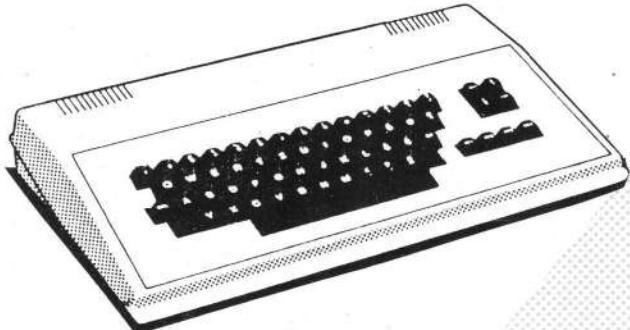


ORAO



PRIRUČNIK

**MIKRORAČUNALO
ORAO 102 i 103**

(EAGLE EXTENDED BASIC)

Autor: ZLATAREK DRAŽEN

PEL PROFESSIONALNA ELEKTRONIKA Varaždin
ORAO KLUB Varaždin

ORAO
PRIRUČNIK

ZLATAREK
DRAŽEN
PEL PROFESSIONALNA
ELEKTRONIKA
Varaždin TIP 102 i 103
za mikroracunalo

ORAO

TIP 102 i 103 - EAGLE EXTENDED BASIC

Stručna recenzija: prof. VLAŠIĆ ZORAN

IZDANJE:

ORAO KLUB Varaždin

PEL Profesionalna elektronika Varaždin

Varaždin, 1989.

NARUČITE BESPLATNO !

PEL® PROFESIONALNA ELEKTRONIKA Varaždin

ORAO KLUB Varaždin



PEL Profesionalna elektronika Varaždin

tel. 042/51-333

ORAO KLUB Varaždin

tel. 042/47-875

UVOD

U ovom prirucniku je opisan postupak povezivanja mikroracunala ORAO s pripadajućim perifernim jedinicama (TV prijemnik, crno-bijeli video monitor, stampac, kazetofon i disketna jedinica), te nacin programiranja u BASICU i miniASSEMBLERU. Prirucnik je organiziran u sest dijelova:

- nacin pocetnog koristenja
- programiranje u BASIC-u
- prikaz mnemonickih instrukcija mikroprocesora 6502
- opis nacina adresiranja mikroprocesora 6502
- opis i koristenje MONITOR-a
- koristenje potprograma iz ROM-a

Za sve korisnike koji jos nisu imali priliku raditi na racunalima, preporuca se pazljivo pracenje uvodnog dijela ovog prirucnika, kako bi se izbjegle nepotrebne neugodnosti, koje su posljedica nepravilnog rukovanja.

Mikroracunalo ORAO uglavnom je namijenjeno za koristenje u BASIC-u, pa ce stoga veci dio prirucnika biti usmjeren na objasnjene BASIC naredbi, uključujući i nacine njihovog koristenja. Za one koje ce zanimati detaljniji rad mikroracunala i mikroprocesora 6502, predvidjen je rad u MONITOR-u s mogucnoscu koristenja programiranja u strojnom jeziku.

Ovaj prirucnik nije zamisljen kao udzbenik BASIC-a ili ASSEMBLER-a, nogo kao pomagalo prilikom savladavanja osnovne problematike rada na mikroracunalu ORAO. Za sve one kojima ce sadržaj ovog prirucnika biti suvise skroman, preporučuje se nabavka specijaliziranih knjiga za ucenje BASIC-a i ASSEMBLERA.

Mikroracunalo ORAO se, jos od kraja 1985. godine, produžovi s ugradjenom poboljsanom i prosirenom verzijom BASIC-a. Ovaj je prirucnik napisan prvenstveno za tu verziju BASIC-a i predstavlja prosirenje i dopunu prirucnika za mikroracunalo ORAO sa starom verzijom BASIC-a. Programi napisani na staroj verziji BASIC-a mogu se koristiti i ako imate prosireni BASIC. Korisnici mikroracunala ORAO koji bi htjeli zamijeniti stari za novi, prosireni BASIC, neka se obrate proizvodjacu cija je adresa:

RO PEL - profesionalna elektronika
Nazorova 2
42000 Varazdin
tel. 042/51-333

SADRZAJ

1. POGлавље 1 - ОСНОВНИ ПОЈМОВИ	1
1.1. САСТАВ МИКРОРАСУНСКОГ СИСТЕМА "ОРАО"	1
1.2. ПРИКЉУЧИВАЊЕ И ПУСТАЊЕ РАСУНАЛА У РАД	1
1.3. ТАСТАТУРА МИКРОРАСУНАЛА "ОРАО"	4
1.4. ШТО ЈЕ "BASIC" ?	6
1.5. ПРОГРАМИРАЊЕ У BASIC-У	7
1.5.1. Директни начин рада	7
1.5.2. Програмски начин рада	8
1.6. КАКО СЕ КОРИСТЕ ТАСТЕРИ ЗА ПОМИКАЊЕ КУРСОРА И КОПИ ТАСТЕР (PF4)	9
1.7. ПРИМЈЕР ПРОГРАМА У СТРОЈНОМ ЈЕЗИКУ	14
1.8. УЧИТАВАЊЕ ПРОГРАМА С КАЗЕТЕ (УВОД)	17
2. ПОГЛАВЉЕ 2 - НАРЕДБЕ ПРОГРАМСКОГ ЈЕЗИКА BASIC И НЈИХОВА УПОТРЕБА	18
2.1. УВОД - МОНИТОРСКЕ НАРЕДБЕ BW и BC	18
2.2. КАКО СЕ ПИСУ ПРОГРАМИ У BASIC-У	19
2.3. ВАРИЈАВЛЕ	19
2.3.1. Бројеви и numericke varijable	19
2.3.2. Stringovi i string varijable	21
2.4. КОНТРОЛНЕ НАРЕДБЕ	22
- Return (Carriage return)	22
- Break	22
- RUN	22
- STOP	22
- CONT	23
- END	23
- NEW	23
- REM	23
- CLS	24
- LIST	24
- EXIT	24
2.5. УЛАЗНО - ИЗЛАЗНЕ НАРЕДБЕ	25
- PRINT	25
- INPUT	28
2.6. ПЕТЛJE И ПОТПРОГРАМИ	30
- IF-THEN	30
- GOTO	32
- FOR-TO-STEP-NEXT	32
- GOSUB-RETURN	36
- ON	38
2.7. ПОДАЦИ У ПРОГРАМУ	38
- READ-DATA	38
- RESTORE	40
- CLEAR	40

SADRZAJ

2.8. ARITMETICKE OPERACIJE	41
2.9. POLJA PODATAKA	44
- DIM	44
2.10. STRING FUNKCIJE	49
- Usporedjivanje stringova	50
- Zbrajanje stringova	50
- ASC	51
- CHR\$	51
- LEN	52
- LEFT\$	53
- RIGHT\$	53
- MID\$	54
- STR\$	55
- VAL	56
2.11. LOGICKE OPERACIJE	56
- AND	56
- OR	57
- NOT	57
2.12. ORGANIZACIJA EKRANA	59
- CUR	61
- TAB	61
- SPC	62
- POS	62
- INV	63
- VDU	64
- SCREEN\$	65
2.13. NUMERICKE FUNKCIJE	66
- ABS	67
- INT	67
- RND	68
- SGN	69
- SQR	70
- LOG	70
- EXP	70
- SIN	71
- COS	71
- TAN	71
- ATN	71
2.14. FUNKCIJE DEFINIRANE OD STRANE KORISNIKA	72
- DEF FN	72
2.15. PISANJE PROGRAMA I PRINCIPI PROGRAMIRANJA	73
2.16. CUVANJE PROGRAMA - RAD S KAZETOFOONOM	80
- Povezivanje MR Drao s kazetofonom	80
- SAVE	82
- LOAD	83
- TGAP	85

