decimalni ekvivalent

```
LIST
10 REM:
11 REM: HEXABECIMALNO U DECIMALNO
12 REM:
15 INPUT"HEXABECIMALNI BROJ=";A$
20 A$=RIGHT$("0000"+A$,4):A=0
30 FORJ1=1TO4:C$=MID$(A$,J1,1)
40 IFC$>="A"ANDC$<="F"THENC=ASC(C$)-ASC("A")+10
50 IFC$>="0"ANDC$<=" 9" THENC=ASC(C$)-ASC("0")
60 A=A*16+C:NEXTJ1
70 PRINT"DECIMALNI EKVIVALENT=";A
>
RUN

HEXADECIMALNI BROJ=? FFFF
DECIMALNI EKVIVALENT= 65535

>
RUN
HEXADECIMALNI BROJ=? DD11
DECIMALNI EKVIVALENT= 56593
```

```
>
RUN
HEXADECIMALNI BROJ=? C274
DECIMALNI EKVIVALENT= 49780

>
RUN
HEXADECIMALNI BROJ=? 007F
DECIMALNI EKVIVALENT= 127

>
RUN
HEXADECIMALNI BROJ=? 03FF
DECIMALNI EKVIVALENT= 1023

>
```

Primjer 4: Pretvorba decimalnog broja <=65535 u heksadecimalni ekvivalent

```
LIST

10 REM:
20 REM: DECIMALNO U HEXADECIMALNO
30 REM*
40 INPUT"DECIMALNI BROJ=";Q
45 GOSUB50:PRINT"HEXADECIMALNI EKVIVALENT=";Q$
46 END
50 B=INT(Q/256):GOSUB90
60 H1$=H$
70 B=Q-B*256:GOSUB90
80 H2$=H$:Q$=H1$+H2$:RETURN
90 V=BAND15:H=(BAND240)/16
100 IFH>9THENH=H+55*GOTO120
110 H=H+48
120 IFV>9THENV=V+55;GOTO140
130 V=V+48
140 H$=CHR$(H)+CHR$(V);RETURN
>

RUN
DECIMALNI BROJ=? 65535
HEXADECIMALMI EKVIVALENT=FFFF

>
RUN
DECIMALNI RROJ=? 1023
HFXADECIMALNI EKVIVALENT=03FF


>RUN
DECIMALNI BROJ=? 49780
HEXADECIMALNI EKVIVALENT=C274
```

Primjer 5: Određivanje dana u tjednu iz datuma.

```
LIST

 10 REM:
 12 REM: DAN U TJEDNU
 15 REM:
 20 DIMM$(12),W$(7)
 25 REM: FORMIRANJE POLJA
 30 FORI=1TO12
 40 READ M$(I)
 50 NEXT I
 60 FORI=1TO7
 70 READW$(I)
 80 NEXT I
 85 REM: OČITAVANJE DATUMA
 86 REM: I IZRAČUNAVANJE DANA U TJEDNU
 90 GOSUB300:REM UNOS DATUMA
 95 M=M1
 100 IFM>2 THEN 140
 110 M=M+10
 120 Y=Y-1
 130 GOTO150
 140 M=M-2
```

```
 150 I=Y-INT(Y/100)*100
 160 K=INT(Y/100)
 170 J=INT((13*M-l)/5)+INT(I/4)4INT(K/4)
 180 W=J+I+T-2*K
 190 W=W-INT(W/7)%7+l
 195 REM* NEDJELJA=1,PONEDJELJAK=2, ,..
 198 PRINT
 200 PRINTW$(W);" ";T;",';M$(Ml);Yl;"."
 210 DATA JANUAR ,FEBRUAR,MART,APRL,MAJ,JUNI,JULI
 220 DATA AUGUST,SEPTEMBAR,OKTOBAR,NOVEMBAR,DECEMBAR
 230 DATA NEDJELJA,PONEDJELJAK,UTORAK,SRIJEDA,ČETVRTAK,PETAK,SUBOTA
 240 END
 300 INPUTT,M1,Y
 310 Y1=Y
 320 IFM1=2ANDT=29THEN GOSUB400
 330 RETURN
 400 P=Y/4*P1=INT(P)
 410 DP=P-P1
 420 IFDP=0THEN RETURN
 430 PRINT"POGRESAN UNOS !!! "
 440 PRINT"GODINA NIJE PRESTUPNA"
 450 END
>

RUN
? 16,03,1959
PONEDJELJAK 16 .MART 1959

>
RUN
? 01,01,1985
UTORAK 1 .JANUAR 1985 .

>
RUN
? 03,08,1985
SUBOTA 3 .AUGUST 1985 .

>
RUN
? 08,03,1985
PETAK 8 .MART 1985 .

>
RUN
? 24,11,1956
SUBOTA 24,NOVEMRAR 1956
RUN

>
? 29,02,1985
POGREŠAN UNOS !!!
GODIMA NIJE PRESTUPNA

>
RUN
? 29,02,1984
SRIJEDA 29 .FEBRUAR 1984 .
```

Primjer 6: Sortiranje naziva po abecednom redu QUICK SORT

```
50 REM:
60 REM: QUICK SORT
70 REM:
110 INPUT"KOLIKO NAZIVA ŽELITE SORTIRATI":N
115 PRINT
120 DIMI$(N),A(N)
130 FORI=1TON
140 A(I)=I
160 PRINTI;:INPUTI$(I)
180 NEXT I
550 DIMS(30,2)
560 Sl=l
570 S(l,l)=l
580 S(1,2)=N
590 L=S(Sl,l)
600 R=S(S1,2)
610 Sl=Sl-1
620 I=L
630 J=R
```

```
640 H=A(INT(L+R)/2)
650 IFI$(A(1))>=I$(H)THEN680
655 IFI>=RTHEN680
660 I=I+1
670 GOTO650
680 IFI$(A(J)X=I$(H)THEN710
685 IFJ<=LTHEN710
690 J=J-1
700 G0T0680
710 IFI>JGOTO770
720 Z=A(I)
730 A(I)=A(J)
740 A(J)=Z
750 I=I+1
760 J=J-1
770 IFI<=JGOTO650
780 IF(R-I)<=(J-L)GOTO850
790 IFL>=JGOTO830
300 S1=S1+1
810 S(S1,1)=L
820 S(S1,2)=J
830 L=I
840 GOTO900
850 IFI>=RGOTO89O
860 Sl=Sl+l
870 S(S1,1)=I
880 S(S1,2)=R
890 R=J
900 IFR>LGOTO620
910 IFS1<>0G0T0590
950 PRINT
1000 FORI=1TON
1010 PRINTI;".";" ";l$(A(l))
1020 NEXT
1030 END
>

RUN
KOLIKO NAZIVA ŽELITE SORTIRATI? 10

1 ? FALKESTEINERHORSTUIEBER LIMPY
2 ? BOBALEC BJURA
3 ? PETKOVIC FRANE
4 ? FUCEK BJUKA
5 ? PECAROS ISTVAN
6 ? RABIC STIJEPAN
7 ? SCHUNBGLIEBER HANS
8 ? ANIC ROBERT
9 ? ALIC BLAGOJE
10 ? BOROVIC TOMISLAV

1 . ALIC BLAGOJE
2 . ANIC ROBERT
3 . BOBALEC BJURA
4 . BOROVIC TOMISLAV
5 . FALKESTEINERHORSTUIEBER LIMPY
6 . PECAROS ISTVAN
7 . PETKOVIC FRANE
8 . PUCEK BJUKA
9 . RABIC STIJEPAN
10 . SCHUNBGLIEBER HANS
```

2.6 ARITMETIČKO/LOGIČKE OPERACIJE

U odjeljku 2.1.2 upoznali smo nekoliko osnovnih aritmetičkih operacija: zbrajanje, množenje, potenciranje. Osim aritmetičkih operacija, BASIC mikroračunala ORAO, ima mogućnost obrade logičkih operacija.

Operacije i pripadajući operatori po prioritetu izvršavanja u aritmetičko/logičkom izrazu su:

OPERACIJA	**OPERATOR**
Potenciranje	^
negacija	-
množenje i dijeljenje	*/
zbrajanje i oduzimanje	+ -
jednakost	=

različito	<>
manje ili jednake	<=
veće ili	>=
logičko NE	NOT
logičko I	AND
logičko ILI	OR

Ostale aritmetičko/logičke operacije tvorimo pomoću osnovnih, pod uvjetom da pazimo na prioritet izvođenja osnovnih operacija, jer u protivnom možemo dobiti pogrešan rezultat.
Za pravilno izračunavanje kompleksnijih izraza koristimo operatore za grupiranje:

(lijeva zagrada
) desna zagrada

2.6.1 Aritmetičke operacije:

Primjer 1: Kodiranje pojedinih aritmetičkih operacija u BASIC-u

IZRAZ	KODIRANJE U BASIC-U
k	K
3,14159	3.14159
a+2,54	A+2.54
$b-c_a$	B-CA
xy	X*Y
i:j	I/J
c^2	C^2
$\frac{a+b}{c+2}$	(A+B)/(C+2)
$\frac{1}{x^2+y^2}$	1/(X^2+Y*2)
[(a+b)c-1]d	((A+B)*C-1)*D)

U slučaju da izraz u vanjskoj zagradi sadrži još izraz u unutarnjoj zagradi, prvo će se izvršiti izraz u unutarnjoj zagradi.
Broj lijevih zagrada, mora biti jednak broju desnih zagrada, u protivnom nam računalo šalje poruku o pogrešci.

Primjer 2:

Izraz A^B+C/D -E*F je ekvivalentan izrazu
(A*B)+(C/D)-(E*F) jer su operacije u zagradama većeg prioriteta od zbrajanja i oduzimanja.

Primjer 3:

Izraz A/B*C je ekvivalentan izrazu
(A/D)*C dok je izraz
I-J+K ekvivalenta izvršavanja operacija.

Ponekad je potrebno naglasiti redoslijed izvršavanja prema prioritetu jednako vrijednih operacija s lijeve strane prema desnoj kod množenja i dijeljenja.

Primjer 4:

8/4*2 nam da rezultat 4 dok
8/(4*2) da rezultat 1

Prilikom pisanja izraza, programer se mora držati slijedećih pravila:

1. Dva aritmetička operatora se ne smiju pojaviti jedan iza drugoga
   A*/B nije dozvoljeno
2. Sve računske operacije moraju biti izražene eksplicitno
   (A+8)*(C+D) a ne (A+B)(C+D)

3. Izraz a^{b^c} je dvosmislen, zato izbjegavamo zapis A^B^C
Nedvosmisleni zapisi su
A^(B^C) što znači $a^{(b^c)}$ odnosno (A^B)^C što znači $(a^b)^c$

Jednostavan primjer pokazuje da je

2^(3^2)=512

dok je (2^3)^2=64

2.6.2 Logičke operacije

Logičkim operacijama tvorimo logičke ili Boole-ove funkcije između operanada.
Osnovne logičke funkcije su:

negacija	NOT
konjukcija	AND
disjunkcija	OR

Navedeni redoslijed je ujedno i nivo prioriteta izvođenja logičkih operacija u BASIC-u.

Kompleksnije logičke funkcije tvorimo pomoću osnovnih.

Primjer 1: Tabela istine osnovnih logičkih operacija

A	B	A AND B	A OR B	NOT A	NOT B
0	0	0	0	1	1
0	1	0	1	1	0
1	0	0	1	0	1
1	1	1	1	0	0

Primjer 2: Tvorba kompleksnijih logičkih operacija pomoću osnovnih

a.) Ekskluzivna disjunkcija: Ekskluzivna disjunkcija je definirana kao

$$Z = A\overline{B} + \overline{A}B = A \oplus B$$

dok je ekvivalencija definirana kao negacija ekskluzivne disjunkcije, tj.

$$Z = AB + \overline{A}\,\overline{B} = \overline{A} \oplus \overline{B}$$

A	B	$A \oplus B$	$\overline{A} \oplus \overline{B}$
0	0	0	1
0	1	1	0
1	0	1	0
1	1	0	1

Gornja tabela sadrži tabelu istine za ekskluzivnu disjunkciju i ekvivalenciju između jednobitnih varijabli.
Sve logičke operacije možemo koristiti i na nivou višebitnih operanada,npr.:

Primjer 3: Logičke operacije s 8-bitnim operandima

a.) konjukcija :

A=	1010 1111	; 175 decimalno
B=	1111 1010	; 250 decimalno
A AND B=	1010 1010	; 170 decimalno

b.) disjunkcija

	0100 0001	; 65 decimalno
B=	1000 0000	; 128 decimalno
A OR B=	1100 0001	; 193 decimalno

c.) negacija

A=	1111 1110	; 254 decimalno
NOT A=	0000 0001	; 1 decimalno

d.) ekskluzivna disjunkcija

A=	1100 1011	; 203 decimalno
B=	1010 1101	; 173 decimalno
A EOR B=	0110 0110	; 102 decimalno

e.) ekvivalencija

A=	1100 1011	; 173 decimalno
B=	1010 1101	; 45 decimalno
A ENOR B=	1001 1001	; 153 decimalno

Primjer 4: Kodirajmo sada u BASIC-u logičke operacije iz primjera 3

a.) Konjukcija

```
10 REM: KONJUKCIJA
20 A=175:B=250
30 Z=A AND B

RUN
 170
```

b.) disjunkcija

```
10REM: DISJUNKCIJA
20 A=65:B=128
30 Z=A OR B
40 PRINTZ

RUN
 193
```

c.) negacija

```
10 REM: NEGACIJA
20 A=254
30 Z=NOT A
40 Z=Z AND 255
50 PRINTZ

RUN
 1
```

d.) ekskluzivna disjunkcija

```
10 REM: EOR
20 A=20
30 Z=NOT A AND B OR NOT B AND A

RUN
 102
```

e.) ekvivalencija

```
10 REM: ENOR
20 A=203:B=173
30 Z=NOT A AND B OR NOT B AND A
40 Z=NOT Z AND 255
50 PRINTZ

RUN
 153
```

Pokažimo slijedećim primjerom kako je moguće alfanumerički znak programski prikazati na ekranu u invertiranom obliku - (crno na bijeloj pozadini).

Primjer 5: Inverzija znaka

```
10 A$="A"
20 PRINTA$
30 A=ASC(A$)
40 A=A OR 128
50 AI$=CHR$(A)

RUN
A
A
>
```

2.7 FUNKCIJE

Funkcije, koje su uključene u BASIC-u mikroračunala ORAO možemo podijeliti u tri osnovne grupe:

- numeričke funkcije
- posebne funkcije
- korisnički definirane funkcije

2.7.1 Numeričke funkcije

Numerička funkcija je definirana u obliku:

y=f(x)

tj. varijabli "y" priredimo funkcijsku vrijednost "f" od argumenta x". Argument "x" može, također, biti numerički ili Boole-ov izraz.

2.7.1.1 ABS(X) funkcija

Funkcijom ABS(X) dobivamo apsolutnu vrijednost argumenta X u smislu:

ABS(X)=X ako je X>=0
ABS(X)=-X ako je x<0

2.7.1.2 INT(X) funkcija

Funkcijom INT(X) dobivamo prvu manju cjelobrojnu vrijednost decimalnog argumenta X.
npr.

INT(9.7)=9
INT(-5.1)=-6

2.7.1.3 RND(X) funkcija

Funkcija RND(X) generira slučajni broj između 0 i 1 na šest decimalnih mjesta. Ukoliko je potrebno generirati slučajni broj u nekom drugom intervalu, tada koristimo jednadžbu:

(B-A)*RND(7)+A

gdje je "B" gornja granica željenog intervala, a "A" donja granica intervala.

Upozorenje: RND(0) generira uvijek isti broj.

Primjer 1: Generiranje slučajnog broja između 0 i 100

```
10 Y=100"RND(7)
20 PRINTY
30 GOTO10
```

2.7.1.4 SGN(X) funkcija

Funkcija Y=SGN(X) priredi varijabli Y predznak argumenta X u smislu:

Y=0 za X=0
Y=1 za X>0
Y=-1 za X<0

Primjer 4:

```
10 A=0.125
20 B=0
30 C=-12.74
40 PRINTSGN(A),SGN(B),SGM(C)

RUN
1       0       -1
```

2.7.1.5 SQR(X) funkcija

Funkcija Y=SQR(X) priredi varijabli Y kvadratni korijen argumenta X.

npr.

```
PRINT SQR(2) [CR] daje
1.41421
```

2.7.1.6 EXP(X) funkcija

Funkcija Y=EXP(X) priredi varijabli Y potenciju broja "e" (e=2.71828). Vrijednost argumenta definira potenciju broja "e".

Primjer 1: operacija

Y=2.71828*X je ekvivalentna operaciji
Y=EXP(X)

2.7.1.7 LOG(X) funkcija

Funkcija Y=LOG(X) priredi varijabli Y prirodni logaritam argumenta X (po bazi "e").
Za dobivanje logaritma po bazi "b" od nekog broja koristimo jednadžbu:

log x=logx/log b

Primjer 1: Odredite dekadski logaritam od broja 2

```
10 B=10
20 Y=(LOG(2))/(LOG(10))
30 PRINTY
```

2.7.1.8 SIN(X) funkcija

Funkcija Y=SIN(X) priredi varijabli Y sinus od argumenta X izraženog u radijanima.