2.17. ISPITIVANJE TASTATURE	85
- INKEY	85
2.18. GRAFIKA I ZVUK	87
- MOVE	89
- PLOT	89
- DRAW	90
- CIR	92
- MODE	93
- DOT	94
- Animirana grafika	96
- SOUND	96
- LETTER	98
2.19. RAD S MEMORIJOM	100
- Brojevni sustavi	100
- Decimalni brojevni sustav	101
- Binarni brojevni sustav	101
- Heksadecimalni brojevni sustav	102
- Binarni broj, bit, bajt	104
- Mikroprocesor i memorija	104
- Memorijска карта микрорачунара Орао	105
- FRE	107
- POKE	108
- PEEK	110
- Definiranje vlastitog seta znakova	111
- CHAR	114
- SMOVE	114
- animacija pomocu naredbe SMOVE	116
- Pozivanje strojnih potprograma iz Basic-a ..	121
- LNK	121
- USR	122
- Spremanje strojnog programa na kazetu - DMEM	122
- Ucitavanje strojnog programa s kazete - LMEM	123
2.20. POVEZIVANJE SA STAMPACEM	123
- Nacin koristenja stampaca	124
2.21. RAD S DATOTEKAMA NA KAZETI	126
- Spremanje Basic podataka na kazetu	126
(OPENW, WRITE, CLOSEW)	
- Citanje Basic podataka s kazete	127
(OPENG,CLOSEG)	
2.22. DODATNE NAPOMENE	131
- Tipovi datoteka kod rada s kazetofonom	131
- Primjer koristenja strojnog potprograma iz Basic-a	132
- Ocitavanje PADDLE-a - PDL	134
- Ogranicavanje memorije	134
- Rad s disketnom jedinicom	135

SADRZAJ

2.23. GRESKE PRILIKOM PROGRAMIRANJA	137
3. PROGRAMIRANJE U STROJNOM JEZIKU	138
3.1. STROJNI JEZIK - UCITI ILI NE ?	138
3.2. PRIKAZ MNEMONICKIH INSTRUKCIJA	
ZA MIKROPROCESOR 6502	141
- Programski model 6502	141
- Prikaz mnemonika za 6502	143
3.3. NACINI ADRESIRANJA MIKROPROCESORA 6502	157
- Izravno adresiranje	157
- Akumulatorsko adresiranje	157
- Relativno adresiranje	158
- Apsolutno adresiranje	158
- Adresiranje nulte stranice	159
- Uključujuće adresiranje	160
- Indirektno apsolutno adresiranje	160
- Indeksno adresiranje nulte stranice	160
- Preindeksno adresiranje	161
- Postindeksno adresiranje	162
4. OPIS MONITORA	163
4.1. B - Pozivanje Basica iz Monitora	164
4.2. A - Miniasembler	164
4.3. X - Disasembler	166
4.4. U - Izvršenje strojnog programa	167
4.5. M - Citanje i mijenjanje mem. lokacije	167
4.6. E - Citanje memorijskog bloka	168
4.7. H - Citanje mem. bloka (ASCII)	169
4.8. F - Punjenje mem. bloka konstantom	169
4.9. C - Provjera sume mem. bloka	170
4.10. Q - Kopiranje dijela memorije	170
4.11. # - Pretvaranje broja iz heksa. u decim. ..	170
5. ROM I UPOTREBA RUTINA IZ ROM-a	171
5.1. RUTINE ISPISIVANJA	171
- ispisivanje karaktera	171
- ispisivanje stringa	172
- ispisivanje numerickih vrijednosti	173
5.2. RUTINE ZA ISPITIVANJE TASTATURE	174
- Unos znaka s tastature	174
- Unos znaka s prikazom na ekranu	174
5.3. RUTINE ZA CRTANJE	174
- Crtanje tocke (PLOT)	175
- Crtanje linije (DRAW)	175
- Crtanje kruznice (CIR)	176