2.7.1.9 COS(X) funkcija

Funkcija Y=COS(X) priredi varijabli Y kosinus od argumenta X izraženog u radijanima.

2.7.1.10 TAN(X) funkcija

Funkcija Y=TAN(X) priredi varijabli Y tangens od argumenta X izraženog u radijanima.

2.7.1.11 ATN(X) funkcija

Funkcija Y=ATN(X) priredi varijabli Y vrijednost arkus tangens.

Na kraju navedimo da BASIC mikroračunala ORAO ne posjeduje funkcije kao:
y=arcsinx i y=arccosx
pa stoga moramo koristiti slijedeće jednadžbe:

a.) $\arcsin x = \arctan\left(\frac{x}{\sqrt{1-x^2}}\right)$; za |x| <1

b.) $\arccos x = \arctan\left(\frac{\sqrt{1-x^2}}{x}\right)$; za 0<x<=1

2.7.2 Posebne funkcije

Posebne funkcije BASIC-a za mikroračunalo ORAO koristimo za formatiranje ispisa na ekranu, odnosno štampaču.

2.7.2.1 POS(X) funkcija

Funkcija Y=POS(X) priredi varijabli Y broj kolone u kojoj se kursor nalazi nakon ispisa.

Primjer 1:

```
10 PRINT"ORAO";
20 PRINTPOS(X)

RUN

ORAO 4
```

jer se kursor nakon ispisa stringa ORAO nalazio u 4-toj koloni.

Primjer 2:

```
10 PRINT"ORAO"
20 PRINTPOS(X)
RUN
0
```

zašto 0 ?!

2.7.2.2 SPC(X) funkcija

Funkcija SPC(X) ubacuje X praznih mjesta u ispisu, tj. pomiče kursor prilikom ispisa za X pozicija u desno, s obzirom na trenutačnu poziciju.

Primjer 1:

```
10 PRINT SPC(10);"ORAO";
20 PRINT POS(X)
RUN
 ORAO 14
>
```

2.7.2.3 TAB(X) funkcija

Funkcija TAB(X) postavlja kursor na X-u kolonu brojeći od početka reda. U slučaju da je trenutačna pozicija veća od specificirane, tj. X-te, ispis se nastavlja u prvoj slijedećoj koloni.

Primjer 1:

10 PRINTTAB(10);"MIKRORAČUNALO";TAB(15);"ORAO"

ispis počinje od 10-te kolone s MIKRORAČUNALO,a od 15-te s ORAO

Primjer 2:

10 PRINTTAB(10);"MIKRORAČUNALO";TAB(5);"ORAO"

ispis počinje od 10-te kolone s MIKRORAČUNALO i tada se kursor nalazi u 23-oj koloni što je veće od vrijednosti specificirane u slijedećoj TAB funkciji, tako da će se ispis nastaviti od 23-e kolone s ORAO.
Na kraju napomenimo,da se funkcije POS(X),SPC(X) i TAB(X) koriste isključivo s PRINT naredbom.

2.7.3 Korisnički definirane funkcije

BASIC mikroračunala ORAO omogućava korisniku da sam definira svoju funkciju naredbom
DEF FNy(X)
gdje y može biti znak YU (ASCII) koda, dok X predstavlja argument definirane funkcije.

Primjer 1: Definirajmo funkciju y=exp(x) kao sumu prvih 7 članova Taylor-ovog reda funkcije $y=e^x$. Taylor-ov red funkcije $y=e^x$ je definiran kao:

$$y = e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + ... + \frac{x^n}{n!} + ...$$

Suma privih 7 članova je tada

$$y = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24} + \frac{x^5}{120} + \frac{x^6}{720}$$

odnosno kodirano u BASIC-u

```
10 DEFFNE(X)=1+X+(1/2)*X^2+(1/6)*X^3+(1/24)*X^4+(1/120)*X^5+(1/720)*X^6
20 FOR X=-5 TO 5
30 PRINT X;FNE(X)
40 NEXT X
```

Primjer 2: Definirajte funkciju koja računa površinu kruga

```
10 PI=3.141593
20 DEF FNP(R)=R*2*PI
30 INPUT R
40 PRINT FNP(R)

RUN
? 2
12.5664
```

2.8 RAD S MEMORIJOM

Dosadašnjim primjerima pokazali smo kako se simboličkim varijablama u BASIC-u priredi numerička vrijednost. Sve numeričke varijable su izražene aritmetikom kliznog zareza ("flooating point"), u opsegu od 10E-38 do 10E38 i svaka varijabla tada okupira 4 byte-a memorije.
Slijedećim primjerima ćemo pokazati kako je moguće da nam memorijska lokacija bude varijabla.

2.8.1 Upisivanje podatka u memorijsku lokaciju

Mikroprocesor 6502, koji predstavlja centralni hardware-ski element mikroračunala ORAO, podržava samo 8-bitnu aritmetiku, dok aritmetiku kliznog zareza generira BASIC interpreter.
Svaka memorijska lokacija je veličine 8 bita. U jednoj memorijskoj lokaciji je, dakle, moguće pohraniti 8 bitnu informaciju, tj. broj između:

0 i 255 ; (2^8-1)

Kada želimo da nam neka memorijska lokacija predstavlja 8 bitnu varijablu koristimo naredbu:

```
POKE I,J
```

tj. na lokaciju s decimalnom adresom I upišemo dekadski sadržaj J.

0<= I <=65535
0<= J <=255

Primjer 1: Na memorijsku lokaciju 0300 heksadecimalno upišite F0 heksadecimalno

- 0300 hexa =768 decimalno
- F0 hexa =240 decimalno

```
10 1=768:J=240
20 POKE I,J
RUN
>
```

Da se uvjerimo u ispravnost našeg programa pritisnimo tipku RESET Na ekranu sada imamo ispis:

```
                                        *** O R A O ***
*
_
```

Unesimo sada
`M0300` [CR]
`0300 F0` [@]

```
*
_
```

Vidimo da je naš program dobro izveden. Vratimo se ponovo u BASIC.

```
*BW [CR]
>
```

UPOZORENJE !!!

Naredbom POKE moguće je uništiti korisnički BASIC program, ukoliko upisujete na memorijske lokacije između 0400 i 1FFF heksadecimalno, odnosno 1024 i 8191 decimalno, ako prethodno ne ograničite opseg memorije, o čemu ce biti govora u odjeljku 2.8.4 (Primjer 2.).

2.8.2 Ispisivanje sadržaja memorijske lokacije

Naredbom POKE upisujemo na memorijsku lokaciju tako, da nam memorijska lokacija može predstavljati 8 bitnu varijablu.
Ako želimo ispisati sadržaj te varijable odnosno memorijske lokacije, služimo se naredbom:

```
PEEK(X)
```

gdje je X decimalna adresa željene memorijske lokacije između 0 i 65535.

Primjer 1: Ispišite sadržaj memorijske lokacije 0300 heksadecimalno:

```
10 PRINT PEEK(768)
RUN
240
>
```

jer je to bila posljednja vrijednost koju smo upisali na tu memorijsku lokaciju.

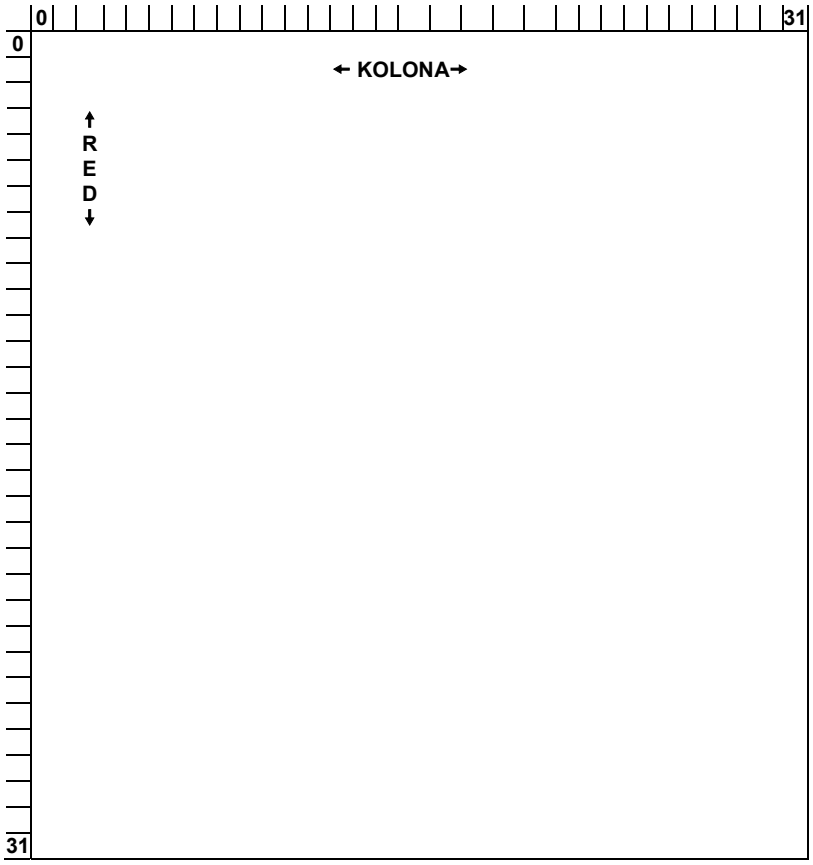
Primjer 2: Prikaz sadržaja memorije na ekranu LIST

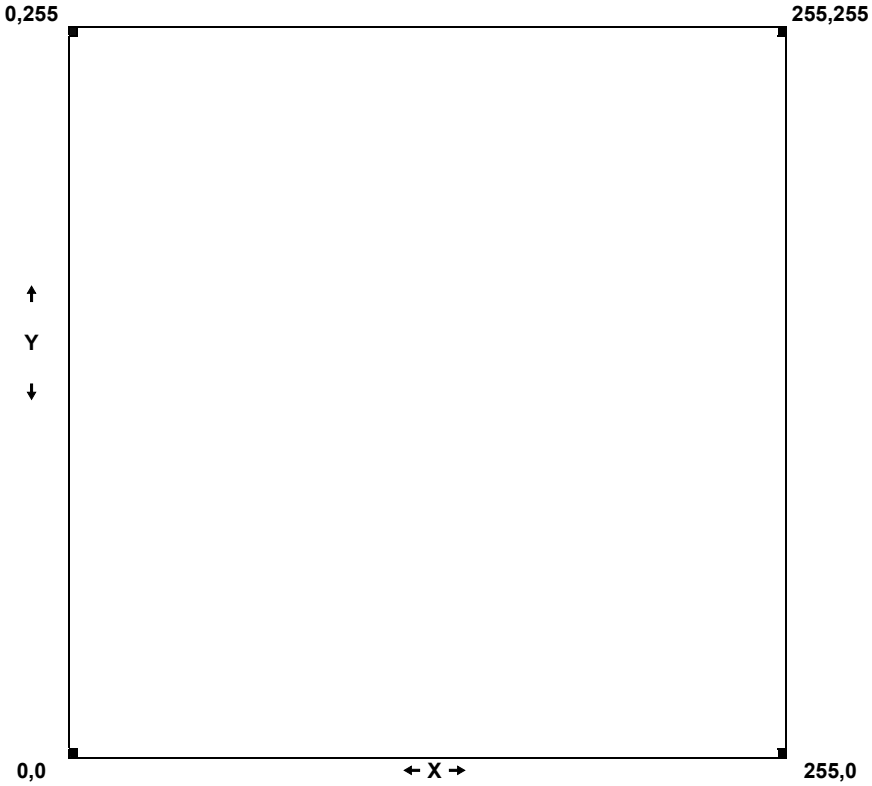
```
1 REM:=========================
2 REM:
3 REM: PRIKAZ SADRŽAJA MEMORIJE
4 REM: NA EKRANU HEXADECIMALNO
5 REM:
6 REM:=========================
7 :
10 INPUT"POČETNA ADRESA (dec.) =";S
20 INPUT"KONAČNA ADRESA (dec.) =";E
30 DEFFNM(X)=PEEK(X)
40 FORX=STOE
50 Q=X;GOSUB900
55 X$=Q$
60 Y=FNM(X):Q=Y:GOSUB900
70 Y$=RIGHT$(Q$,2)
80 PRINTX$;SPC(2);Y$
90 NEXT X
100 END
900 REM: DECIMALNO/HEXABECIMALNO
1000 B=INT(Q/256):GOSUB1400
1100 H1$=H$
1200 B=Q-B*256:GOSUB1400
1300 H2$=H$:Q$=H1$+H2$:RETURN
1400 V=BAND15:H=(BAND240)/16
1500 IFH>9THENH=H+55:GOTO1700
1600 H=H+48
1700 IFV>9THENV=V+55:GOTO1900
1800 V=V+48
1900 H$=CHR$(H)+CHR$(V):RETURN
>
```

```
RUN
POČETNA ADRESA (dec.) =? 56593
KONAČNA ADRESA (dec,) =? 56608
DD11 A2
DD12 FF
DD13 86
DD14 88
DD15 9A
DD16 A9
DD17 11
DD18 A0
DD19 BB
DD1A 85
DD1B 01
DD1C 84
DD1D 02
DD1E 85
DD1F 04
DD20 84
```

2.8.3 Ekran

Mikroračunalo ORAO podržava prikaz 32 alfanumerička znaka u 32 linije na ekranu TV prijemnika ili TV monitora, i grafiku rezolucije 256 x 256 točaka (o grafici će biti govora u poglavlju 2.11.). Slika prikazuje raster ekrana u alfanumeričkom modu i grafičkom modu. Korisniku je, također, dozvoljeno da sam definira veličinu aktivnog dijela prikaza. Ujedno je moguće programski redefinirati osnovni set znakova ("programmable character generator") mikroračunala ORAO.





2.8.3.1 Ograničavanje i pozicioniranje aktivnog djela prikaza

Definiramo:

POKE 236,L početnu kolonu postavimo
na L-tu poziciju
0<=L<=31
na RESET L=0

POKE 237,D posljednju kolonu postavimo na D-tu poziciju
0<=D<=31
na RESET D=31

Podrazumijeva se da L uvijek mora biti manji od D.

POKE 234, G prvi redak postavimo na G-tu poziciju
0<=G<=31
na RESET G=0

POKE 235,P posljednji redak postavimo
0<=P<=31
na RESET P=31

Podrazumijeva se da G uvijek mora biti manji od P.

Primjer 1: Ograničavanje aktivnog dijela prikaza

Želimo aktivni dio prikaza ograničiti na 32 kolone i 4 retka

```
L=0
D=31
G=0
P=3

10 PRINT CHR$(12)
20 L=0:D=31:G=0:P=3
30 POKE 236,L:POKE237,D
POKE234,G:POKE235,P
```

```
RUN
>
```

Unesemo li sada direktno u liniji :

`LIST` [CR]

vidimo da je sada moguće listanje samo u ograničenom dijelu prikaza.

2.8.3.2 Definiranje korisničkog seta znakova

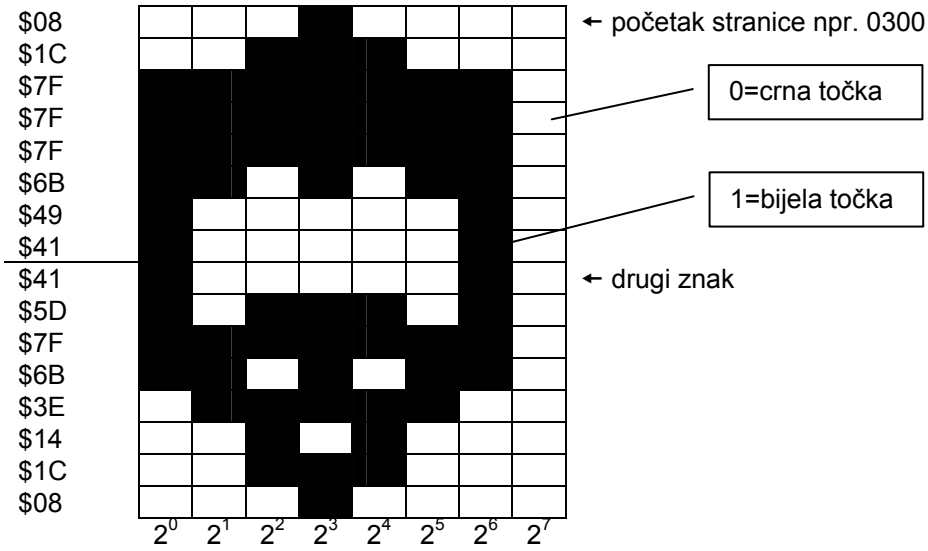
Mikroračunalo ORAO,osim standardnog seta YU (ASCII) znakova,nudi korisniku mogućnost da definira svoj vlastiti set znakova. Takva mogućnost postaje zanimljiva, kada, npr., korisnik želi, umjesto velikih slova latinice, velika slova ćirilice, ili, npr., umjesto malih slova, set grafičkih simbola itd.
Postupak se svodi na definiranje novog karakter generatora i inicijalizaciju vektora adrese novog seta znakova, a pri tome važe slijedeća pravila:

1. Novi karakter generator mora biti isključivo lociran na početku memorijske stranice isključivši nultu, prvu i drugu stranicu
2. Svaki znak okupira blok od 8 byte-a memorije
3. Ukoliko se novi karakter nalazi u memoriji koja je predviđena za tekstualni prostor BASIC-a, memoriju je potrebno ograničiti (vidi Primjer 2: poglavlja 2.8.4)
4. Vektori adrese su slijedeći:
   ZNAKOVI = 513 decimalno
   VELIKA SLOVA = 514 decimalno
   MALA SLOVA = 515 decimalno
5. Na vektor adrese upišemo stranicu memorije na kojoj se nalazi novi karakter generator

Primjer 1: Izmjena seta znakova malih slova

Želimo da nam se pritiskom na mala slova (uključimo s [PF1]) pojavljuju grafički simboli novog seta.