5.4. RUTINE SNIMANJA I UCITAVANJA	176
- Rutine snimanja	176
- snimanje zaglavlja	176
- snimanje bloka memorije bez zaglavlja ..	177
- Rutine ucitavanja	178
- ucitavanje zaglavlja	178
- ucitavanje bloka mem. bez zaglavlja	178
5.5. ADRESE OSTALIH VAZNIJIH POTPROGRAMA IZ ROM-a	179
5.6. GENERIRANJE ZVUKA	179
- jednoglas, dvoglas, troglas (viseglas) ...	179
5.7. ISPITIVANJE TASTATURE 2	182
- Ispitivanje pritisnutosti vise tastera u jednom trenutku	182
DODATAK A	184
DODATAK B	185
DODATAK C	186
- 1. Sadržaj -	
- 2. Sadržaj -	
- 3. Sadržaj -	
- 4. Sadržaj -	
- 5. Sadržaj -	
- 6. Sadržaj -	
- 7. Sadržaj -	
- 8. Sadržaj -	
- 9. Sadržaj -	
- 10. Sadržaj -	
- 11. Sadržaj -	
- 12. Sadržaj -	
- 13. Sadržaj -	
- 14. Sadržaj -	
- 15. Sadržaj -	
- 16. Sadržaj -	
- 17. Sadržaj -	
- 18. Sadržaj -	
- 19. Sadržaj -	
- 20. Sadržaj -	
- 21. Sadržaj -	
- 22. Sadržaj -	
- 23. Sadržaj -	
- 24. Sadržaj -	
- 25. Sadržaj -	
- 26. Sadržaj -	
- 27. Sadržaj -	
- 28. Sadržaj -	
- 29. Sadržaj -	
- 30. Sadržaj -	
- 31. Sadržaj -	
- 32. Sadržaj -	
- 33. Sadržaj -	
- 34. Sadržaj -	
- 35. Sadržaj -	
- 36. Sadržaj -	
- 37. Sadržaj -	
- 38. Sadržaj -	
- 39. Sadržaj -	
- 40. Sadržaj -	
- 41. Sadržaj -	
- 42. Sadržaj -	
- 43. Sadržaj -	
- 44. Sadržaj -	
- 45. Sadržaj -	
- 46. Sadržaj -	
- 47. Sadržaj -	
- 48. Sadržaj -	
- 49. Sadržaj -	
- 50. Sadržaj -	
- 51. Sadržaj -	
- 52. Sadržaj -	
- 53. Sadržaj -	
- 54. Sadržaj -	
- 55. Sadržaj -	
- 56. Sadržaj -	
- 57. Sadržaj -	
- 58. Sadržaj -	
- 59. Sadržaj -	
- 60. Sadržaj -	
- 61. Sadržaj -	
- 62. Sadržaj -	
- 63. Sadržaj -	
- 64. Sadržaj -	
- 65. Sadržaj -	
- 66. Sadržaj -	
- 67. Sadržaj -	
- 68. Sadržaj -	
- 69. Sadržaj -	
- 70. Sadržaj -	
- 71. Sadržaj -	
- 72. Sadržaj -	
- 73. Sadržaj -	
- 74. Sadržaj -	
- 75. Sadržaj -	
- 76. Sadržaj -	
- 77. Sadržaj -	
- 78. Sadržaj -	
- 79. Sadržaj -	
- 80. Sadržaj -	
- 81. Sadržaj -	
- 82. Sadržaj -	
- 83. Sadržaj -	
- 84. Sadržaj -	
- 85. Sadržaj -	
- 86. Sadržaj -	
- 87. Sadržaj -	
- 88. Sadržaj -	
- 89. Sadržaj -	
- 90. Sadržaj -	
- 91. Sadržaj -	
- 92. Sadržaj -	
- 93. Sadržaj -	
- 94. Sadržaj -	
- 95. Sadržaj -	
- 96. Sadržaj -	
- 97. Sadržaj -	
- 98. Sadržaj -	
- 99. Sadržaj -	
- 100. Sadržaj -	
- 101. Sadržaj -	
- 102. Sadržaj -	
- 103. Sadržaj -	
- 104. Sadržaj -	
- 105. Sadržaj -	
- 106. Sadržaj -	
- 107. Sadržaj -	
- 108. Sadržaj -	
- 109. Sadržaj -	
- 110. Sadržaj -	
- 111. Sadržaj -	
- 112. Sadržaj -	
- 113. Sadržaj -	
- 114. Sadržaj -	
- 115. Sadržaj -	
- 116. Sadržaj -	
- 117. Sadržaj -	
- 118. Sadržaj -	
- 119. Sadržaj -	
- 120. Sadržaj -	
- 121. Sadržaj -	
- 122. Sadržaj -	
- 123. Sadržaj -	
- 124. Sadržaj -	
- 125. Sadržaj -	
- 126. Sadržaj -	
- 127. Sadržaj -	
- 128. Sadržaj -	
- 129. Sadržaj -	
- 130. Sadržaj -	
- 131. Sadržaj -	
- 132. Sadržaj -	
- 133. Sadržaj -	
- 134. Sadržaj -	
- 135. Sadržaj -	
- 136. Sadržaj -	
- 137. Sadržaj -	
- 138. Sadržaj -	
- 139. Sadržaj -	
- 140. Sadržaj -	
- 141. Sadržaj -	
- 142. Sadržaj -	
- 143. Sadržaj -	
- 144. Sadržaj -	
- 145. Sadržaj -	
- 146. Sadržaj -	
- 147. Sadržaj -	
- 148. Sadržaj -	
- 149. Sadržaj -	
- 150. Sadržaj -	
- 151. Sadržaj -	
- 152. Sadržaj -	
- 153. Sadržaj -	
- 154. Sadržaj -	
- 155. Sadržaj -	
- 156. Sadržaj -	
- 157. Sadržaj -	
- 158. Sadržaj -	
- 159. Sadržaj -	
- 160. Sadržaj -	
- 161. Sadržaj -	
- 162. Sadržaj -	
- 163. Sadržaj -	
- 164. Sadržaj -	
- 165. Sadržaj -	
- 166. Sadržaj -	
- 167. Sadržaj -	
- 168. Sadržaj -	
- 169. Sadržaj -	
- 170. Sadržaj -	
- 171. Sadržaj -	
- 172. Sadržaj -	
- 173. Sadržaj -	
- 174. Sadržaj -	
- 175. Sadržaj -	
- 176. Sadržaj -	
- 177. Sadržaj -	
- 178. Sadržaj -	
- 179. Sadržaj -	
- 180. Sadržaj -	
- 181. Sadržaj -	
- 182. Sadržaj -	

1 - P O G L A V U L J E

1-1. SASTAV MIKRORACUNARSKOG SISTEMA "ORAO"

Standardni mikroracunarski sistem "ORAO" sastoji se od mikroracunala Orao, kucnog TV prijemnika (ili video monitora) i kazetofona. Moze se prosiriti dodavanjem disketne jedinice, stampaca, palice za igru i drugih dodataka.

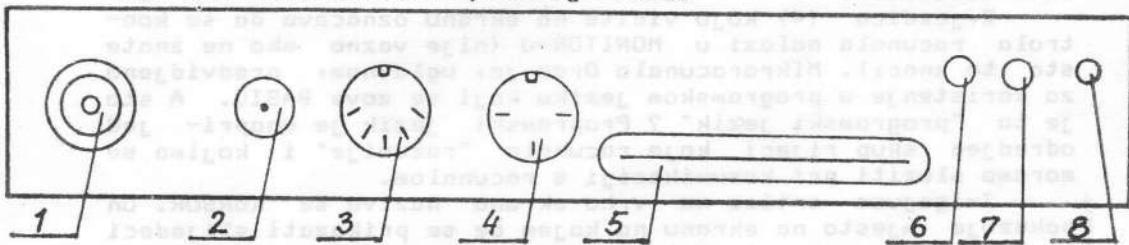
Mikroracunalo Orao sastoji se od tri osnovna dijela: osnovne ploce s elektronickim komponentama mikroracunala, tastature i ispravljača. Tastatura nam služi za unosjenje programa i podataka, a ispravljač osigurava 5V istosmernog napona potrebnog za napajanje elektronickih komponenti koje se nalaze na osnovnoj placi mikroracunala, ispod tastature. Ispod tastature nalazi se i zvucnik, pa je moguce proizvoditi razlicite tonove i melodije, naravno uz odgovarajuci program u memoriji.

Izvodjenje programa, razlicite poruke i crteze mozemo pratiti na ekranu TV prijemnika ili video monitora. Dovoljno je da imate crno-bijeli TV prijemnik, jer mikroracunalo Orao nema grafiku u boji.

Podaci i programi mogu se pomocu kazetofona biljeziti na magnetnoj vrpci (obicnoj kazeti) i s nje kasnije prebacivati, odnosno ucitavati u mikroracunalo.

1-2. PRIKLJUČIVANJE I PUŠTANJE U RAD MIKRORACUNALA ORAO

Minimalna konfiguracija potrebna za rad sastoji se od mikroracunala Orao i kucnog TV prijemnika ili video monitora koji služe za vizuelno pracenje rada.



slika 1.

1. prikljucak za video monitor
2. prikljucak za TV prijemnik
3. prikljucak za kazetofon
4. prikljucak za stampac (printer)
5. prikljucak za proširenje sistema
6. RESET taster
7. prekidac mreznog napona
8. kabel mreznog napona

Ukoliko koristite TV prijemnik, potrebno ga je prilozenim TV kablom povezati na antenski izlaz mikroracunala. Dakle, jedan kraj kabla prikljucite na antenski izlaz mikroracunala (slika 1 - 1. prikljucak za TV prijemnik), a drugi kraj na antenski ulaz TV prijemnika.

Prikljucite Orao i TV prijemnik na mrežni napon. Ukljucite televizor i postavite ga na "prvi program", na područje oko devetog kanala. Ukljucite racunalo tako da prekidac mrežnog napona potisnete prema gore (slika 1 - 7. prekidac mrežnog napona). Ako se na ekranu ne pojavi zmigajuća crtica (kursor), tada se racunalo pokreće pritiskom na RESET taster (slika 1 - 6. RESET taster).

Ako koristite video monitor, spojite video izlaz racunala (slika 1 - 1. prikljucak za video monitor) s video ulazom monitora pomocu prije spomenutog kabla. Ukljucite monitor, a mikroracunalo Orao ukljucite na ranije opisan nacin.

Nakon pravilno izvršenog spajanja i ukljucivanja mikroracunala i TV prijemnika, na ekranu cete ugledati ovakvu sliku:

*—

Razlog moguceg izostanka slike moze biti u nepodesnosti TV prijemnika. Zbog toga je obicno potrebno naknadno podešavanje slike dok se ne dobije najpovoljniji kontrast i ostrina prikazanih znakova. Buduci da mikroracunalo Orao ima u sebi ugradjen zvucnik, ton na TV prijemniku mozete potpuno iskljuciti (stisniti).

Zvjezdica (*) koju vidite na ekranu označava da se kontrola racunala nalazi u MONITOR-u (nije važno ako ne znate sta to znači). Mikroracunalo Orao je, uglavnom, predviđeno za koristenje u programskom jeziku koji se zove BASIC. A sto je to "programski jezik"? Programski jezik je unaprijed određen skup riječi koje racunalo "razumije" i kojima se moramo služiti pri komunikaciji s racunalom.

Zmigajuća crtica na vrhu ekrana naziva se KURSOR. On pokazuje mjesto na ekranu na kojem će se prikazati sljedeci znak koji unesemo s tastature.

Sada cemo kontrolu racunala prebaciti iz MONITOR-a u BASIC (precice cemo u BASIC), i to na sljedeci nacin:

Upisemo li

*BC (pritisnemo taster B i taster C)
a nakon toga pritisnemo taster (CR)
na ekranu će se ispisati:

>EAGLE< EXTENDED BASIC

V 1.0 (c) 85

MEM SIZE ?

Sada ponovo pritisnite taster <CR> i na ekranu cete imati slijedecu sliku:

>EAGLE< EXTENDED BASIC

V 1.0 (c) 85

MEM SIZE ?

23534 BYTES FREE

Dodatna objasnjenja u vezi s ispisanim porukama biti ce naknadno opisana.

Uz kurzor (zmigajucu crticu), na ekranu više nema zvjezdice (*), sto znaci da se nalazimo u BASIC-u. Sad mozemo komunicirati s mikroracunalom Orao pomocu instrukcija, odnosno rijeci koje pripadaju skupu instrukcija programskog jezika BASIC i koje mikroracunalo Orao razumije.

Pritisnemo li neki taster, dobit cemo ispis odgovarajućeg znaka na ekranu. Ako pritisnemo taster <A>:

A

pojavit ce se znak A na ekranu. Pritiskom na taster <CR> racunalo ce ispisati:

?SN ERROR

Naime, racunalo ne prepoznaje A kao instrukciju koja pripada programskom jeziku BASIC sto rezultira ispisom poruke o greski (engl. ERROR=greska).

Unesite sada slijedecu naredbu:

CLS

i pritisnite taster <CR>.

Nesto se dogodilo. Naredbom CLS obrisali smo sadrzaj ekrana (CLS je kratica od Clear Screen, sto na engleskom jeziku znaci "obrisi ekran").

Racunalo je sada razumjelo i prihvatio nasu naredbu.

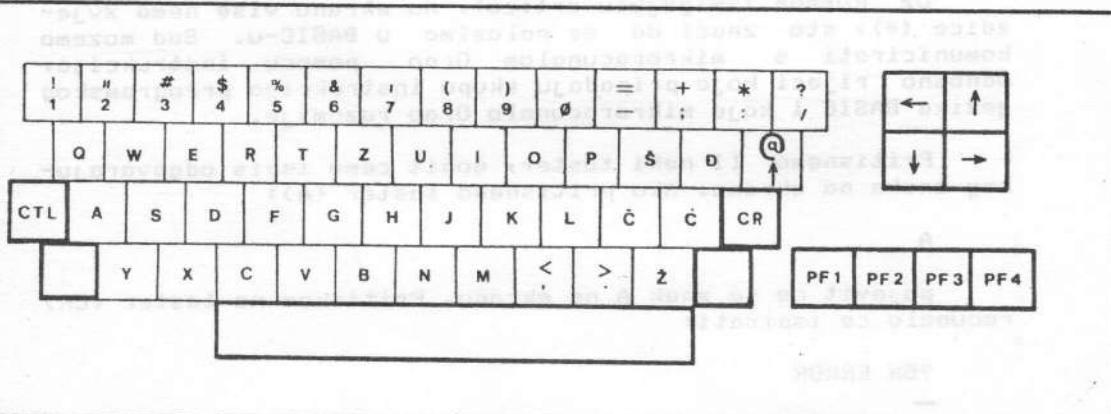
CLS je samo jedna od devedeset naredbi BASIC-a koje razumije mikroracunalo Orao. Da biste naucili programirati u BASIC-u, osim ovih naredbi potrebno je znati kako i gdje se one mogu primjeniti, uz nekoliko pravila iz sintakse (npr. na kojem mjestu dolazi tocka, dvotocka, tocka-zarez, gdje treba ostaviti razmak, a gdje ga ne smije biti ...).

Iako je BASIC programski jezik koji se vrlo brzo i lako uči, potrebno je naglasiti da se programiranje ne može naučiti samo citanjem knjiga. Potrebno je provesti mnogo vremena uz racunalo i uciti prvenstveno na svojim greskama.

1.3. TASTATURA MIKRORACUNALA ORAO

Tastatura služi za unosjenje podataka u racunalo od strane korisnika racunala. Tastatura mikroracunala Orao podijeljena je u tri dijela, na tri grupe tastera.

Najveći dio je sloviste sa standardnim rasporedom tastera i razmaknicom, uključujući i znakove C,C,S,Z,D.



slika 2.

Na slovistu mozete uociti nekoliko specijalnih tastera:

- <CR> RETURN taster
- <CTL> KONTROL taster
- < >
- < > SIFT tasteri

U trenutku kad pritisnemo taster <CR>, racunalo prihvaca, analizira i izvrsava ono što smo mu naredili preko tastature. Kod drugih mikroracunala ovu funkciju obavlja taster s označkom RETURN, ENTER ...

Taster <CTL> sluzi za izvodjenje nekih specijalnih operacija. Na primjer, ako istovremeno pritisnete <CTL> i <G>, rezultat ce biti zvucni signal iz zvucnika racunala.

Ako istovremeno pritisnete <CTL> i <L>, rezultat ce biti brisanje sadrzaja ekrana.

Slijedeca tabela daje popis svih postojećih kontrolnih operacija na mikroracunalu Orao:

✓	<CTL>, <L>	- brise sadrzaj ekrana
✓	<CTL>, <G>	- zvucni signal (BEEP)
✓	<CTL>, <F>	- brise tekst od kursora do kraja ekrana
✓	<CTL>, <H>	- pomice cursor ulijevo
✓	<CTL>, <K>	- pomice cursor prema gore
✓	<CTL>, <I>	- pomice cursor udesno
✓	<CTL>, <J>	- pomice cursor prema dolje
✓	<CTL>, <E>	- brise tekst od kursora do kraja linije
✓	<CTL>, 	- uključuje stampac
✓	<CTL>, <U>	- isključuje stampac
✓	<CTL>, <D>	- vracanje kurzora na pocetak ekrana
✓	<CTL>, <M>	- vracanje kurzora na pocetak reda (CR)
✓	<CTL>, <C>	- prekida izvodjenje BASIC programa

Od svih ovih kontrolnih funkcija najcesce cete koristiti <CTL>, <C> i <CTL>, <L>.