Svaki blok okupira blok od 8 byte-a u smislu



Opažamo da su težine bitova u karakter generatoru inverzne težinama binarnog koda, tj. pozicija bita 2^7 ima težinu 2^0,2^6 → 2^1 itd.
Nakon izmjene svih znakova pritiskom na tipku [PF1] uključimo mala slova. Sada će se pritiskom na tipku redom:

[^] pojaviti prvi simbol
[A] pojaviti drugi simbol itd.

U priloženom programu izmijenjena su samo prva dva znaka malih slova tj. [^] i [A], a ostale tipke daju neki slučajni oblik.

```
LIST
1 REM:
2 REM: IZMJENA SETA ZNAKOVA MALIH SLOVA
3 REM:
5 REM:
10 REM: NOVI SET ZNAKOVA SE NALAZI
11 REM: NA TREĆOJ STRANICI MEMORIJE
12 MP=3:REM STRANICA MEMORIJE
20 VA=515:REM VEKTOR ADRESE MALIH SLOVA
30 POKEVA,MP
40 REM: DEFINICIJA PRVIH DVAJU SIMBOLA
50 FORX=768TO783
60 READ Y:POKE X,Y
70 NEXT X
72 REM: OSTALE ZNAKOVE DEFINIRAMO BIJELO
75 FORX=784TO1023:POKEX,255:NEXT
80 END
90 DATA 8,28,127,127,127,107,73,65
100 DATA 65,93,127,107,62,20,28,8
>
```

2.8.4 Pozivanje strojnih programa iz BASIC-a

BASIC mikroračunala ORAO nudi nam mogućnost, da osim osnovnih naredbi, pozivamo pod kontrolom BASIC programa strojne programe koje smo sami napisali ili već postoje u ROM-u.
To postižemo funkcijom : U=USR(U)
Nakon što BASIC interpretira ovu funkciju, počinje izvođenje programa čija je startna adresa upisana na lokacijama:

- 11 (decimalno) - niži dio adrese decimalno
- 12 (decimalno) - viši dio adrese decimalno

Svi strojni programi koje pozivamo iz BASIC-a funkcijom USR moraju završavati naredbom RTS, kako bismo nakon izvođnja, kontrolu ponovo vratili BASIC interpreteru.

Primjer 1: Crtanje kružnice strojnim programom

U MONITOR ROM-u mikroračunala ORAO se nalazi strojni program za crtanje kružnice. Startna adresa tog programa je FF06 (heksadecimalno). Za korektno izvođenje navedeni program zahtijeva, tri parametra :

- koordinata središta XS,YS
- polumjer R

Prije pozivanja programa moramo na memorijske lokacije, kojima su definirani navedeni parametri, upisati željene numeričke vrijednosti.

ADRESE tih lokacija su :

- XS = 226 decimalno
- YS = 227 decimalno
- R = 248 decimalno

Dalje postupamo na slijedeći način:

1. 1.Inicijaliziramo vektor startne adrese -startna adresa heksadecimalno je

   FF 06
   viši dio ↑ ↑ niži dio

   - viši dio decimalno = 255
   - niži dio decimalno = 6

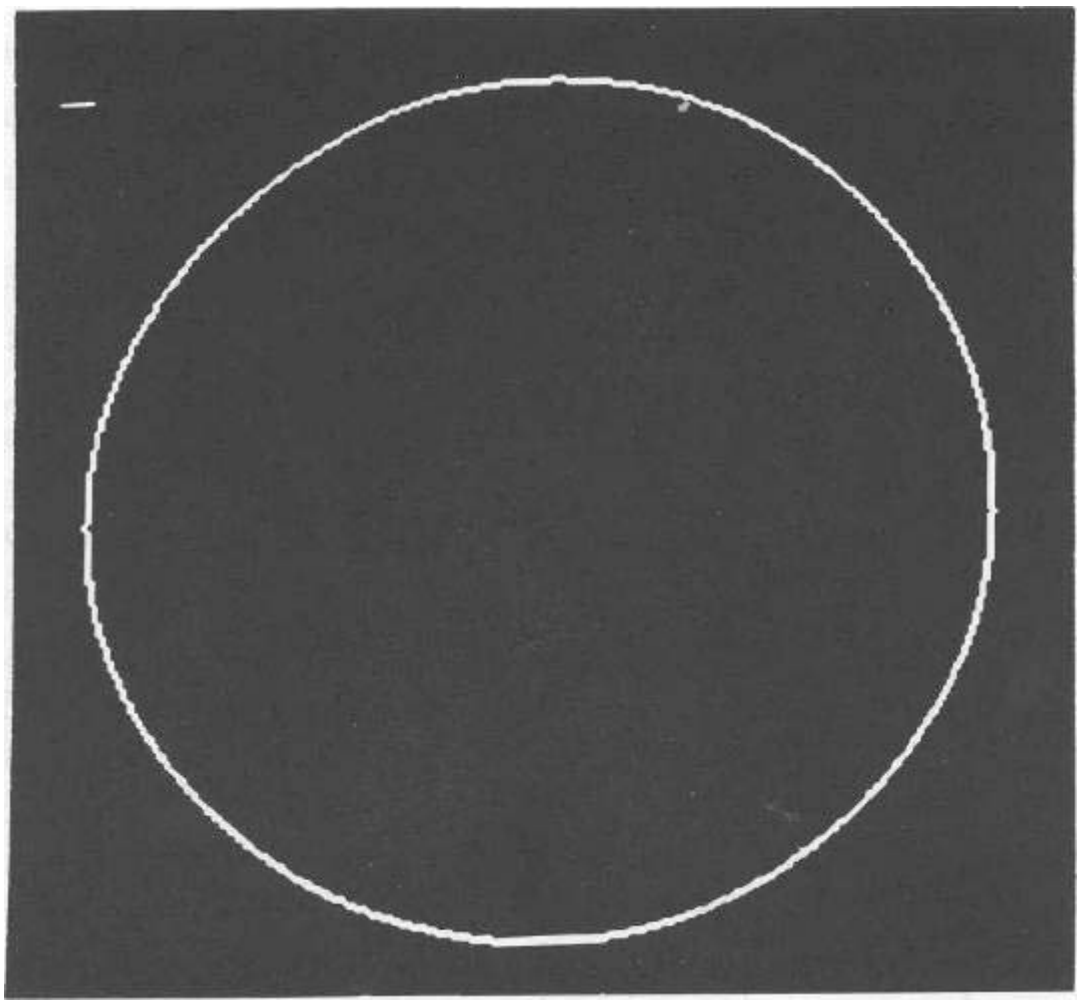
   -na lokaciju 11 dec. upišemo niži dio adrese 6 → 11

-na lokaciju 12 dec. upišemo visi dio adrese 255 → 12

3. Na lokacije XS,YS, i R upišemo željene parametre

4. Funkcijom USR pozovemo željeni program Navedeni postupak kodiran u BASIC-u je:

```
LIST
10 REM;
20 REM: POZIVANJE STROJNIH PROGRAMA
25 REM: USR funkcijom
30 REM:
50 REM: 1. INICIJALIZACIJA LEKTORA STARTNE ADRESE
60 POKE11,6:POKE12,255
70 REM: 2. INICIJALIZACIJA PARAMETARA
80 INPUT"KOORDINATE SREDIŠTA";XS,YS
90 INPUT"POLUMJER";R
100 POKE224,XS:POKE227,YS:POKE248,R
110 REM: 3. POZIVANJE STROJNOG PROGRAMA
120 U=USR(U)
130 END
>

RUN
KOORDINATE SREDIŠTA? 127,127
POLUMJER? 100
>
```



Primjer 2: Generiranje zvuka

Program za generiranje zvuka sastoji se od

1. 10-20 upisivanje strojnog programa u memoriju od lokacije 1000-1011 heksadecimalno odnosno 4096-4113 decimalno

2. 30 inicijalizacija vektora početne adrese

početna adresa strojnog programa je 1000 heksadecimalno
niži dio adrese je 00 = 0 decimalno
viši dio adrese je 10 = 16 decimalno

3. 40-50 unos parametara tona i upis na lokacije E0 i E1 heksadecimalno
   00E0 = 224 decimalno 00E1 = 225 decimalno

4. 52 pozivanje strojnog programa funkcijom USR

5. 55-80 18 byte-a strojnog programa (strojni kod je izražen decimalno)

UKOLIKO STE NAKON UKLJUČIVANJA RAČUNALA, BASIC POZVALI KAO ŠTO JE OPISANO U POGLAVLJU 1.1, TJ. BEZ PRETHODNOG OGRANIČAVANJA MEMORIJE, TADA PRIJE UNOSA PROGRAMA ZA GENERIRANJE ZVUKA PRITISNITE TIPKU "RESET" I IZVEDITE SLIJEDEĆU SEKVENCIJU:

```
*BC [CR]
MEMORIJA ? 4095 [CR]
DULJINA LINIJE ? [CR]
3070 LOKACIJA

>
_
```

Sada smo ponovo u BASIC-u, ali na raspolaganju imamo za BASIC programe samo 3070 lokacija umjesto 7167 kad memorija nije ograničena. Mehanizam ograničavanja memorije funkcionira na slijedeći način:

Na pitanje MEMORIJA ? mi unosimo maksimalnu adresu (decimalno) koja određuje vrh korisničke memorije ("TOP OF MEMORY"), koju BASIC interpreter smije uzeti za memoriranje našeg izvornog programa. U našem primjeru to je 4095 decimalno, odnosno 0FFF heksadecimalno, te smo na taj način dio memorije mikroračunala ORAO od 1000 heksadecimalno do 1FFF rezervirali za strojne programe ili u nekom drugom slučaju za podatke iz npr A/D pretvarača.
Minimalni iznos koji smijemo unijeti na pitanje MEMORIJA ? je 1026, ali u tom slučaju imamo na raspolaganju samo 1 byte memorije, što nije dovoljno niti za najmanji BASIC program.
Trenutno raspoloživi iznos memorije možemo ispitati funkcijom

```
PRINT FRE(X) [CR] ili
PRINT FRE(X$)[CR]

LIST

1 REM:======================
2 REM:
3 REM: GENERIRANJE ZVUKA
4 REM:
5 REM:======================
6 :
7 REM: UPISIVANJE STROJNOG PROGRAMA U MEMORIJU
8 :
10 FORX=4096TO4113
15 READY:POKEX,Y
20 NEXTX
25 REM: INICIJALIZACIJA VEKTORA POČETNE ADRESE $1000
30 POKE11,0:POKE12,16
45 INPUT"VISINA TONA (MAX. 255)";F %
48 T=255/F
49 REM: UPISIVANJE PARAMETARA TONA U MEMORIJU $E0 I $E1
50 POKE224,F:POKE225,T
52 U=USR(U):REM POZIVANJE STROJNOG PROGRAMA
54 END
55 REM: STROJNI PROGRAM ZA ZVUK
60 DATA 162,255,164,224,140,0
70 DATA 136,136,208,253,202,208
80 DATA 245,198,225, 16,239,96
>

*
*
*B*
*B* GENERIRANJE ZVUKA
*B*
*X10001011
 1000 A2 FF LDX *FF
```

```
 1002 A4 E0 LDY E0
 1004 8C 00 88 STY 8800
 1007 88 DEY
 1008 D0 FD BNE 1007
 100A CA DEX
 100B D0 F5 BNE 1002
 100D C6 E1 DEC E1
 100F 10 EF BPL 1000
 1011 60 RTS
*
*RUN
VISINA TONA (MAX. 255)? 255

>
RUN
VISINA TONA (MAX. 255)? 47
```

2.9 SISTEMSKE KOMANDE

Komande su naredbe koje BASIC interpreter izvršava, nakon što su unesene preko tastature u računalo,bez numeriranja programske linije -izravno.
Sistemske komande se koriste za izravan ulaz ili izlaz podataka sa, ili u periferne jedinice.
Pojedine sistemske komande,također, direktno mijenjaju sadržaj tekstualnog prostora BASIC-a.

2.9.1 LOAD I LOADC komande

LOAD komanda briše memoriju tekstualnog prostora i automatski učitava novi korisnički program s audio kazete. Kompletna sintaksa LOAD komande je :

```
LOAD"ime programa"početna adresa
```

1. Ime programa smije sadržavati maksimalno 12 znakova YU (ASCII) koda osim znakova navodnika
2. Početna adresa mora obavezno biti izražena heksadecimalno
3. Za pozivanje BASIC programa s audio kazete početna adresa je uvijek 0400 i nije obavezna u sintaksi komande LOAD

Primjer 1: Pozivanje BASIC programa s kazete

Pozovimo s demo kazete program pod imenom

```
LOAD"SORT.BAS"
REPRODUKCIJA ?
```
uključimo reprodukciju na kazetofonu i [CR]

Sada računalo traži na kazeti program pod imenom SORT.BAS. Ukoliko se program SORT.BAS ne nalazi na početku kazete računalo će ispisati imena svih programa koji se nalaze ispred traženog. Do pojave prompta ">" i kursora računalo čita program sa kazete i memorira ga u tekstualni prostor.

Nakon :

```
>
_
```

Računalo je učitalo program koji je sada spreman za izvođenje.

Za pozivanje strojnih programa moramo koristiti potpunu sintaksu naredbe LOAD, tj. moramo navesti početnu lokaciju od koje će program nakon učitavanja biti smješten.

Primjer 2: Pozivanje strojnih programa s kazete

Pozovimo s demo kazete program ZVUK.BIN

```
LOAD"ZVUK.BIN"1000
```
[CR]
```
REPRODUKCIJA ?
```
uključimo reprodukciju [CR]

```
>
_
```

Sada je program ZVUK.BIN učitan u memoriji od lokacije 1000.
Budući da je program za zvuk pozicijski neovisan, možemo ga pozvati u memoriju i od lokacije 1500

```
LOAD"ZVUK.BIN"1500 [CR]
REPRODUKCIJA ? [CR]

>
_
```

Sada je naš program smješten u memoriji od lokacije 1500.

LOADC komanda služi za ispis kataloga imena programa koji su na kazeti.

```
LOADC [CR]
REPRODUKCIJA ? [CR]
```

Nakon toga na ekranu će nam se sekvencijalno pojavljivati imena svih programa koje je računalo našlo na kazeti.

```
LOADC
REPRODUKCIJA ?

TORANJ.BAS 0400 063C 023C
HISTOGRAM.BAS 0400 067D 029B
SORT.BAS 0400 05BC 01BC
DATUM.BAS 0400 072C 032C
HEXADECBAS 0400 0500 0100
DECHEXA,BAS 0400 052E 012E
QUICKSORT,BAS 0400 06C7 02C7
BUTWORTH.BAS 0400 071A 031A
ZVUK.BIN 1000 1011 0011
VURA.BAS 0400 0B13 0913
FILTRIRANJE 0400 0615 0215
```

Vidimo da je ispis podijeljen u četiri polja:

1.polje : IME
2.polje : početna adresa
3.polje : konačna adresa
4.polje : veličina programa (heksadecimalno)

Katalogiranje kazete prekidamo pritiskom na tipku [CTL]

2.9.2 SAVE komanda

SAVE komandu koristimo za pohranjivanje programa koje smo napisali u BASIC-u ili MINIASSEMBLER-u.
Potpuna sintaksa naredbe SAVE je

```
SAVE"ime programa"početna adresa,konačna adresa
```

1. Ime programa smije sadržavati 12 znakova YU (ASCII) koda (isključeni znakovi navodnika i kontrolni kodovi)
2. Početna i konačna adresa moraju obavezno biti izražene heksadecimalno
3. Za pohranjivanje BASIC programa početnu i konačnu adresu ne treba navesti, jer računalo automatski uzima 0400 za početnu adresu, a konačnu izračuna iz dužine programa

Primjer 1: Pohranjivanje BASIC programa na kazetu

```
SAVE"TEST" [CR]
SNIMANJE ? uključimo snimanje i [CR]
```

Nakon toga iz zvučnika će se čuti zvuk programa koje računalo kodira u audio signal i šalje prema kazetofonu.
Nakon:

```
>
_
```

program je snimljen na kazetu.