Neki tasteri slovista imaju na sebi dva znaka. Donji znak ispisuje se izravnim pritiskom na taster, dok ispis gornjeg dobijemo koristeci taster SHIFT (sift). Postoje dva tastera SHIFT - lijevi se nalazi ispod tastera CTL, a desni ispod tastera CR. Na taj nacin moguce je dobiti:

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 mozemo izvoditi izravno, bez upotrebe CTL tastera. U tu svrhu koristimo cetiri tastera smjestena u gornjem desnom uglu tastature. Kursor se moze pomicati u sva cetiri pravca po ekranu, sto je narocito korisno prilikom ispravljanja (editiranja) BASIC programa.

5 *** ORAO PRIRUCNIK ***

Preostala su nam još četiri FUNKCIJSKA tastera. To su tasteri s označama:

PF1 i PF2 su za izvođenje priznata programa na ekran
PF3 je za prebacivanje znakova u INVERTIRANI oblik.
PF4 je za kopiranje dijela teksta.

PF1 - pritiskom na ovaj taster racunalo ispisuje MALA slova. Povratak na ispis velikim slovima postize se ponovnim pritiskom na PF1.

PF2 - sluzi za uključivanje stampaca (printera). Uključivanje se postize ponovnim pritiskom na PF2.

PF3 - prebacuje ispis znakova u INVERTIRANI oblik. Povratak na prvobitno stanje postize se ponovnim pritiskom na ovaj taster.

PF4 - KOPI taster, sluzi za kopiranje dijela teksta prilikom ispravljanja programa. Uloga ovog tastera detaljno ce biti opisana naknadno.

Ukoliko drzimo neki taster pritisnut dulje vrijeme, ispis znaka ce se ponavljati automatski (AUTOREPEAT).

1.4. ŠTO JE BASIC

Svima nam je poznato da se u međusobnom komuniciranju ljudi služe govorom. Međutim, ne govore svi ljudi istim jezikom. Svi ljudi koji govore istim jezikom međusobno se razumiju. Ako nekome tko poznaje samo jedan jezik kazemo nesto na nekom drugom jeziku, on nas neće razumjeti.

Kao i ljudi, i racunala imaju svoje jezike. Jedan jezik na racunalu obuhvaca skup riječi koje tvore taj jezik. Jedan od takvih jezika je i BASIC. Kad kazemo da neko racunalo "razumije" BASIC, to znači da racunalo razumije skup riječi koje zajedno sачinjavaju jezik BASIC. Da bismo ove jezike razlikovali od jezika kojima govore ljudi, nazivamo ih programske jezice.

Medju ljudima koji govore istim jezikom ima manjih ili vecih razlika između riječi koje oni upotrebljavaju, zavisno od kraja iz kojeg potjecu. Takve podskupove jezika nazivamo dijalektima.

Sva racunala istog tipa razumiju isti BASIC, pa možemo program napisati na jednom Orau prenijeti na drugi Orao. Na racunalima razlicitih proizvodjaca postoje manje ili veće razlike između programskih jezika. Tako program napisan u BASIC-u na mikroracunalu "spectrum" neće raditi na mikroracunalu Orao dok se ne izvedu potrebne izmjene u programu. Zbog toga govorimo o dijalektima programskih jezika.

Ovaj prirucnik bavi se dijalektom programskog jezika BASIC za mikroracunalo Orao.

Za razliku od jezika u svakodnevnom govoru za cije je učenje potreban visegodisnji rad, programske jezike možemo naučiti u vrlo kratkom vremenu. To slijedi iz toga što programski jezici sadržavaju vrlo malen skup riječi (stotinjak). Kada naučimo jedan dijalekt nekog programskog jezika, potrebno nam je samo nekoliko sati da savladamo drugi dijalekt istog programskog jezika. Isto tako, ako dobro poznajemo jedan programski jezik, vrijeme potrebno za učenje svakog slijedećeg bit će sve kraće.

1.5. PROGRAMIRANJE U BASIC-U

Za vrijeme programiranja u BASIC-u na mikroracunalu Orao, može se raditi na dva načina. Prvi način je direktni, a drugi je programski.

1.5.1. DIREKTNI NACIN RADA

Kad se radi u direktnom načinu rada, prvo se upisuje naredba, a zatim se pritisne taster **<CR>**, nakon čega računalo odmah izvršava tu naredbu (naredbe se upisuju bez brojeva programskih linija).

Upisite slijedeću naredbu:

SOUND 100,100

(pritisnite redom tastere **S,O,U,N,D,razmaknica,1,0,0,zarez,1,0,0**).

Pritisnite sada taster **<CR>**. Time ćemo računalu javiti da smo završili s upisom. Orao će sada izvršiti nasu naredbu, odnosno iz zvučnika ćemo čuti jedan zvučni signal.

Napomena:

Umjesto broja **0** nije dozvoljeno pisati slovo **O**. Za broj **0** postoji poseban taster. Isto tako, umjesto broja **1** ne može se koristiti malo slovo **I**.

U direktnom načinu rada, u jednoj se liniji može navesti i više naredbi, ali tada one moraju biti odvojene dvotockom. Upisite slijedeći primjer:

CLS:SOUND 50,150 i pritisnite taster **<CR>**.

Vidimo da je računalo izvršilo oba dvije naredbe - prvo je obrisan ekran, a onda je odsvirao odabrani ton.

U direktnom načinu rada računalo ne pamti naredbe poslije njihovog izvršavanja. Ako želimo ponoviti prethodni primjer, moramo upisati oba dvije naredbe ponovo.

Na ovaj način ne mogu se napisati gotovo nikakvi programi. Direktni način pogodan je za brza i kratka matematička izračunavanja u slučajevima da se rezultati neće dalje koristiti u računalu.

1.5.2. PROGRAMSKI NACIN RADA

U programskom nacinu rada prvo se napise program koji se spremi i pamti u memoriji racunala, a zatim se taj program izvrsava pomocu naredbe RUN.

Program se sastoji od niza programske linija. Svaka od njih pocinje brojem koji moze biti cijeli broj u opsegu od 0 do 63999. Iza broja linije slijedi jedna ili vise naredbi medjusobno odvojene dvotockom.

Da bi se olaksalo ispravljanje programa i umetanje novih programske linija, brojevi linija ne bi trebali biti uzastopni brojevi. Uobicajeno je da se razlikuju za 5 ili 10.

Program pisemo na slijedeci nacin:

```
10 CLS           i pritisnemo <CR>
20 PRINT "MIKRORACUNALO"   i pritisnemo <CR>
30 PRINT "ORAO"         i ponovo <CR>
40 SOUND 150,100:SOUND 50,200 <CR>
```

Ovo je sada jedan BASIC program. Za izvrsavanje ovog (kao i svakog drugog) BASIC programa potrebno je napisati:

```
RUN   i pritisnuti <CR>
```

Kada ste izvrsili ovaj program, otkucajte naredbu:

```
LIST   i <CR>, naravno
```

Cijeli program opet je pred nama na ekranu, sto znaci da ga nismo izgubili, nego se on nalazi pohranjen u memoriji racunala.

Dodajmo sada naredbu:

```
15 PRINT "JA SAM" <CR>
```

Otkucajte ponovo

```
LIST <CR>
```

Na ekranu se vidi da je programska linija s brojem 15 smjestena izmedju linija 10 i 20, sto znaci da ce se naredba:

```
15 PRINT "JA SAM"
```

izvrsiti odmah nakon izvrsavanja naredbe u liniji 10, odnosno neposredno prije izvrsenja naredbe u liniji 20.

Izvrsite i ovaj program naredbom RUN <CR>.

Racunalo ce ispisati

```
JA SAM
```

```
MIKRORACUNALO
```

```
ORAO
```

Kako da obrišemo, odnosno uklonimo programsku liniju iz naseg programa? Vrlo jednostavno. Zelimo li obrisati liniju:

15 PRINT "JA SAM" i da je samo upisati broj te programske linije i pritisnuti <CR>.

Upisite najprije naredbu:

LIST <CR>

a zatim:

15 <CR>

i nakon toga ponovo:

LIST <CR>

Vidimo da je programska linija broj 15 izbrisana iz programa.

Prije nego zapocnemo pisati novi, potrebno je obrisati stari program iz memorije računala.

To se postize naredbom NEW. Upisite dakle:

NEW <CR>

a nakon toga:

LIST <CR>

Od naseg programa nije ostalo nista, izbrisani je.

1.6. KAKO SE KORISTE TASTERI ZA POMICANJE KURSORA I KOPI TASTER <PF4>

Da bi lakse shvatili na koji nacin se koriste tasteri za pomicanje cursora u kombinaciji s tasterom <PF4>, odnosno kako se ispravlja (editira) program, napisat cemo ponovo nas stari program:

10 CLS
20 PRINT "MIKRORACUNALO"
30 PRINT "ORAO"
40 SOUND 150,100:SOUND 50,200

Program cemo izvrsiti naredbom RUN:

MIKRORACUNALO

ORAO

Sadrzaj ekrana mozemo obrisati ako istovremeno pritisnemo tastere **<CTL>** i **<L>**. Pokusajte.

Obrisali smo, znaci, ekran, ali ne i cijeli program iz memorije racunala. Izlistajte program na ekran:

LIST <CR>

i pred nama je ponovo cijeli program:

```
10 CLS
20 PRINT "MIKRORACUNALO"
30 PRINT "ORAO"
40 SOUND 150,100:SOUND 50,200
```

Pretpostavimo da sada zelimo promijeniti liniju:

```
20 PRINT "MIKRORACUNALO"
```

tako da cemo umjesto "MIKRORACUNALO" ispisati rijec "MIKROKOMPJUTER".

To se moze napraviti na dva nacina. Prvi je taj da napisemo novu liniju 20:

```
20 PRINT "MIKROKOMPJUTER"
```

Na ovaj nacin nova linija broj 20 zamijenila nam je steru.

Drugi nacin je mnogo laksi. Mikroracunalo Orao pruza nam mogucnost vrlo brzog, jednostavnog i ugodnog ispravljanja programske linije. U tu svrhu koriste se tasteri za pomicanje kursora i KOPI TASTER **<PF4>**.

Postupak ispravljanja linije (u nasem slucaju linije broj 20) je slijedeci:

Pritisnite taster za pomicanje kursora prema gore (**↑**) nekoliko puta, tako da nam zmigajuca crtica dodje ispred linije broj 20:

```
10 CLS
20 PRINT "MIKRORACUNALO"
30 PRINT "ORAO"
40 SOUND 150,100:SOUND 50,200
```

Primjecujete da na ekranu sada imamo dva cursora, zmigajuca crtici i bijeli kvadratic.

Sad nam bijeli kvadratic označava mjesto na ekranu na kojem će se prikazati slijedeci znak, a zmigajuca crtici mozemo pomocu cetiri tastera za pomicanje cursora voditi po cijelom ekranu i zeljene znakove s bilo kojeg mesta ekrana

kopirati pritiskom na taster <PF4> u novo formirajuću liniju u kojoj se nalazi bijeli kvadrat.

Pritisnite sad <PF4> toliko puta da zmagajuća crtica dodje ispod drugog slova "R" u rjeci "MIKRORACUNALO":

```
10 CLS  
20 PRINT "MIKRORACUNALO"  
30 PRINT "ORAO"  
40 SOUND 150,100:SOUND 50,200  
20 PRINT "MIKRO" ■
```

Vidimo da se ispod linije 40 formira nova linija 20 koja će zamijeniti staru.

Preostalo nam je još jedino da rijec "RACUNALO" zamijenimo s "KOMPJUTER". Kako? Vrlo jednostavno - otkucajte na tastaturi "KOMPJUTER":

```
10 CLS  
20 PRINT "MIKRORACUNALO"  
30 PRINT "ORAO"  
40 SOUND 150,100:SOUND 50,200  
20 PRINT "MIKROKOMPJUTER" ■
```

Nedostaje nam još samo navodnik iz rjeci "MIKROKOMPJUTER". Njega ćemo iskopirati iz stare linije broj 20 na sljedeći nacin:

dovedite zmagajucu crticu (koja se još uvijek nalazi ispod slova "R"), pritiskom na taster za pomicanje kursora udesno (→), ispod drugog navodnika u liniji 20:

```
20 PRINT "MIKRORACUNALO" ■
```

Sad pritisnite <PF4> i navodnik je iskopiran u novu liniju 20.

Pritisnite <CR>. Bijeli kvadrat je nestao s ekrana, ostao nam je samo pravi cursor.

Uvjerite se da sad u liniji broj 20 umjesto "MIKRORACUNALO" imamo rijec "MIKROKOMPJUTER":

```
LIST <CR>  
10 CLS  
20 PRINT "MIKROKOMPJUTER"  
30 PRINT "ORAO"  
40 SOUND 150,100:SOUND 50,200
```

Otkucajte sad, umjesto naredbe RUN, naredbu RUM, ali nemojte pritisnuti <CR>. Naredba RUM, naravno, ne postoji i zato RUM trebamo zamijeniti s RUN. Na koji nacin?

Primjetili ste da sada, kad imamo dva kursora na ekranu, taster (\leftrightarrow) vise ne sluzi za brisanje pogresno unesenog znaka s tastature. Svaki znak kojeg bi sad unesli s tastature prikazao bi se na onom mjestu na koje pokazuje bijeli kvadrat, a brise se istovremenim pritiskom na tastere $\langle CTL \rangle$ i $\langle H \rangle$. Dakle, u trenucima kad ispravljamo program i imamo dva kursora na ekranu, funkciju brisanja znaka vrsimo pomocu $\langle CTL \rangle \langle H \rangle$.

Sada cemo u donjoj liniji 20 obrisati navodnik. Pritisnite istovremeno tastere $\langle CTL \rangle$ i $\langle H \rangle$ (bolje je prvo pritisnuti $\langle CTL \rangle$ i drzati ga pritisnutim, a onda jedanput i pritisnuti $\langle H \rangle$). U gornjoj liniji rezultat je na sljedećem slike.

```
10 CLS
20 PRINT "MIKROKOMPJUTER"■
30 PRINT "ORAO"■
40 SOUND 150,100:SOUND 50,200
30
20 PRINT "MIKROKOMPJUTER"■
```

Ovime smo obrisali navodnik u donjoj liniji 20. Sada cemo u donju liniju 20 ubaciti jedno prazno mjesto, a onda iz linije 30 iskopirati rijec ORAO u donju liniju 20.

Pritisnite razmaknicu, a nakon nje $\langle PF4 \rangle$ toliko puta da iskopirate rijec ORAO zajedno s navodnikom.