Primjer 2: Pohranjivanje sadržaja ekrana na kazetu

`SAVE"EKRAN"6000,7FFF` [CR]
`SNIMANJE ?` uključimo snimanje i [CR]

do ponovne pojave prompta i kursora računalo šalje u kazetofon sadržaj memorijskih lokacija od 6000 do 7FFF koje služe za video prikaz ("VIDEO RAM")

Srednja brzina prijenosa podataka između računala i kazetofona je 1500 baud-a (1500 bita u sekundi).

2.9.3 LIST komanda

LIST komanda može biti korištena na četiri načina:

1. LIST bez dodatnih specifikacija
2. LIST sa jednom specifikacijom
3. LIST sa dvije specifikacije
4. LIST sa tri specifikacije

Komanda LIST nam služi za ispisivanje izvornog programa (listanje) na ekranu ili štampač.

1. LIST lista cijeli izvorni program
2. LIST n lista samo n-tu programsku liniju
3. LIST-n lista program do n-te linije
   LISTn- lista program od n-te linije do kraja
4. LISTn-m lista od n-te do m-te linije uz uvjet da je m>n

Pritiskom na tipke [CTL]i[C] istovremeno, prekidamo listanje programa.

2.10 KONTROLNE KOMANDE/NAREDBE

Kontrolne komande nam služe za kontrolu izvođenja BASIC programa.

2.10.1 RUN komanda

Komanda RUN starta izvođenje BASIC programa počevši s linijom koja ima najmanji redni broj, te ujedno pred izvođenjem programa sve varijable postavi na nulu. Komandu RUN možemo, također, koristiti sa dodatnom specifikacijom,tj.:

`RUN n` počinje izvoditi program od n-te linije i postavi sve varijable na nulu

Izvođenje BASIC programa prekinemo pritiskom na tipke [CTL] i [C]

2.10.2 STOP naredba

STOP naredba zaustavlja izvođenje programa i ispisuje poruku s rednim brojem linije u kojoj je interpretirana.

Primjer 1:

```
10 PRINT"MIKRORACUNALO"
20 STOP
30 PRINT"ORAO"
RUN
MIKRORAČUNALO
STOP U 20
>
_
```

2.10.3 CONT komanda

Kada računalo interpretira komandu CONT,nastavlja s izvođenjem programa od mjesta gdje je bio prekinut naredbom STOP ili pritiskom na tipke [CTL] [C].

2.10.4 END naredba

Naredba END definira kraj programa, ali nije obavezna.

2.10.5 NEW komanda

Komandom NEW brišemo izvorni BASIC program iz memorije 3 postavljamo kazaljku tekstualnog prostora na početak.

2.11 GRAFIKA

Mikroračunalo ORAO podržava grafiku visoke rezolucije 256 x 256 u pet grafičkih načina. Ekran u svih pet grafičkih načina predstavlja prvi kvadrant kartezijevog koordinatnog sustava.
Grafički način definiramo tako, da na lokaciju GS=522 (decimalno) upišemo grafički kod M.

M=0 crta bijelo
0<M<=127 crta crno
127<M<=255 crta inverzno

Očito je da,s obzirom na pozadinu i odabrani i M, možemo tvoriti pet grafičkih načina.

M	POZADINA	NAČIN CRTANJA
0	crno	bijelo na crnom
1	crno	crno na crnom
0	bijelo	crno na bijelom
255	xxx	inverzno
	xxx ili bijelo ili crno	

Nakon RESET-a mikroračunala,sadržaj lokacije GS je 0.

2.11.1 MOV naredba

Naredbom MOV X,Y samo postavljamo grafički kursor na apsolutne pozicije X i Y, pri čemu niti X niti Y ne smiju biti veći od 255.

Unesite direktno u liniji:

```
MOV 0,0 [CR]
>
```

Opažate da se ništa atraktivno nije dogodilo, jer grafički kursor nije vidljiv, iako smo ga prethodnom komandom postavili u ishodište koordinatnog sustava.

2.11.2 DRAW naredba

Naredbom DRAW X,Y izvučemo liniju X,Y.

unesite direktno u liniji:

```
DRAM 255,255 [CR]
```

čime ste izvukli dijagonalu preko ekrana. Grafički kursor se sada nalazi na poziciji 255,255. Da se u to uvjerite unesite ponovo direktno u liniji:

```
DRAW 127,0 [CR;]
```

čime ste izvukli liniju do točke 127,0 relativno od posljednje pozicije grafičkog kursora.

Primjer 1: Crtanje kvadrata

```
10 A=0:B=127:C=255
20 MOVA,B
30 DRAWB,C
40 DRAWC,B
50 DRAWB,A
```

2.11.3 PLOT naredba

Unesite direktno u liniji
Naredba PLOT X,Y postavlja grafički kursor na poziciju X,Y i na toj poziciji nacrta točku.

```
[CTLXL]PLOT 127,127 [CR]
```

Čime ćete nacrtati jednu grafičku točku na sredini ekrana.

Primjer 1: Crtanje eksponencijalne funkcije naredbom PLOT

```
10 FORX=0TO255
20 Y=100*(1-EXP(-X/10))
30 PLOT X,Y
40 NEXT X
```

Primjer 2: Crtanje eksponencijalne funkcije naredbama MOV i DRAW

```
10 MOV0,0
20 FOR X=0 TO 255
30 Y=100*(1-EXP(-X/10))
40 DRAW X,Y
50 NEXT X
```

Opažamo da nam naredba DRAW vrši linearnu interpolaciju izmedju dvije točke.

2.11.4 Animirana grafika

Pojam animirane grafike se odnosi na crtanje objekata koji se pokreću na grafičkom zaslonu. Da bi objektu na ekranu dali pokret, postupamo na slijedeći način:

- KORAK1: Nacrtamo željeni objekt u prvoj fazi pokreta i prikazujemo ga T1 vremena
- KORAK2: Brišemo nacrtani objekt potpuno ili djelomično
- KORAK3: Crtamo objekt u drugoj fazi pokreta na istoj ili promijenjenoj poziciji i prikazujemo ga T2 vremena
- KORAK4: Da li su sve faze pokreta prikazane:
  - ako nisu idi na KORAK2
  - ako jesu idi na KORAK5:
- KORAK5: Kraj animacije

Brisanje objekta na ekranu mikroračunala ORAO omogućava nam promjena grafičkog načina.

Primjer 1: Gibanje radij vektora po kružnom luku

```
LIST
1 REM:
2 REM: ANIMACIJA RADIJ VEKTORA
3 REM:
4 PRINTCHR*(12)
10 K=2*3.141593/256
20 GS=522:MO=6:M1=1
30 FORT=0T063STEP5
40 Y=100*SIN(K*T)
50 X=127*C0S(K*T)
60 POKEGS,M0
80 GOSUB100
90 POKEGS,M1:GOSUB200
95 GOSUB100
97 NEXTT
98 POKEGS,M0:GOSUB100
99 END
100 MOV0,0;DRAWX,Y:RETURN
200 FORA=0TO400:NEXTA
210 RETURN
>
```

Analogni sat je slijedeći primjer animirane grafike.

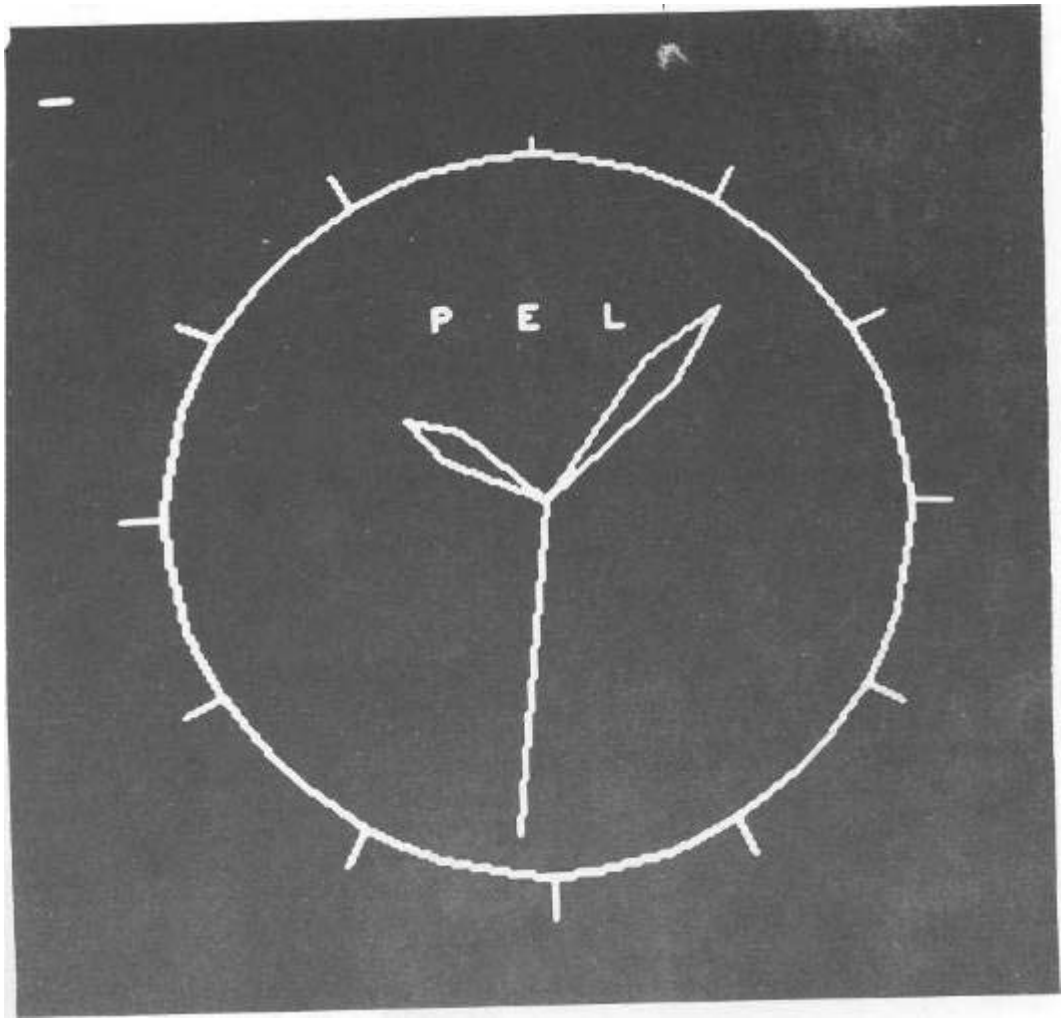
Primjer 2: Analogni sat

Analogni sat je slijedeći primjer animirane grafike.

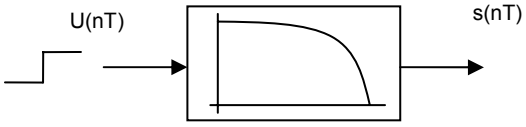
Primjer 2: Analogni sat

```
10 REM:==================
15 REM:
18 REM: ANALOGNI SAT
20 REM:
22 REM:==================
25 PRINTCHR$(12)
37 INPUT"TOČNO VRJEME SAT,MIN";A$,B$
38 GOSUB3000:REM* RAČUNANJE POZICIJA KAZALJKI
40 M=522:CR=1:BJ=O
41 REM
42 REM GLAVNO TIJELO
43 REM
45 PRINTCHR$(12)
46 PI=3.14159$K=2*PI/600
47 X0=127*Y0=127
50 GOSUB200:REM* BROJČANIK
51 REM:
52 REM: POČETAK IGRE
53 REM:
54 R=A:REM==POČETNA POZICIJA MALE KAZALJKE
55 F0RT2=600T00STEP-10
56 R1=B+T2
57 POKEM,0
60 GOSUB6500
62 GOSUB800:GOSUB6000
64 GOSUB600
65 GOSUB6800
66 GOSUB800:GOSUB6000
68 GOSUB600:REM=CRTANJE MALE
70 GOSUB400:REM=SEKUNDER
75 POKEM,l:GOSUB6500:GOSUB800:GOSUB6000*GOSUB600
80 W=R1/6O
90 W1=INT(W)
100 DW=W-W1
110 IFDW=.5THENGOSUB1000
190 NEXT
195 GOTO55
199 END
200 REM*
210 REM* CRTANJE BROJCANIKA
220 REM*
222 POKEM,BJ*REM==BIJELO NA CRNOM==
237 C=100
238 POKE11,6:POKE12,255
240 POKE226,X0*POKE227,Y0
250 POKE248,C-1
260 U=USR(U)
280 FORT=0TO599STEP50
290 X=X0+C*COS(K*T):Y=Y0+C*SIN(K*T)
300 MOVX,Y
310 X2=X+10*C0S(T*K):Y2=Y+10*SIN(K*T)
320 DRAWX2,Y2
330 NEXT
350 RETURN
400 REM*
410 REM* SEKUNDER
420 REM*
430 FORT=600TO0STEP-10
435 POKEM,0
440 MOVX0,Y0
445 GOSUB500
450 DRAWXS,YS
460 FORD=0TO330:NEXT
462 POKE11,0:POKE12,3
464 U=USR(U)
470 GOSUB2000:REM*BRISANJE. SEKUNDERA
480 POKEM,0:GOSUB6800:GOSUB800:GOSUB6000:GOSUB600
482 GOSUB6500:GOSUB800:GOSUB6000:GOSUB600
498 NEXT
499 RETURN
500 REM*
```

```
510 REM* POZICIJA SEKUNDERA
520 REM*
525 T1=T+150
530 XS=X0+90*COS(T1*K)
540 YS=Y0+90*SIN(T1*K)
550 RETURN
600 REM*
610 REM* CRTANJE KAZALJKI
620 REM*
630 MOVX0,Y0:DRAWP1,P2
640 MOVP1,P2:DRAWP3,P4
645 MOVP3,P4:DRAWP5,P6
650 MOVP5,P6:DRAWX0,Y0
660 RETURN
800 REM*
810 REM* POZICIJA KAZALJKI
820 REM*
840 M1=X0+ E*COS((F-G)*K)
845 M2=Y0+ E*SIN((F-G)*K)
850 M3=X0+ H*COS(F*K)
855 M4=Y0+ H*SIN(F*K)
860 M5=X0+ E*COS((F+G)*K)
865 M6=Y0+ E*SIN<(F+G)*K)
870 RETURN
1000 REM*
1100 REM* POMAK MALE
1200 REM*
1300 FOKEM,l:GOSUB6800:GOSUB800:GOSUB6000:GOSUB600
1310 R=R-5:POKEM,0
1320 GOSUB6800:GOSUB800:GOSUB6000:GOSUB600
1400 RETURN
2000 REM*
2100 REM* BRISANJE SEKUNDERA
2200 REM*
2300 POKEM,1
2400 MOVX0,Y0:DRAWXS,YS
2500 RETURN
3000 REM*
3100 REM* RAČUNANJE POČETNIH POZICIJA KAZALJKI
3200 REM*
3300 A=VAL(A$):B=VAL(B$)
3400 B=(60-B)*10+150
3500 A=(12-A)*50+150
3700 I=0
3800 FORX=750TO150STEP-60
3900 IFX<BTHEN4200
4000 I=I+1
4100 NEXTX
4200 I=5*(I-l):A=A-I
4210 F0RX=768TO781
4220 READY:POKEX,Y:NEXT
4300 RETURN
4400 REM:
4500 REM: STROJNI PROGRAM ZA ZVUK
4600 REM:
5000 DATA 162,128,160,112,140,0
5100 DATA 136,136,208,253,202,208
5200 DATA 245,96
5300 REM: POSTAVLJANJE POZICIJA ZA CRTANJE
6000 Pl=Ml:P2=M2:P3=M3:P4=M4:P5=M5:P6=M6
6100 RETURN
6200 REM: OSNOVNI PARAMETRI VELIKE KAZALJKE
6500 E=50:F=Rl:G=10:H=75
6600 RETURN
6700 REM: OSNOVNI PARAMETRI MALE KAZALJKE
6800 E=30:F=R:G=15:H=45
6900 RETURN
```