Na kraju pritisnite $\langle CR \rangle$. Obrisite ekran pomocu $\langle CTL \rangle$ i $\langle L \rangle$ i otukajte LIST:

```
10 CLS
20 PRINT "MIKROKOMPJUTER ORAO"
40 SOUND 150,100:SOUND 50,200
```

Vidi se da linije 30 u programu nisu nema, a linija 20 je kombinacija starih linija 20 i 30.

Izvrsite program pomocu:

Dakle, koristeci tastere za pomicanje kursora, $\langle PF4 \rangle$, $\langle CTL \rangle$ i $\langle H \rangle$, moguce je popravljati tekst, odnosno program na ekranu sve dok se ne dobije zeljeni oblik. Opisane mogucnosti ispravljanja (editiranja) teksta na ekranu od izuzetne su važnosti jer omogucavaju korisniku da uz jednostavne zahvate popravlja ili mijenja sadrzaj bilo koje programske linije.

1.7. PRIMJER PROGRAMA U STROJNOM JEZIKU

Programi napisani u BASIC-u zauzimaju (odnosno "troše") puno memorijskog prostora, a brzina izvršavanja im je relativno mala. Ovi nedostaci dolaze naročito do izrazaja u većim BASIC programima. Prednost BASIC programa je u tome što su oni vrlo pregledni, lako se pisu i ispravljaju.

Pisanjem programa na strojnem jeziku, na jeziku mikroprocesora, osigurava se povećanje brzine izvodjenja programa i smanjenje potrošnje memorije u odnosu na BASIC programe. S druge strane, pisanje programa u strojnem jeziku je mnogo teže i zahtijeva od korisnika dodatna znanja i iskustva.

Programiranje u strojnem jeziku na mikroracunalu Orao izvodivo je pomoći MINIASSEMBLER-a koji se nalazi ugradjen u ROM memoriju mikroracunala i sastavni je dio MONITORA (drugi dio ROM memorije zauzima BASIC). Mogućnost rada u MONITORU dobivamo odmah poslije uključivanja racunala. Ponekad se može dogoditi da se poslije uključenja racunala ne pojavi cursor na ekranu. U tom slučaju dovoljno je pritisnuti RESET taster sa zadnje strane racunala i racunalo će ispisati:

*—
sto je znak da se nalazimo u MONITORU (odakle možemo pisati programe u strojnem jeziku).

Predjite u BASIC:

```
BC <CR>          10 CLS  
<CR>          20 PRINT "MIRKOMPUTER ORAO"  
              30 BEEP 1000,1000,1000
```

i na ekranu se pojavila već poznata nam poruka.

Da biste dobili osjecaj o brzini izvršavanja BASIC programa prema brzini izvršavanja STROJNOG programa, napisat ćemo ISTI program u BASIC-u i u strojnem jeziku. Taj će program "obojiti" u bijelo cijelu površinu ekrana. Naredbe koje ćemo koristiti u ovom primjeru kasnije će biti detaljno objasnijene.

Prepisite pazljivo najprije BASIC program:

```
10 CLS  
20 FOR N=24576 TO 32768  
30 POKE N,255  
40 NEXT N
```

Izvršite ga naredbom RUN:

```
RUN <CR>
```

Pricekajte nekoliko sekundi da se "oboji" cijeli ekran.

Kad se pojavi cursor u gornjem lijevom kutu ekranu, obrisite ekran pomocu <CTL>, <L>.

Sada ćemo napisati isti program, ali u strojnom jeziku. Pomocu naredbe EXIT preci ćemo iz BASIC-a u MONITOR. Napisite:

| EXIT <CR> LIST <CR>

Pojavila se je zvjezdica, znak monitora. Strojni program pocet ćemo pisati na slijedeci nacin:

Napisite:

| A 1000 <CR> NEW <CR>

Na ekranu se pojavilo:

1000_ Sada vrlo pazljivo prepisite slijedeci strojni program:

✓ LDA #0C <CR>
JSR FFF1 <CR>
LDY #00 <CR>
LDX #60 <CR>
STY 20 <CR>
STX 21 <CR>
LDA #FF <CR>
STA (20),Y <CR>
INY <CR>
BNE 100F <CR>
INC 21 <CR>
LDA 21 <CR>
CMP #80 <CR>
BNE 100D <CR>
RTS <CR>
C <CR>

U trenutku kad ste unesli "C", na ekranu se ponovo pojavila zvjezdica (*), ponovo smo u MONITORU.

Izvršite maloprije unesen program tako da napisete:

| U 1000 <CR>

Razlika u brzini izvodjenja ova dva programa je ocita.

Vratimo se natrag u BASIC, ali kako? Nama je poznato jedino pomocu naredbe "BC". Međutim, BC će nepovratno izbrisati sadržaj cijele memorije računala, a mi bi htjeli sacuvati nasa oba dva programa.

Da bi sacuvali sadržaj memorije, u BASIC prelazimo na slijedeci nacin:

Napisite:

BW <CR>

i ponovo smo u BASIC-u.

Ako jos niste, obrisite bijeli ekran pomocu <CTL>, <L>. Otkucajte naredbu LIST:

LIST <CR>

i BASIC program je ponovo pred vama na ekranu. Mozete ga izbrisati iz memorije naredbom NEW:

NEW <CR>

uz napomenu da je obrisan samo BASIC program, a strojni je jos u memoriji racunala.

Ne brinite previse ako niste razumjeli sto znace pojedine instrukcije u ovim programima. Kad proucite sadrzaj cijelog prirucnika, ovakvi primjeri cinit ce vam se vrlo smijesni. Vazno je da ste uocili prednosti, mane i razlike izmedju BASIC i strojnih programa.

Poslije ovog uvodnog dijela, koji je prije svega namijenjen laksem savladavanju osnovnih problema rada na racunalu, preci cemo na rjesavanje slozenijih i konkretnijih problema, koristeci pri tom dvije izuzetne prednosti racunala:

- brzinu
- tocnost

Mozda ce nekome od vas prvi koraci u radu s mikroracunalom biti malo tezi, međutim, svakog dana naučit cete nesto novo, ulazeci tao u jedan izuzetan svijet, svijet (mikro)racunala.

1.8. UČITAVANJE PROGRAMA S KAZETE (UVOD)

Ucitavanje programa s kazete predstavlja prebacivanje programa koji se nalazi snimljen na obicnoj muzickoj kazeti, u memoriju mikroracunala Orao. Da bi se to postiglo, potrebno je povezati mikroracunalo i kazetofon na slijedeci nacin:

- povezivanje se vrsti specijalnim kablom tako da se jedan kraj kabla usteka u prikljucak za kazetofon koji se nalazi na straznjoj strani mikroracunala. Drugi kraj kabla priključuje se na kazetofon i to tako da se EAR prikljucak racunala poveze s EAR prikljuckom na kazetofonu, a isto tako MIC prikljucak racunala treba povezati s MIC prikljuckom kazetofona.

Programe koji su prethodno upisani treba izbrisati. Najbolji nacin da to uradite je da "resetirate" racunalo (tj. da pritisnete RESET taster koji se nalazi na straznjoj strani racunala). Nakon reseta na ekranu se pojavljuje zvjezdica (znak MONITORA) i cursor. Brisanje memorije i prelazak u Basic postize se naredbom

BC <CR>
<CR>

Sada smo racunalo pripremili za ucitavanje programa s kazete. Kazetu premotajte na pocetak programa, a potenciometar za glasnocu reprodukcije na kazetofonu podesite na 2/3 maksimuma.