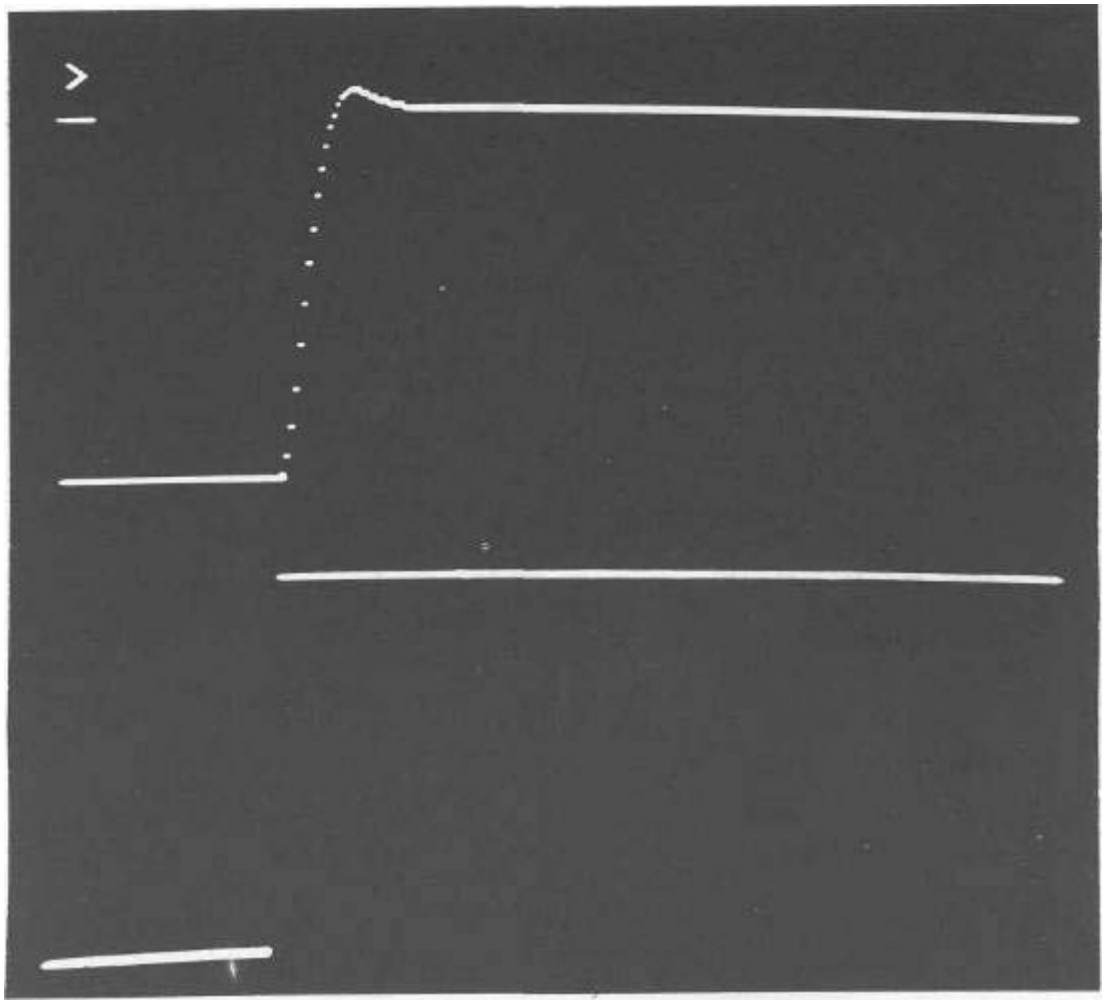
2.11.5 Primjeri grafičkih programa



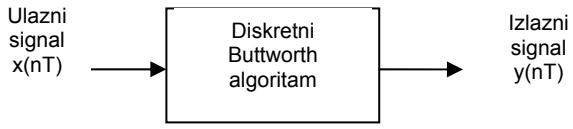

```
10 REM: ====================================
15 REM:
20 REM: DIGITALNI Buttworth FILTER
30 REM:      2-og reda
35 REM:
36 REM:====================================
40 REM: FREKVENCIJA REZANJA (-3db)=1000  Hz
50 REM: Filter je realiziran
60 REM: BILINEARNOM TRANSFORMACIJOM
65 REM: analognog filtra uz
67 REM: T=40E-6 sec ("mapping factor")
70 REM:
80 REM: Program generira 256 točaka  odziva
90 REM: filtra na STEP funkciju
95 REM:
98 DIMYN(255)
100 X=l:W2=0:Wl=0
110 FORN=0TO255
120 W0=X+1.649*Wl-0.702*W2
130 Y=1.323E-2*W0+2.646E-2*Wl+W2*1.323E-2
140 PRINTN,Y
145 YN(N)=Y
150 REM:
155 REM:  REALIZACIJA KAŠNJENJA  *
156 REM:  W2=W0*(z**(-2))
158 REM:  Wl=W0*(z**(-l))
160 W2=W1*W1=W0
170 NEXT  N
175 REM:
```

```
176 REM:  CRTANJE ODZIVA
178 REM:
180 PRINTCHR$(12)
190 FORN=0TO50
200 PL0TN,0
210 PL0TN,127
220 NEXT N
230 FORN=50TO255
240 PLOTN,100
250 PLOTN,YN(N-50)*100+127
260 NEXT N
300 END

RUN
 0             .01323
 1             .0615063
 2             .145056
 3             .248941
 4             .361593
 5             .474431
 6             .581419
 7             .678629
 8             .763822
 9             .836066
 10            .895389
 11            .942499
 12            .978537
 13            1.00489
 14            1.02306
 15            1.0345
 16            1.04063
 17            1.0427
 18            1.04181
 19            1.03889
 20            1.0347
 21            1.02983
 22            1.02476
 23            1.01981
 24            1.0152
 25            1.01108
 26            1.00752
 27            1.00454
 28            1.00213
 29            1.00024
 30            .998828
 31            .997816
 32            .997141
 33            .996739
 34            .996549
 35            .996519
 36            .996602
 37            .996761
 38            .996964
 39            .997187
 40            .997413
 41            .997628
 42            .997825
 43            .997999
 44            .998147
 45            .998269
 46            .998366
 47            .998441
 48            .998496
 49            .998535
 50            .998559
 51            .998573
 52            .998578
 53            .998577
 54            .998572
 55            .998564
 56            .998554
 57            .998544
 58            .998534
 59            .998525
 60            .998516
 61            .998509
 62            .998503
 63            .998498
 64            .998494
 65            .998492
 66            .998489
 67            .998488
```

```
 68            .998487
 69            .998487
 70            .998487
 71            .998487
 72            .998487
 73            .998487
 74            .998488
 75            .998488
 76            .998488
 77            .998488
 78            .998489
 79            .998489
 80            .998489
 81            .998489
 82            .998489
 83            .998489
 84            .99849
 85            .99849

 STOP U  130
>
CONT
 210           .998491
 211           .998491
 212           .998491
 213           .998491
 214           .998491
 215           .998491
 216           .998491
 217           .998491
 218           .998491
 219           .998491
 220           .998491
 221           .998491
 222           .998491
 223           .998491
 224           .998491
 225           .998491
 226           .998491
 227           .998491
 228           .998491
 229           .998491
 230           .998491
 231           .998491
 232           .998491
 233           .998491
 234           .998491
 235           .998491
 236           .998491
 237           .998491
 238           .998491
 239           .998491
 240           .998491
 241           .998491
 242           .998491
 243           .998491
 244           .998491
 245           .998491
 246           .998491
 247           .998491
 248           .998491
 249           .998491
 250           .998491
 251           .998491
 252           .998491
 253           .998491
 254           .998491
 255           .998491
```



Primjer 2: Filtriranje signala



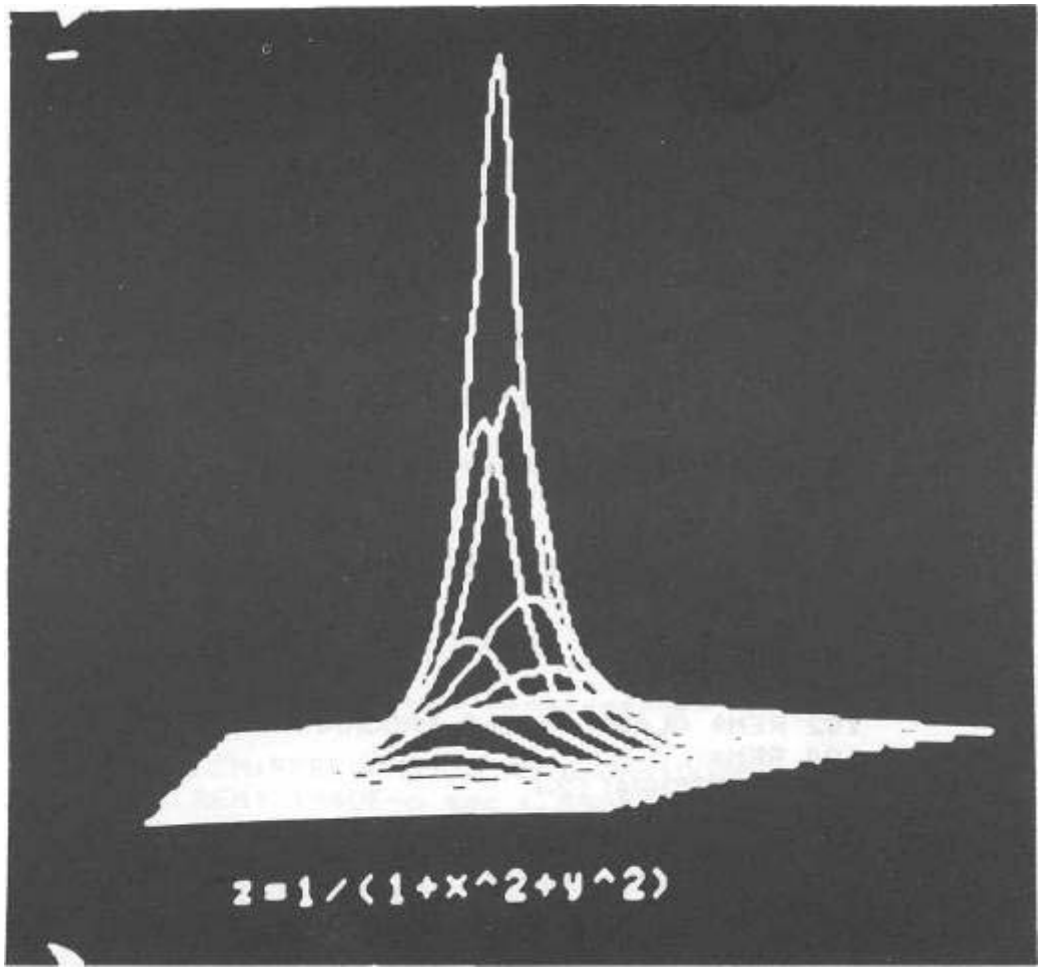
U programu se koristi isti algoritam kao i u primjeru 1

```
LIST
 10 REM:======================
 15 REM:
 20 REM:    FILTRIRANJE
 35 REM:
 36 REM:======================
 37 :
 40 REM:FREKVENCIJA REZANJA=1000Hz
 67 REM: T=40E-6 sec ("sampling rate")
 95 :
 96 PRINTCHR$(12)
 98 PRINT"OSNOVNA FREKVENCIJA = 97.65 Hz"
 99 PRINT"SUPERPONIRANA FREKVENCIJA=3906.25 Hz"
 100 W2=0:W1=0
 101 K=2*3.14159/256
 102 DEFFNS(N)=SIN(K*N)+.5*SIN(40*K*N)
 110 FORN=0TO255
 120 W0=FNS(N)+1.649*W1-.702*W2
 130 Y=1.323E-2*W0+2.646E-2*Wl+W2*1.323E-2
 140 PLOTN,160+50*Y
 145 PLOTN,100+50*FNS(N)
 150 REM:
 155 REM: REALIZACIJA KAŠNJENJA
 156 REM:
 160 W2=W1:W1=W0
 170 NEXT N
 180 END
```

>

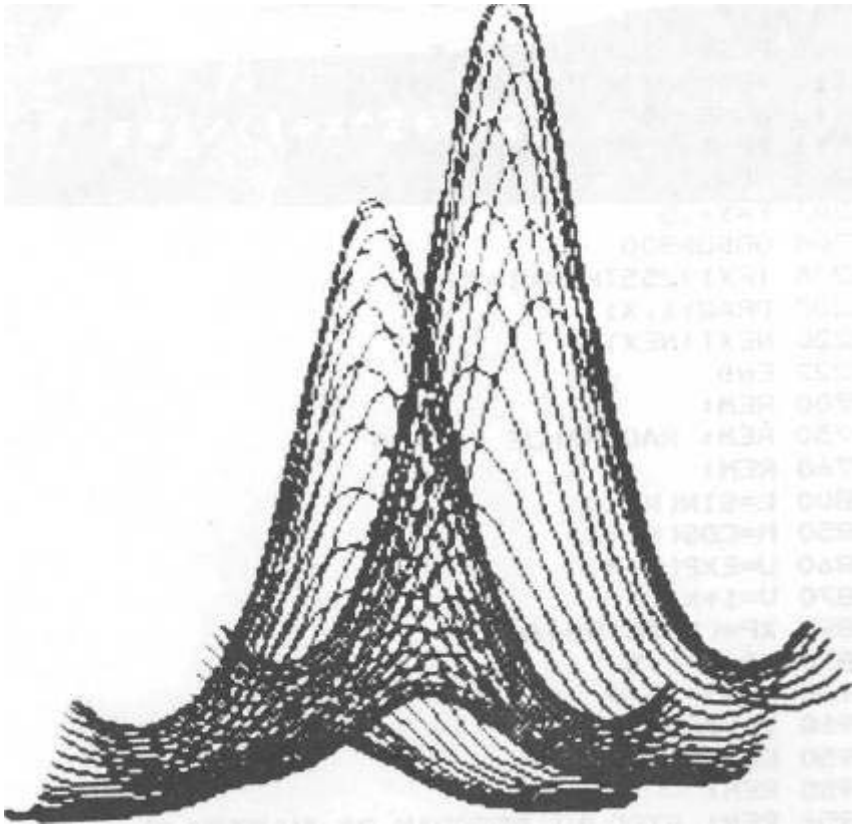
Primjer 3: Crtanje prostorne funkcije z=1/(1+x^2+y^2) na crnoj pozadini

```
LIST

 10 REM**********************
 20 REM*                     *
 30 REM*    T 0 R A N J      *
 40 REM*                     *
 50 REM*   z=1/(l+x^2+y^2)   *
 60 REM*                     *
 70 REM**********************
 75 :
 80 REM*
 81 REM* DEFINICIJA GRAFIKE
 82 REM*
 97 :
 98 P0KE522,0
 100 REM*
 102 REM* GLAVNO TJELO PROGRAMA
 104 REM*
 105 PRINTCHR$(12)
 110 CZ=30
 120 FORX=10TO-6STEP-1
 130 FORY=10TO-10STEP-.O125
 135 GOSUB800
 140 MOVY1,X1
 150 Y=Y-.25
 160 GOSUB800
 170 DRAWY1,X1
 190 NEXT:NEXT
 200 END
 800 REM*
 810 REM* RAČUNANJE TOČAKA
 820 REM*
 830 U=1/(1+X*X+Y*Y):V=1+X/CZ
 840 XP=(U*150+30)/V
 850 X1=XP*1.35
 860 YP=(Y+15)/V
 870 Y1=YP*8
 880 RETURN
 >
```



Primjer 4: Crtanje prostorne funkcije z=exp(sinx+cosy) na bijeloj pozadini

```
LIST

 10 REM**********************
 12 REM*
 20 REM* G R E B E N I
 22 REM*
 25 REM* Z=EXP(SIN(X)+C0S(Y))
 30 REM*
 32 REM**********************
 33 :
 34 REM: INVERZIJA EKRANA
 35 PRINTCHR$(12):GOSUB36:GOTO40
 36 FORX=768TO791*READC:POKEX,C:NEXTX
 37 POKEll,0:POKE12,3:U=USR(U)
 38 RETURN
 39 REM* GLAVNO TJELO PROGRAMA
 40 CZ=60
 50 PI=3.141593
 60 K=2*PI/16
 80 P0KE522,1
 100 FORX=-15TO15STEP,5
 110 FORY=-10TO10STEP.03125
 115 GOSUB800
 116 IFX1>255THENX1=255
 200 MOVY1,X1
 202 Y=Y+,5
 204 GOSUB800
 205 IFX1>255THENX1=255
 207 DRAWY1,X1
 220 NEXT:NEXT
 222 END
 700 REM:
 750 REM:  RAČUNANJE  TOČAKA
 760 REM:
 800 L=SIN(K*X)
 850 M=COS(K*Y)
 860 U=EXP(L+M)
 870 V=1+X/CZ
 880 XP=(U*50+50)/V
 890 X1=XP*.5
 900 YP=Y+15/V
 910 Y1=YP*8
 950 RETURN
 955 REM:
 956 REM:  STROJNI PROGRAM ZA INVERZIJU
 957 REM:            EKRANA
 960 DATA 169,96,162,127,160,0
 962 DATA 133,225,132,224,169,255
 964 DATA 145,224,200,208,251,230
 966 DATA 225,228,225,176,245,96


B*********************
*B*
*B* INVERZIJA EKRANA
*B*
*B*********************
*B
*X03000317
 0300 A9 60         LDA *60
 0302 A2 7F         LDX *7F
 0304 A0 00         LDY *00
 0306 85 E1         STA  E1
 0308 84 EO         STY  EO
 030A A9 FF         LDA *FF
 030C 91 EO         STA <EO),Y
 030E C8            INY
 030F DO FB         BNE  030C
 0311 E6 E1         INC  E1
 0313 E4 E1         CPX  E1
 0315 BO F5         BCS  030C
 0317 60            RTS
```



z=exp(sinx+siny)

2.12 GREŠKE PRILIKOM PROGRAMIRANJA

Prilikom pisanja izvornog programa, kao i tokom izvođenja, moguće je pojavljivanje grešaka. BASIC kao interaktivni jezik, odmah će registrirati pogrešku i ispisati njezin kod, te broj linije u kojoj se greška pojavila. Prilikom rada na mikroračunalu ORAO, mogu se pojaviti greške slijedećih kodova:

KOD	**DEFINICIJA**
NF	NEXT bez prethodnog FOR
SN	sintaktička greška
RG	RETURN bez prethodnog GOSUB
NP	nema podataka, više READ nego DATA
FP	greška kod pozivanja funkcije (vrijednosti izvan opsega)
RO	rezultat matematičke operacije je izvan opsega
MP	memorija je puna
NN	nedefinirana naredba ili pokušaj skoka na nepostojeću liniju
PP	pogrešan poziv, nije definiran element polja
DD	dva puta dimenzionirana matrica, argument izvan opsega
/0	pokušaj dijeljenja s nulom
PN	pogrešna naredba, nije dozvoljen izravan način
PT	pogrešan tip podataka
DN	duljina stringa je prevelika, max. 255 znakova
IK	suviše složen izraz
ND	nije definiran povratak, CONT bez prethodnog STOP ili [CTL][C]
DF	nije definirana funkcija

POGLAVLJE 3

3.1 ORGANIZACIJA MEMORIJSKIH LOKACIJA (memorijska mapa)

Za lakše razumijevanje MONITORA i instrukcija koje su uz njega vezane, moramo poznavati memorijsku mapu, koja nam pokazuje kakva je organiziranost računala s obzirom na korištene memorijske lokacije. Na slici 3.1 vidi se memorijska mapa mikroračunala "ORAO".

VIDEO RAM	7FFF 6000
KORISNIČKI RAM 23K	5FFF 0400
NULTI BLOK	03FF 0000

SISTEMSKI ROM	FFFF E000
BASIC – GR ROM	DFFF C000
DOS	BFFF B000
EKSTENZIJA	AFFF A000
SISTEMSKE LOKACIJE "ORLA"	9FFF 8000

Mikroprocesor 6502 može adresirati 65535+1 memorijskih lokacija ili FFFF+1 u heksadecimalnoj notaciji.
Nulti blok (0000-03FF) je predviđen za korištenje u sklopu sistemske memorije računala, koja je rezervirana za omogućavanje rada mikroprocesora 6502 i BASIC-a. Prilikom rada u BASIC-u svi programi počinju iznad nultog bloka.
Od 0400-7FFF nalazi se korisnička memorija (23 k) u koju možemo smjestiti BASIC program. Osnovna verzija mikroračunala "ORAO" sadržava 8k korisničke memorije, koja se vrlo lako može proširiti na 23k. Za proširenje memorije dovoljno je ubaciti odgovarajuće memorijske čipove (6264 ili ekvivalentne) u za to predviđena mjesta i memorija je proširena. Pri tome moramo paziti na pravilnu orijentiranost čipova da ne uništimo integrirani krug. Da bi se izbjegla nedoumica dovoljno je pogledati usmjerenost već postojećih memorijskih čipova i problem je riješen.
Područje od 6000-7FFF zauzima tzv. video RAM. To je 8k memorije u kojoj se nalazi sadržaj ekrana.
Dio memorije od 8000-9FFF predviđen je za sistemske lokacije mikroračunala "ORAO" namijenjene za komunikaciju s perifernim jedinicama. Slijedećih 8k (C000-DFFF) zauzima BASIC interpreter, a područje od E000-FFFF zauzima sistemski software (MONITOR).

3.2 MONITORSKE NAREDBE

Rad u MONITOR-u predviđen je za korisnike koji će raditi u strojnom kodu neovisno o BASIC-u. U mikroračunalu "ORAO" postoje slijedeće monitorske naredbe:

- miniassembler [A]
- disassembler [X]
- čitanje memorijskog bloka (dump) [E]
- čitanje i mijenjanje memorijske lokacije [M]
- provjera sume [C]
- punjenje memorijskog bloka [F]
- izvršenje strojnog programa [U]
- kopiranje dijela memorije [Q]

Sve adrese koje se odnose na rad u MONITOR-u moraju se unositi u heksadecimalnoj notaciji.

3.2.1 Heksadecimalna notacija

U svakodnevnom životu susrećemo se sa decimalnim označavanjem brojeva. U radu s računalima koristi se uglavnom heksadecimalna notacija, koja se bitno razlikuje od decimalne.

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

Maksimalni broj znamenaka u jednoj "heksadi" je 16 (u decimalnom brojevnom sustavu 10). Najveći broj koji se može pojaviti u decimalnom sustavu je 9 (misli se na jednu dekadu), a u heksa sustavu maksimalni broj u jednoj "heksadi" može biti F (15).
Neki decimalni broj možemo napisati u obliku djelomičnih suma:

123=1*100+2*10+3*1 ili
123=1*10^2+2*10^1+3*10^0

Znak ^ označava potenciju (3^2 je tri na kvadrat). Na sličan način možemo rastaviti neki heksadecimalni broj (baza je 16),

123=1*16^2+2*10^1+3*10^0

Ako izračunamo gornji zbroj dobijemo odnos: 123hexa=291decimalno

Neuobičajenost kod heksadecimalnog brojevnog sustava predstavljaju slova (A,B,C,D,E,F), koja imaju funkciju brojeva. Pokažimo nekoliko jednostavnih primjera zbrajanja i oduzimanja,uspoređujući decimalni i heksadecimalni brojevni sustav.

decimalni sustav	heksadecimalni sustav
12+2=14	12+2=E
100+156=256	100+156=256
100-1=99	100-1=FF
32-16=16	32-16=10

Često se zbog lakšeg razlikovanja uz neki heksadecimalni broj dopise znak $,

$123, $A23, $FF10

3.3 MINIASSEMBLER [A]

Pisanje programa u strojnom kodu omogućeno je u mikroračunalu "ORAO" pomoću "miniassemblera", koji izravno pretvara raspoznatljivi mnemonik u odgovarajući strojni kod. Svakom mnemoniku odgovara strojni ekvivalent u heksadecimalnoj notaciji. Mnemoničko označavanje olakšava rad prilikom strojnog programiranja, jer predstavlja početna slova operacije koja će se izvršiti. Na primjer:
LDA (LoaD Accumulator with memory) -napuni akumulator sa sadržajem memorije
JSR (Jump to SubRoutine) -skok na potprogram
DEX (DEcrement indeX register by one) -smanji indeksni registar X za jedan
Vidimo kako se tvore mnemonici LDA, JSR i DEX. Koristeći postojeći miniassembler napišimo program za zbrajanje dva heksadecimalna broja:

8+7 = ?

u decimalnoj notaciji spomenuti zbroj bi bio 15, dok u heksadecimalnom načinu prikazivanja iznos istog zbroja je F(heksa).

```
*A1000
1000   LDA # 08
1002   STA 0500
1004   LDA # 07
1006   CLC
1007   ADC 0500
1009   STA 0501
1006   RTS
100C   Q
?
* _
```

Pogledajmo kako program radi! U prvom koraku (LDA #08) smo izravno upisali broj 8 (heksa) u akumulator i pospremili (STA 0500) ga u memorijska lokaciju 0500. Zatim smo ponovo napunili sadržaj akumulatora brojem 7 (LDA #07} i prijenos stavili na nulu (CLC). U slijedećem koraku zbrojii smo trenutni sadržaj akumulatora (07) i sadržaj memorijske lokacije 0500. Dobivena suma privremeno se nalazi u akumulatoru, koju u slijedećem koraku pospremimo na lokaciju 0501. Komentari s desne strane unijeti su samo zbog lakšeg razumijevanja rada pojedinih programskih koraka i nemaju nikakvu neposrednu vezu s funkcionalnošću.

Monitorskom naredbom [U], izvedimo program:

```
*U1000
*_
```

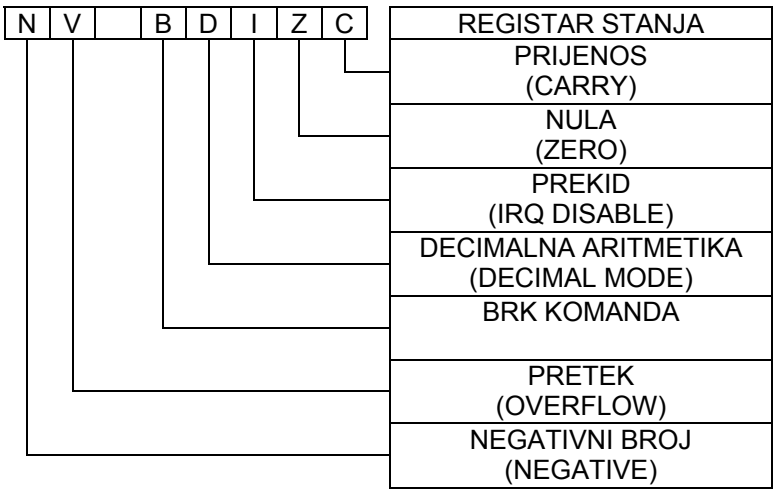
Vidimo da se ništa atraktivno nije dogodilo. Da bi se uvjerili u pravilnost zbrajanja moramo ispitati sadržaj memorijske lokacije 0501 na koju smo pospremili dobiveni zbroj. Čitanje neke memorijske lokacije možemo postići s monitorskom naredbom [M],

```
*M0501
0501    0F
0502    Q
```

Znak "Q" smo iskoristili da bismo kontrolu mikroračunala prebacili u MONITOR. Na taj smo način zbrojili dva 8-bitna broja ne razmišljajući o mogućnosti da njihov zbroj može biti veći od FF (255 dec), što znači pojavljivanje tzv. prijenosa (carry C). Potpuno je ista situacija kod zbrajanja dvaju heksadecimalnih brojeva čiji zbroj prelazi mogućnost upisa u jednu memorijsku lokaciju. Pogledajmo primjer

```
   $80
 + $8F
 ------
  $10F
```

Primjećuje se potreba za još jednim bitom (deveti bit), koji, na žalost, ne postoji niti u jednoj memorijskoj lokaciji. Podatak o prijenosu mora biti raspoloživ da bi se moglo raditi s višebajtnim podacima, pa se postavlja pitanje, kako je podatak o prijenosu dostupan kod mikroprocesora 6502. Problem je riješen poznavanjem jednog specijalnog registra (8-bitni) u mikroprocesoru 6502, koji se zove REGISTAR STANJA, (Processor Status Register). Riječ je o vrlo važnom registru, koji osim podataka o prijenosu (C), sadržava još nekoliko različitih podataka važnih za rad s procesorom. Na slici 3.2 vidi se o kojim je važnim bitovima riječ



Nula (ZERO flag) se indicira kada je rezultat neke operacije nula.
BRK pokazuje da li je zahtjev za prekidom (interrupt request) bio prouzrokovan mnemonikom, instrukcijom prekida (BRK) ili je prekidni zahtjev bio generiran nekom perifernom jedinicom.
Pretek (V) se pojavljuje samo kod označene aritmetike i to u slučaju Kada je rezultat zbrajanja ili oduzimanja veći od +127dec. ili manji od -128dec.
Bit N indicira da li je rezultat aritmetičke operacije negativan.
Aktiviranjem bita (D) način računanja procesora 6502 prelazi iz potpuno binarnog u decimalno binarni.

3.3.1 Zbrajanje sa prijenosom

Vratimo se programu zbrajanja dvaju heksa brojeva čija je suma veća od FF, dakle potrebno je registrirati i prisutnost
prijenosa.

```
*B*   ZBRAJANJE DVAJU HEKSA BROJEVA

*A1000
 1000 LDA #0A      PUNJENJE AKUMULATORA SA OA (LINE FEED)
 1002 JSR E762     SKOK U NOVI RED
 1005 CLC          BRISANJE PRIJENOSA
 1006 LDA #80      PUNJENJE AKUMULATORA SA 80
 1008 STA 0500     POSPREMANJE BROJA $80 NA 0500
 100B LDA #8F      PUNJENJE AKUMULATORA SA DRUGIM BROJEM $8F
 100D ADC 0500     ZBRAJANJE AKUMULATORA SA MEM.LOKACIJOM 0500
 1010 STA 0501     POSPREMANJE ZBROJA NA 0501
 1013 ROL A        ROTIRANJE SADRŽAJA AKUMULATORA
 1014 AND #01      MASKIRANJE SADRŽAJA AKUMULATORA
 1016 JSR E803     ISPIS PRIJENOSA NA EKRANU
 1019 LDA 0501     PUNJENJE AKUMULATORA SA ZBROJEM
 101L JSR E803     ISPIS ZBROJA NA EKRANU
 101F RTS          POVRATAK U MONITOR
 1020Q
```

```
*DIS ASSEMBLERSKI LISTING

*X10001020
 1000 A9 0A            LDA #0A
 1002 20 62 E7 JSR E762
 1005 18                      CLC
 1006 A9 80            LDA #80
 1008 8D 00 05 STA 0500
 100B A9 8F            LDA #8F
 100D 6D 00 05 ADC 0500
 1010 8D 01 05 STA 0501
 1013 2A                      ROL A
 1014 29 01            AND #01
 1016 20 03 E8 JSR E803
 1019 AD 01 05 LDA 0501
 101L 20 03 E8 JSR E803
 101F 60                      RTS
```

Pokretanjem programa dobijemo ispis zbroja zadanih brojeva.
Možemo unositi brojeve po želji.

```
*U1000
 010F
```

Komentari s desne strane služe samo za lakše razumijevanje rada pojedinih koraka u programu i ne moraju se unositi u računalo.
Poslije unosa nekog programa dobro je provjeriti pravilnost unesenih mnemoničkih kodova. To jednostavno postižemo disassemblerskim listingom u granicama zadanog memorijskog bloka.
Zanimljivost kod disassemblerskog listinga je u tome, što se uz mnemonički kod ispisuje i strojni kod.

3.3.2 Ispisivanje teksta

Prilikom rada s računalom često se pojavljuje potreba za ispisivanjem teksta. Na jednostavnom programu vidjet ćemo kako se to može izvesti:

```
*B*****************************
*
*    ISPISIVAJE TEKSTA        *
*
*B*****************************
*
*
*A1000
 1000 LDX FF           PUNJENJE X-REGISTRA SA $FF
 1002 INX              POVEĆANJE SADRŽAJA X-REG, ZA JEDAN
 1003 LDA 1100,X       PUNJENJE AKUM. SA SADRŽAJEM TABELE NA $1100
 1006 JSR FFF1 SKOK NA ISPISNU RUTINU
 1009 CMF #00          PROVJERA *END MARKERA"
 100B BNE 1002 AKO NIJE KRAJ TABELE GRANAJ NA POČETAK
 100D RTS              PRONAĐEN JE  "END MARKER" #00 I ISPIS TABELE
                              JE ZAVRŠEN
 100EQQ               SKOK U MONITOR
?
*
```

Da bi program bio kompletan moramo računalu definirati KOJE znakove želimo ispisati, dakle,moramo tvoriti TABELU u memoriji koja će se gornjim programom moći pozvati i izvršiti. Sadržaj tabele može biti bilo kakav, jasno tekst mora biti unesen u računalo na način razumljiv procesoru. Svaki znak mora biti unesen korištenjem ASCII notacije po kojoj je npr. slovo A zapisano kao $41. Našu tabelu tvorit ćemo koristeći monitorsku naredbu [M]. Unesimo slijedeće bajtove:

```
*B************************
*
*B*  TABELA ZA ISPIS
*
*B************************
*
*M1100
 1100 24 0A
 1101 24 0D
 1102 24 4D
 1103 24 69
```

```
 1104 24 6B
 1105 24 72
 1106 24 6F
 1107 24 72
 1108 24 61
 1109 24 63
 110A 24 75
 110B 24 6E
 110C 24 61
 110D 24 6C
 110E 24 6F
 110F 24 20
 1110 24 22
 1111 24 4F
 1112 24 52
 1113 24 41
 1114 24 4F
 1115 24 22
 1116 24 00      END MARKER'
 1117 24 Q       POVRATAK  U MONITOR
```

XY su neki slučajno postavljeni sadržaji u memoriji koje zamijenimo sa predloženim znakovima na desnoj strani. Program izvodimo sa monitorskom naredbom [U].

```
*U1000
```

Svi znakovi do 00 bit će ispisani na ekranu. Znak 00 (end marker) označuje završetak tabele.

3.4 DISASSEMBLER [X]

Monitorska naredba [X] pretvara raspoznatljivi strojni kod u odgovarajući mnemonik (obrnuti rad od miniassemblera). Upotrebljivost disassemblera je vrlo velika. Ako nas zanima dio sistemskog software-a dovoljno je da unesemo početnu i konačnu adresu bloka memorije koji želimo disassemblirati. Za primjer disassemblirajmo rutinu u BASIC-u koja se nalazi na lokaciji DD11:

```
*X DD11 DD30
 DD11 A2 FF         LDX #FF
 DD13 86 88         SIX  88
 DD15 9A            TXS
 DD16 A9 11         LDA #11
 DD18 A0 DD         LDY #DD
 DD1A 85 01         STA  01
 DD1C 84 02         STY  02
 DD1E 85 04         STA  04
 DD20 84 05         STY  05
 DD22 A9 05         LDA #05
 DD24 AO CE         LDY #CE
 DD26 85 06         STA  06
 DD2W 84 07         STY  07
 DD2A A9 C1         LDA #C1
 DD2C A0 CF         LDY #CF
 DD2E 85 08         STA  08
 BD30 84 09         STY  09
*
```

Nadalje, pomoću naredbe [X] možemo provjeriti pravilnost prethodno unesenog programa u miniassembleru. Disassembliranje po blokovima (28 linija dis. listinga) postižemo unosom samo početna adrese od koje se disassemblira.

Nastavljanje se postiže jednostavnim unosom X koji automatski nastavlja disassembliranje.

Na primjer:

```
*XDD11
 DD11 A2 FF         LDX #FF
 DD13 86 88         STX  88
 DD15 94            TXS
 DD16 A9 11         LDA #11
 DD18 A0 D0         LDY #DD
 DD1A 85 01         STA  01
 DD1C 84 02         STY  02
 DD1E 85 04         STA  04
 DD20 84 05         STY  05
 DD22 A9 05         LDA #05
```

```
 DD24 A0 CE         LDY #CE
 DD26 85 06         STA  06
 DD28 84 07         STY  07
 DD2A A9 Cl         LDA #C1
 DD2C A0 CF         LDY #CF
 DD2E 85 08         STA  08
 DD30 84 09         STY  09
 DD32 A9 4C         LDA #4C
 DD34 85 00         STA  00
 DD36 85 03         STA  03
 DD38 85 Al         STA  Al
 DD3A 85 0A         STA  0A
 DU3C A9 88         LDA #88
 DD3E A0 CE         LDY #CE
 DD40 85 0B         STA  0B
 DD42 84 0C         STY  0C
 DD44 A9 3F         LDA #3f
 DD46 85 0F         STA  0F
 *X
 DD48 A9 38         LDA *38
 DD4A 85 10         STA  10
```

3.5 ČITANJE MEMORIJSKOG BLOKA [E]

Ponekad želimo ispis sadržaja memorije u određenom bloku. To nam omogućava monitorska naredba [E]. Ispis je u heksadecimalnom obliku, a za primjer uzmimo:

```
*
*E C000 C100
 C000 39 C6 55 C5 3F CA 0B C7
 C008 22 C9 00 CD 4E C9 98 DE
 C010 B8 C6 90 C6 3B C7 19 C6
 C018 9B CA E5 C6 4E C7 37 C6
 C020 5E C7 80 DE 8L DE 26 F2
 C028 F4 FO DD CF 28 D4 2E C8
 C030 60 C6 B4 C4 8B C6 60 C4
 C038 D8 D7 62 B8 F5 D7 OA 00
 C040 AD CF CE CF AC DA CO DB
 C048 BD D5 IB DB FC UB 03 BC
 C050 4C DC 99 DC IE D4 8C D3
 C058 BC D0 BD D3 9B D3 FC D2
 C060 10 D3 3C D3 47 D3 79 6E
 C068 D4 79 57 D4 7B FD D5 7B
 C070 CC D6 7F B5 DA 50 68 CC
 C078 46 65 CC 7D EE DA 5A D7
 C080 CB 64 95 CC 45 4E C4 46
 C088 4F D2 4B 45 58 D4 44 41
 C090 54 C1 49 4E 50 55 D4 44
```

3.6 ČITANJE I MIJENJANJE MEMORIJSKE LOKACIJE [M]

U jednom od prethodnih primjera susreli smo se s monitorskom naredbom [M]. Pomoću te naredbe možemo u bilo kojem trenutku čitati i mijenjati sadržaj neke memorijske lokacije u korisničkoj memoriji.

```
*M 1000 A3 CD
   1001 EF itd
```

Na gornjem se primjeru vidi način mijenjanja memorijske lokacije $1000, čiji je sadržaj (recimo) bio $A8, a mi smo ga željeli promijeniti na $CD, Pritiskom na [CR] ispisuje se sadržaj slijedeće memorijske lokacije bez bilo kakvih promjena. Znakom "-" možemo ispisivanje pokrenuti u suprotnom smjeru (smanjivanje adrese). Izlazak iz naredbe [M] jednostavno postižemo upisivanjem nekog slova koji nije element heksadecimalnih brojeva, npr: Unesimo slovo Q:

```
*M 1000 CD Q
*_
```

i kontrola računala opet je u monitoru.

3.7 PROVJERA SUME [C]

Naredbom [C] možemo provjeriti kolika je zajednička suma određenog memorijskog bloka. Za primjer možemo izračunati kolika je suma svih bajtova u našem BASIC-u (8k C000 DFFF):

```
C C000 DFFF
=10 D8 7F
```

3.8 PUNJENJE MEMORIJSKOG BLOKA [F]

Želimo li invertirati ekran potrebno ga je napuniti sa $FF.

Unesimo:

```
*F 6000 7FFF FF
```

Prilikom rada u monitoru ponekad je potrebno koristiti naredbu [F], posebno kada želimo neki memorijski blok brzo napuniti nekim bajtom.

3.9 IZVRŠENJE STROJNOG PROGRAMA [U]

Poslije unosa strojnog programa (miniassembler) potrebno je taj program izvesti. Naredbom [U] postavlja se programsko brojilo mikroprocesora na željenu memorijska lokaciju (početak našeg programa) i izvođenje počinje.
Upotrebom [U] mogu startati već postojeći sistemski programi, npr.:

```
*U E762
```

će ispisati neki slučajni sadržaj akumulatora na ekran.

3.10 KOPIRANJE DIJELA MEMORIJE [Q]

Za kopiranje memorijskog bloka služi naredba [Q]. Želimo li prebaciti dio BASIC-a iz EPROM-a u korisničku memoriju (RAM), dovoljno je upisati:

```
*Q 1000 DD11 DFFF
*
```

Prvi dvobajtni heksadecimalni broj predstavlja adresu memorije na koju će se neki blok prebaciti. Druga i treća adresa predstavljaju početnu i završnu adresu bloka koji se prebacuje. Da bi se uvjerili u korektnost rada naredbe [Q], možemo se poslužiti naredbom [X] ili [C]. Disassemblirajmo dio "novog" sistemskog programa:

```
*
*
*X 1000 1020
1000 62 FF            LDX #FF
1002 86 88            STX 88
1004 96               TXS
1005 69 11            LDA #11
1007 60 DD            LDY #DD
100? 35 01            STA 01
100B 84 02            STY 02
100R 85 04            STA 04
100F 84 05            STY 05
1011 69 05            LDA #05
1013 60 CE            LDY #CE
1015 85 06            STA 06
1017 84 07            STY 07
1019 69 C1            LDA #C1
101B 60 CF            LDY #CF
101B 85 08            STA 08
101F 84 09            STY 09
*
```

Slično ispitajmo za dio sistemskog software-a:

```
*
*X  DD11  DD31
DD11  A2 FF           LDX #FF
DD13  86 88           STX  88
DD15  9A                 TXS
DD16  A9 11           LDA #11
DD18  A0 DD           LDY #DD
DD1A  85 01           STA  01
DD1C  84 02           STY  02
DD1E  85 04           STA  04
DD20  84 05           STY  05
DD22  A9 05           LDA #05
DD24  A0 CE           LDY #CE
DD26  85 06           STA  06
DD28  84 07           STY  07
DD2A  A9 C1           LDA #C1
DD2C  A0 CF           LDY #CF
DD2E  85 08           STA  08
DD30  84 09           STY  09
*
```

Primjećujemo da su listinzi gotovo identični. Razlika naizgled postoji samo u relativnom adresiranja (BNE, BEQ, BVC,...).
Drugi način provjere koristi monitorsku naredbu za provjeru sume [C]. Izračunajmo sumu:

```
*C 1000 1020
= 00 0E C3
```

Sličan postupak ponovimo za "originalni" memorijski blok:

```
*C DD11 DD31
= 00 0E C3
```

Sume oba bloka su identične, Što je dokaz da je transfer bio potpune korektno izvršen.

3.11 OPIS VAŽNIJIH POTPROGRAMA U MIKRORAČUNALU "ORAO"

Postepenim upoznavanjem strojnog programiranja doći ćemo u poziciju da pišemo vlastite strojne programe određene namjene. Međutim, velika je vjerojatnost da u mikroračunalu "ORAO" već postoje potprogrami koji će biti potrebni u sklopu rješavanja; NAŠE problematike. Potpuno je jasno, da bi ponovno pisanje već postojećih programa bilo suvišno, pa će zbog toga u ovom dijelu priručnika biti navedeni neki najčešće korišteni potprogrami (subroutine), koji se uglavnom odnose na komuniciranje korisnika s računalom.

3.11.1 Potprogram za unos znaka s tastature (INCH)

Kada želimo unijeti neki znak s tastature (bez ispisa na ekranu) koristit ćemo INCH (input character) potprogram koji se nalazi na $E500. Upotrebljivost INCH je vrlo velika kada je izvođenje programa vezano s dinamičkim mijenjanjem parametara koje korisnik želi unijeti izravno (npr. kod pisanja igara). Opisat ćemo mali demo program, koji će ispitivati da li je pritisnuta tipka "A" ili ne. Za bilo koji znak (osim "A") na ekranu će se ispisati znak "@" i kontrola računala bit će vezana za naš strojni program. Pritiskom na tipku "A* naš program se prekida i kontrola računala se vraća u MONITOR.

```
*B***************************
*
*B*  UNOS ZNAKA S TASTATURE  *
*
*B***************************
*
*A1000
 1000 JSR E500  POTPROGRAM ZA ISPITIVANJE TASTATURE
 1003 LDA FC
 1005 CMP #41   DA LI JE PRITISNUTI TASTER  "A"
 1007 BEQ 1011  AKO JE  "A" IZLAZIMO IZ PROGRAMA
 1009 LDA #40   PUNIMO  AKUM. SA  "@"
 100B JSR E762  ISPIS  "@"
 100E JMP 1000  SKOK NA POČETAK  PROGRAMA
 1011 CLC       BRIJANJE PRIJENOSA
 1012 RTS       POVRATAK U MONITOR
 1013Q
```

3.11.2 Unos znaka s prikazom na ekranu (INCHE)

Potprogram INCHE (input character with echo) je gotovo isti prethodno opisanom. Razlika je samo u dodatnom prikazivanju unesenog znaka s tastature na ekran. Adresa INCHE je $B71C.

```
*
*A1000
1000 JSR E71C
1003 RTS
1004Q
```

3.11.3 Ispis znaka na ekran (OUTCH)

Zadatak potprograma OUTCH (output character to the screen) je ispisivanje sadržaja akumulatora na ekran. Primjenljivost OUTCH je praktički svugdje gdje se želi ispisati neki alfanumerički znak. U nekoliko dosadašnjih primjera već smo se susreli s upotrebom potprograma OUTCH,čija se adresa nalazi na $E762, odnosno na $FFF1.

3.11.4 Skok u MONITOR bez RESET sekvencije

Ponekad se može pojaviti potreba za miješanjem BASIC programa sa strojnim programom. U tom slučaju, korištenjem USR naredbe, prebacujemo kontrolu računala iz BASIC-a u MONITOR. Adresa spomenutog potprograma je $FFA7 (ili. u decimalnoj notaciji 255 (FF) i 167 (A7)). Formiranje adrese postiže se naredbom POKE. Slijedeći program prikazuje kako se to radi.

```
10 POKE 11,167
20 POKE 12,255
30 U=USR(U)
RUN
*_
```

Vidimo da smo programski promijenili kontrolu računala

3.11.5 Ispisivanje stringa na ekran (STROUT)

Često se prilikom pisanja tzv. sistemskog software-a pojavljuje potreba za ispisivanjem tekstualnog bloka. U mikroračunalu "ORAO" postoji potprogram koji to relativno lako omogućava. Potrebno je unijeti adresu tabele teksta koji će se ispisati, a pravilan način korištenja pokazat će slijedeći program.

```
*A1000
1000 LDY $00          NIŽI DIO ADRESE TABELE TEKSTA
1002 LDA $11          VIŠI DIO ADRESE TABELE TEKSTA
1004 JSR E63B         SKOK NA STROUT POTPROGRAM
1007 RTS
```

Tabelu teksta organiziramo pomoću [M], uz imperativ da na kraju tabele MORA biti upisan znak $04.

3.11.6 Ulazak u BASIC iz MONITOR-a

Kontrolu računala možemo prebaciti iz MONITOR-a u BASIC jednostavno koristeći naredbu [U] i adresu za ulazak u BASIC.

```
*U DD11          hladni start
MEMORIJA ?
```

itd

Kada već imamo neki BASIC program pohranjen u memoriji* adresa uz [U] je $C274.

```
*U C274          "vrući" start
```

3.11.7 Generiranje zvuka

Mikroračunalo "ORAO" ima ugrađeni zvučnik koji, u povezanosti s računalom, može generirati vrlo atraktivne zvukove. Slijedeći primjer pokazuje kako se to može lako postići. Nakon pravilno unesenog programa i pokrenutog s instrukcijom [U] iz zvučnika se mora čuti zvuk promjenljive frekvencije. Zaustavljanje se postiže

pritiskom na tipku [A]. Ovaj primjer je zanimljiv utoliko, što pokazuje kako se postiže kontrola izvođenja preko tastature.

```
*B*****************************
*B*                            *
*B*   GENERIRANJE ZVUKA        *
*B*                            *
*B*****************************
*
*A1000
 1000 LDX #FF          PUNJENJE X-REGISTRA SA $FF
 1002 LDY #FF          PUNJENJE Y-REGISTRA SA $FF
 1004 TYA              PRIJENOS SADRŽAJA Y-REG. U AKUMULATOR
 1005 STY 8800 ADRESA ZVUČNIKA
 1008 DEY
 1009 BNE 1008
 100B TAY              PRIJENOS SADRŽAJA AKUM. U Y-REGISTAR
 100C DEX
 100D DEY
 100E BNE 1004
 1010 JSR E5B0 SKOK NA POTPROGRAM ZA ISPITIVANJE
                            PRITISNUTOSTI TASTERA
 1013 BCC 1000 C=0 TASTER NIJEPRITISNUT
 1015 JSR E500 DA LI JE PRITISNUTO SLOVO  'A"
 1018 CMP #41
 101A BNE 1000 A NIJE PRITISNUTO; PROGRAM SE PONAVLJA
 101C CLC
 101D RTS              POSLIJE PRITISKA NA  "A" PREKIDAMO
                            PROGRAM ZA ZVUK
 101EQQQ
?
*
```

3.11.8 Crtanje točke iz MONITOR-a (PLOT)

U prethodnom poglavlju je bio opisan način korištenja grafičke naredbe u BASIC-u. Prednost korištenja naredbe PLOT bila je u lakom korištenju, međutim, izvođenje je bilo relativno sporo. Povećanje brzine se postiže pozivanjem potprograma PLOT iz MONITOR-a. Adresa potprograma za crtanje točke je $FE69 što, međutim, nije dovoljno. Potrebno je definirati poziciju na kojoj se točka želi vidjeti. U tu svrhu se koriste memorijske lokacije $E2 i $E3. U gornjem desnom kutu biti će nacrtana točka, ako su sadržaji $E2 i $E3 postavljeni na $FF.

```
*M00E2 XY FF
 M00E3 XY FF

*UFE69
```

Koordinata X se upisuje na memorijsku lokaciju $E2, a Y koordinata na lokaciju $E3. Jednostavnim skokom na $FE69 nacrta se točka. Slijedećim programom nacrtat ćemo dijagonalu na ekranu.

```
*A1000
1000 LDA #FF
1002 STA E2
1004 STA E3
1006 JSR FE69
1009 DEC E2
100B DEC E3
100D BNE 1006
100F RTS .
```

Želimo li prikazati svaku drugu točku, moramo još jednom umanjiti $E2 i $E3.

3.11.9 Izvlačenje linije (DRAW)

Potprogram za izvlačenje linije počinje na memorijskoj lokaciji $FE8B. Za izvlačenje linije potrebno je odrediti koordinate početka i koordinate kraja, dakle, potrebne su četiri memorijske lokacije za potpunu definiranost linije. Početne koordinate pospremaju se na $E2 i $E3, a završne na $E4 i $E5. Obrnutu dijagonalu dobit ćemo slijedećim postupkom.

```
*M00E2
00E2 XY FF
00E3 XY 00
00E4 XY 00
00E5 XY FF
```

```
*U FE8B
```

3.11.10 Crtanje kružnice

Kod crtanja kružnice potrebno je unijeti tri parametra:

- X-koordinatu upisujemo na $E2
- Y-koordinatu upisujemo na $E3
- polumjer upisujemo na $F8

Izvršenje se postiže skokom na $FF06.

```
*U FF06
```

3.11.11 Adrese važnijih potprograma u mikroračunalu "ORAO"

$E000 adresa grafičkog karakter generatora
$E3AF form feed
$E3C5 back space
$E3CF vertical tab
$E42B horizontal tab
$E435 line feed
$E47E carriage return
$E4F4 bell
$E500 ispitivanje koji je taster pritisnut
$E5B0 ispitivanje da li je taster pritisnut
$E906 adresa [U]
$E90E adresa [E]
$EAD0 adresa [C]
$EB09 adresa [Q]
$EB48 adresa [F]
$EB6A adresa [M]
$EBAB adresa [A]
$EC80 adresa [X]
$BDDC pospremanje bajta na traku
$EE12 prijem bajta s trake

3.12 RAD S KAZETOFONOM U MONITOR-u

3.12.1 Pospremanje strojnih programa SAVE

Mikroračunalo "ORAO" ima sposobnost pospremanja bilo kojeg dijela memorijske mape na traku. Ova sposobnost bit će od izuzetne koristi svima koji će se baviti programiranjem u assembleru. Naime, dovoljno je snimiti memorijski blok u kojemu se nalazi strojni kod našeg programa. Kasnije, nakon učitavanja, disassembliramo strojni kod, i program je tu. Prilikom rukovanja sa strojnim programima potrebno je uz ime programa navesti i adresu memorijskog bloka koji pospremamo. Postupak pospremanja strojnih programa na traku je slijedeći:

- poslije uključivanja mikroračunala prebacujemo kontrolu u BASIC s *BC (to je potrebno samo prilikom uključivanja na mrežu zbog pravilne inicijalizacije računala)
- iz BASIC-a se vraćamo u MONITOR jednostavnim pritiskom na RESET tipku
- na određeni memorijski blok unesemo naš strojni program
- s *BW se vratimo u BASIC
- koristimo naredbu SAVE

```
SAVE"STR.PROGRAM"1000,1100
```

Memorijski blok u ovom primjeru bio je od $1000 do $1100. Prilikom programiranja u BASIC-u nije bilo potrebno unositi adrese memorijskog bloka, jer se adrese postavljaju automatski.

3.12.2 Pozivanje strojnih programa LOAD

Pozivanje strojnih programa izvođ. Se, također, iz BASIC-a, s tom razlikom da je potrebno navesti samo

početnu memorijsku adresu na koju se program unosi. Vrijednost te adrese može biti ista kao i kod pospremanja ili neka druga. Navedena sposobnost omogućava premještanje određenog memorijskog bloka unutar korisničkog dijela memorije, tzv. relociranje.
Operativni dio pozivanja je sličan kao i kod pozivanja običnih BASIC programa, jasno, uz potrebno dodavanje startne adrese.
Nedostatak upisa početne adrese rezultirat će pospremanjem strojnog programa na lokaciju 0400.

```
LOAD"STR.PROGRAM"1000
```

Maksimalni broj alfanumeričkih znakova, predviđenih u imenu programa, može biti 12 u bilo kojem rasporedu. Povratak u MONITOR izvodimo ili programski ili pritiskom na tipku RESET.

3.12.3 Katalogiranje strojnih programa LOADC

Katalogiranje trake je potpuno isto kao i kod BASIC programa.

```
LOADC
```

Razlika će biti samo u adresama korištenih blokova. Korištenjem naredbe LOADC možemo dobiti podatke o adresama snimljenih strojnih programa,što u nekim momentima može biti od izuzetne koristi.

POGLAVLJE 4

4.1 OPIS MNEMONIKA ZA MIKROPROCESOR 6502

U ovom poglavlju je opisan kompletan set mnemoničkih kodova koji se koriste prilikom strojnog programiranja mikroprocesora 6502. Već na samom početku bilo je napomenuto da ovaj priručnik nije oblikovan kao udžbenik, već samo kao ispomoć prilikom programiranja. Mnogo detaljnija obrada problematike strojnog programiranja opisana je u specijaliziranim udžbenicima za assemblersko programiranje 6502.
Svaka mnemonička instrukcija bit će kratko opisana i simbolički prikazana,uz opis registra stanja i broja ciklusa potrebnih za njeno izvršenje.
Jedna od najvažnijih prednosti mikroprocesora 6502 leži u njegovim načinima adresiranja (13 načina), koji mu omogućavaju rješavanje zadane problematike lako i brzo. Uspoređujući ostale,često korištene mikroprocesore, vidimo da 6502 ima šest adresnih načina više nego Motorola 6800, osam više od Intel-a 8080 ili 8085, tri više nego Zilog Z80 i pet više od Texas Instruments-a 9900.

4.2 NAČINI ADRESIRANJA ZA 6502

Zbog lakšeg razumijevanja i najboljeg korištenja načina adresiranja za 6502 u ovom dijelu ćemo kratko opisati svaki adresni način uz prikaz pravilne sintakse. Napominjemo, da se heksadecimalni brojevi unose sa znakom dolara "$", zbog razlikovanja od decimalnih brojeva. Označavanje sa "$" nije potrebno kod mikroračunala "ORAO", jer on u MONITOR-skom radu prihvaća isključivo heksadecimalnu notaciju bez dodatnog prefiksa.

4.2.1 Izravno adresiranje (Immediate Addressing)

Ovaj način adresiranja omogućava izravni rad s 8-bitnim brojevima. "Izravni" operand mora biti proslijeđen sa znakom # (hash). Takav način omogućava rad s konstantama, kao kada, npr., želimo u akumulator zapisati vrijednost $A1(161 dec.)

```
LDA #$A1
```

Izravno adresiranje zauzima dva bajta memorijske mape, što se uostalom može uočiti na primjerima iz prethodnog poglavlja.

4.2.2 Akumulatorsko adresiranje (Accumulator Addressing)

Mikroprocesor 6502 ima četiri instrukcije koje omogućavaju pomicanje ili rotiranje sadržaja akumulatora za jedan bit u lijevo ili desno. Upućivanje na ovaj adresni način postižemo dodavanjem slova "A" uz mnemonik. Za

primjer možemo uzeti rotiranje u desno:

```
ROR A
```

4.2.3 Relativno adresiranje (Relative Addressing)

Efektivna adresa kod relativnog načina adresiranja dobije se pribrajanjem trenutnog stanja programskog brojila s adresnim dodatkom (offset-om). Adresni dodatak može biti pozitivan ili negativan, što omogućuje kontrolu grananja programskog brojila za +127 ili -128 bajta.
Relativno adresiranje se odnosi na uvjetne instrukcije grananja, a to znači izvršenje samo ako je zadovoljen odgovarajući uvjet. Za primjer uzmimo:

```
1000 CMP       #4F
1002 BNE       1010
1004 RTS
1010 JSR       E762
```

Priloženi primjer opisuje slučaj grananja (pozitivni "offset"), ako je sadržaj akumulatora različit od #$4F. Kod sadržaja akumulatora od #$4F neće doći do grananja, odnosno instrukcija BNE (Branch on Result Not Equal to Zero) se jednostavno preskače.
Sve "branch" instrukcije zauzimaju dva bajta u memoriji s time da drugi bajt određuje veličinu pozitivnog ili negativnog dodatka programskom brojilu.

4.2.4 Apsolutno adresiranje (Absolute Addressing)

Apsolutno adresiranje omogućava direktno adresiranje bilo koje memorijske lokacije mikroprocesora 6502, a to znači unutar granica od 0 do 65535 memorijskih lokacija. Sve instrukcije kod apsolutnog načina adresiranja zahtijevaju tri bajta; prvi bajt predstavlja strojni ekvivalent mnemoničke kode, dok su drugi i treći bajt predviđeni za upis adrese. Napunimo akumulator sadržajem adrese $ABCD:

```
LDA $ABCD
```

Ovdje treba napomenuti, da se sa stanovišta memorijske mape, prvo upisuje strojni kod (u našem slučaju $AD), a zatim manje značajan bajt ($CD) i na kraju više značajni bajt ($AB) efektivne adrese.

4.2.5 Apsolutno indeksno adresiranje (Absolute Indexed Addressing)

Ovaj način adresiranja razlikuje se od prethodno opisanog po tome sto se efektivna adresa tvori kao zbroj apsolutne adrese i sadržaja indeksnog registra (X-a ili Y-a).

- Efektivna adresa = apsolutna adresa + X
- Efektivna adresa = apsolutna adresa + Y

Operand kod apsolutno indeksnog adresiranja dobije se dodavanjem vrijednosti X-a ili Y-a apsolutnoj adresi.

```
LDA $ABC0,X
LDA $ABC0,Y
```

Ako je sadržaj X registra 05, a sadržaj Y registra 09, efektivne adrese za navedene primjer bit će

$ABC0+05=$ABC5
$ABC0+09=$ABC9

Dodavanjem indeksa apsolutnom adresiranju dobili smo vrlo važnu kvalitetu prilikom oblikovanja strojnih programa - utoliko više što raspolažemo sa dva indeksna registra, a to nam omogućuje dostup do svake lokacije u memorijskoj mapi 6502.

4.2.6 Adresiranje nulte stranice (Zero Page Addressing)

Adresiranje nulte stranice slično je apsolutnom adresiranju samo što se kod ovog adresiranja efektivna adresa nalazi unutar jednog bajta.

```
LDA $AA
```

Gornji primjer posprema sadržaj memorijske lokacije $AA u akumulator. Međutim, ovu istu operaciju možemo postići koristeći apsolutno adresiranje:

```
LDA $00AA
```

Razlika je u tome što kod adresiranja nulte stranice trošimo dva bajta memorijske mape, a kod apsolutnog adresiranja tri bajta. Naravno, povećani broj bajtova dodatno opterećuje memorijsku mapu, a gubi se i na brzini. Zbog toga se mora voditi računa prilikom strojnog programiranja, jer je procesor 6502 bitno vezan baš uz nultu stranicu memorijske mape.

4.2.7 Uključujuće adresiranje (Implied Addressing)

Ovo adresiranje za razliku od ostalih adresnih načina nema operanda (on je UKLJUČEN već u mnemoniku), a odnosi se na povećanje ili smanjenje X ili Y registra, te postavljanje ili brisanje bitova u registru stanja.

Navedimo nekoliko primjera:

- DEY smanjenje Y registra
- DEX smanjenje X registra
- SEC postavljanje bita prijenosa

Sve instrukcije vezane uz Uključujuće adresiranje zauzimaju jedan bajt memorijske mape.

4.2.8 Indirektno apsolutno adresiranje (Indirect Absolute Addressing)

Apsolutno indirektno adresiranje koristi se kod mikroprocesora 6502 samo za instrukciju indirektnog skoka. Odredišnu adresu dobijemo preko sadržaja dviju apsolutnih adresa, dakle indirektno. Za lakše razumijevanje navedimo primjer indirektno apsolutnog adresiranja iz mikroračunala "ORAO". Opisat ćemo primjer potprograma za ispis znaka na ekran. Adresa potprograma OUTCH je $E762, međutim, u sistemskom software-u primijetiti ćemo da se OUTCH nalazi na $FFF1, dakle, za ispis znaka koristimo JSR $FFF1. Pogledajmo disassemblerski listing:

```
FFF1 6C 18 02 JMP(0218)
```

Vidimo da se ovaj indirektni skok poziva na sadržaje memorijskih lokacija $0218 i $0219.

```
M0218 62
 0219 E7
```

Stvarna adresa OUTCH potprograma sadržana je u memorijskim lokacijama 0218 i 0219. Možda će se neki korisnici iznenaditi zbog čega se taj način pozivanja OUTCH potprograma koristi, kad bi jednostavnije bilo uvijek pisati JSR E762, međutim, ovaj način omogućava tzv. preusmjeravanje vektora na željenu memorijsku lokaciju, što znači dozvoliti korisniku kreiranje vlastitog OUTCH potprograma.

4.2.9 Indeksno adresiranje nulte stranice (Zero Page Indexed Addressing)

Slično kao što smo imali kod uspoređivanja apsolutnog i apsolutno indeksnog adresiranja,možemo i sada uspoređivati indeksno adresiranje nulte stranice s "običnim" adresiranjem nulte stranice. Dakle, razlika postoji samo u veličini adresnog operanda,od tri bajta prešli smo na dvobajtni operand.

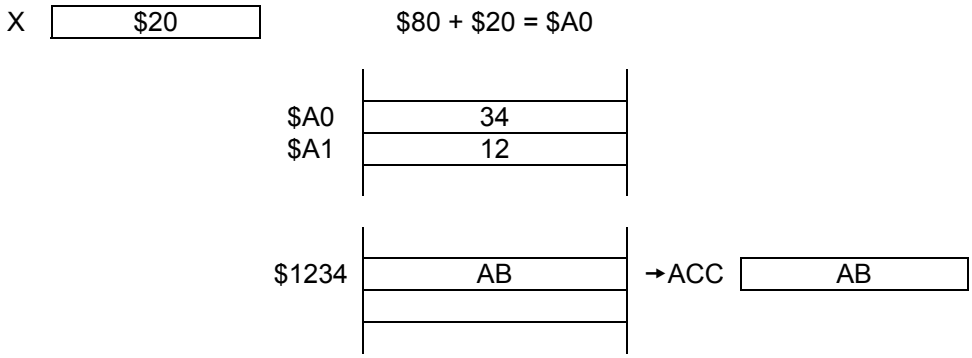
```
LDA $AB,X
LDA $45,Y
```

Ulogu indeksa može na sebe preuzeti X ili Y registar. Odredišna adresa tvori se kao zbroj indeksa i adrese.

4.2.10 Preindeksno adresiranje (Indexed Indirect Addressing)

Ovaj način adresiranja predstavlja kombinaciju indirektnog i indeksnog načina adresiranja. Znamo da kod indeksnog adresiranja efektivnu adresu dobivamo kao zbroj neke osnovne adrese i sadržaja indeksa, a kod indirektnog načina adresiranja operand u instrukciji već predstavlja adresu,čiji sadržaj daje konačnu efektivnu adresu. Kod svih indirekcija koriste se dva bajta u memoriji zbog mogućnosti 16 bitnog adresiranja, dakle, pokrivanja svih 65535 adresnih lokacija. Možda će slijedeći primjer olakšati razumijevanje preindeksnog adresiranja. Uzmimo da je sadržaj indeksnog registra $20, a adresa $80:

```
LDA($80,X)
```



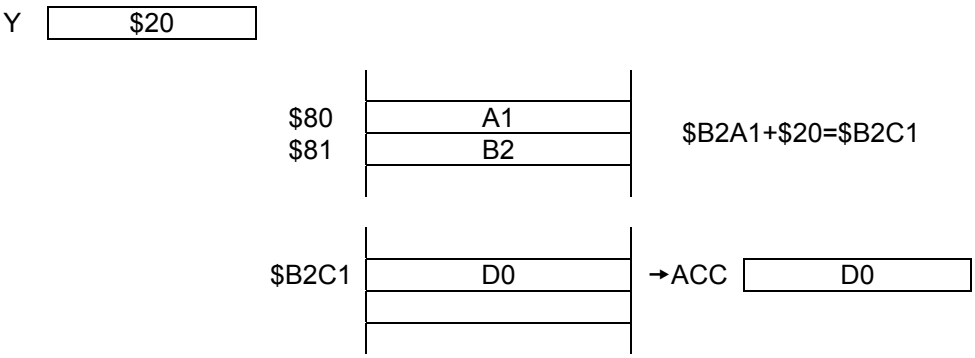
Efektivnu adresu nulte stranice dobijemo zbrojem adrese i indeksa. ($80+$20)=$A0
Manje značajan bajt efektivne adrese dobijemo kao sadržaj $A0, a značajniji bajt kao sadržaj $A1, dakle, konačna efektivna adresa je $1234. Sadržajem $1234 konačno punimo akumulator i nastavljamo program.

4.2.11 Postindeksno adresiranje (Indirect Indexed Addressing)

Postoji izvjesna sličnost između preindeksnog i postindeksnog adresiranja, dakle, u jednom i drugom slučaju kombiniramo indeksno i indirektno adresiranje, a razlika je samo u redoslijedu. Kod prethodnog načina prvo zbrojimo adresu i vrijednost indeksa, pa onda izvršimo indirekciju, a kod postindeksnog prvo načinimo indirekciju a tek onda pribrajamo indeks da bi dobili efektivnu adresu. Cijeli problem pokušajmo olakšati malim primjerom:

```
LDA ($80),Y
```

Y=$20



Iako su vrijednosti naizgled vrlo slične vidimo da je konačna efektivna adresa potpuno drugačija. Preindeksno i Postindeksno adresiranje postaju isto u slučaju da su vrijednosti indeksnih registara nula.

5 NAPOMENA

U nastavku priručnika bili su opisani mnemonici 6502 procesora koji su ovdje izostavljeni jer se slični dokumenti ili tablice vrlo lako mogu naći na Internetu. Sorry! ☺

KRAJ