Postoje dvije naredbe za ucitavanje programa s kazete, ovisno o tome da li se na kazeti nalazi Basic program ili je to strojni program. Ako se ucitava Basic program, treba upisati slijedecu naredbu:

LOAD "ime programa" <CR> ili
LOAD "" <CR>

Ako se ucitava strojni program, potrebno je upisati:

LMEM "ime programa", adresa <CR> ili
LMEM "ime programa" <CR> ili
LMEM "", adresa <CR> ili
LMEM "" <CR>

Na primjer:

LMEM "S.INVADERS",512 <CR>
LMEM "BREAKOUT" <CR>

Buduci da postoji nekoliko varijanti ucitavanja programa s kazete u racunalo, potrebno je da uz svaki program imate i upute za njegovo ucitavanje. Strojni programi snimljeni su obicno tako da se odmah nakon ucitavanja s kazete sami i startaju. Za pokretanje Basic programa potrebno je upisati Basic naredbu RUN <CR>.

Rad s kazetofonom je detaljno objasnjen u poglavlju 2.16

2. POGлавље

NAREDBE PROGRAMSKOG JEZIKA BASIC I NJIHOVA UPOTREBA

2.1 Uvod

- MONITORSKE NAREDBE BC i BW
- Nakon uključivanja mikroracunala Drao, na ekranu će se pojaviti zvjezdica (*) uz koju zmiga cursor. Zvjezdica je znak da se kontrola racunala nalazi u MONITORU.

MONITOR ili MONITORSKI PROGRAM je program koji je zapisan u ROM memoriji racunala (ne brise se isključivanjem racunala). Taj se program može nazvati i OPERATIVNI SISTEM racunala - on obavlja sve "poslove" koji su nuzni za normalno koristenje racunala. Čim se racunalo uključi, taj se program pocije izvršavati. Prije nego što se pojavi zvjezdica i cursor na ekranu, monitorski program je vec odradio dio svojeg posla. U trenutku kad se pojavi cursor na ekranu, racunalo je spremno za rad.

Mi cemo na pocetku uciti programirati na programskom jeziku BASIC. Da bi to mogli, potrebno je iz MONITORA preci u BASIC. To se postize ako u monitoru otkucamo:

BC <CR>

Ovime se pokreće inicializacija BASIC-a uz brisanje sadržaja kompletne memorije koja nam стоји na raspolaaganju.

Nakon uključivanja racunala prelazak iz monitora u BASIC moguc je samo naredbom BC.

Pisuci programe u BASIC-u, ponekad će se javiti potreba za ponovni prelazak u MONITOR. To se postize naredbom:

EXIT <CR>

ili pritiskom na RESET taster na straznjoj strani racunala.

Ponovni prelazak u BASIC sada je moguc i ako napisemo:

BW <CR>

Na ovaj nacin izbjegli smo brisanje sadržaja memorije, odnosno naseg programa.

U BASIC smo se mogli vratiti i pomocu naredbe BC, ali bi pri tome izgubili nas program iz memorije.

Na ovaj nacin izbjegli smo brisanje sadržaja memorije, odnosno naseg programa.

2.2 KAKO SE PIŠU PROGRAMI U BASICU

Odsada ćemo se postepeno upoznavati s pojedinim naredbama BASIC-a. Naredbe BASIC-a na mikroracunalu Orao mogu se pisati samo velikim slovima, a malim ne.

Radi veće preglednosti programa koje ćemo pisati, između pojedinih naredbi, izraza, varijabli i komentara ostavljamo prazno mjesto (razmak). Primjer:

Programska linija

10 PRINT "ORAO"

svakako da je preglednija od linije

10PRINT"ORAO"

Umjesto jednog pravnog mesta, mogu se upotrijebiti dva ili više. Međutim, upotreba razmaka nije obavezna. Koristimo ga samo radi bolje preglednosti programa.

Evo još nekoliko primjera gdje je dozvoljeno koristiti razmak:

DOZVOLJENO JE:

10PRINT "ORAO"
1 0 PRINT "ORAO"
20 0 0 PRINT"OR AO"

NIJE DOZVOLJENO:

10 PRI NT "ORAO"

2.3. VARIJABLE

2.3.1. BROJEVI I NUMERICKE VARIJABLE

Za rad u BASIC-u na mikroracunalu Orao se koriste decimalni brojevi koji se predstavljaju nizom decimalnih cifara. Njih odlikuje slijedeće:

- umjesto uobičajenog decimalnog zareza koristi se decimalna točka
- nula se predstavlja znakom 0
- prazna mesta između cifara nisu dozvoljena
- broj može poceti decimalnom točko (npr. 0.23=.23)
- predznak se navodi ispred broja, a ako nije naveden, vrednost je pozitivna
- broj se može predstaviti i u eksponencijalnom obliku

Primjeri: PRAVILNO NEPRAVILNO

3.14	3,14
12300	12 300
.99	,99
123E2	13.300.5

Brojni izraz je postupak za određivanje brojne vrijednosti i sastoji se od ARGUMENATA (brojeva ili numerickih varijabli) nad kojima se izvršavaju odgovarajuće operacije.

Primjer:
 $\sqrt{4} + 3$

Za određivanje vrijednosti navedenog izraza, potrebno je izvršiti operaciju korjenovanja argumenta 4, a zatim zbrajanje argumenta 2 (to je rezultat korjenovanja) i argumenta 3.

Numericka varijabla je simbolicka oznaka (npr. "X") kojoj se može dodijeliti brojni podatak (npr. "X=35").

Varijabla NE POSTOJI (NIJE DEFINIRANA) tako dugo dok joj se ne dodijeli neka vrijednost.

Simbolicka oznaka varijable (IME varijable) može biti tipa:

XY

X - OBAVEZNO mora biti VELIKO SLOVO iz ASCII seta znakova
Y - SLOVO ili ZNAMENKA iz ASCII seta znakova

PRAVILNO	NEPRAVILNO
A	7C
B2	\$AB
QY	98

Primjer: (unesite direktno)

X=7 <CR>

Sada smo simbolickoj varijabli X dodijelili numericku vrijednost 7. Da se u to uvjerite, napisite:

PRINT X <CR>

7

Operacija "=" ovdje nema u potpunosti isto značenje kao i znak jednakosti u algebarskim izrazima. Ona ovdje isključivo služi za to da varijabli na lijevoj strani (X) dodijeli vrijednost varijable (ili izraza) s desne strane (7).

Pokusajmo sada napisati:

X=X+1 <CR>

a nakon toga: S -> KONTROLNE NAKLJUČKE

```
PRINT X<CR>
PRINT X<CR>
PRINT X<CR>
PRINT X<CR>
PRINT X<CR>
PRINT X<CR>
```

sto znaci da smo simbolicku vrijednost variabile X uvezali za 1 i ponovo je dodijelili varijabli X.

Visedimenzionalnim varijablama (polja, matrice - o tome cemo uciti kasnije) potrebno je dodjeliti prostor u memoriji naredbom DIM. Za imena visedimenzionalnih varijabli vrijedi isto sto i za imena obicnih varijabli.

Potrebno je naglasiti da se ime varijable moze sastojati i od vise znakova, ali su znacajna samo prva dva.

2.3.2. STRINGOVI I STRING VARIJABLE

String je svaki niz bilo kakvih znakova (slova, znakove, specijalni znakovi). Na primjer:

```
abcde
AbC3%
&Mkop4
A2      i slicno.
```

Postoje takodjer i string varijable, a za njihova imena vaze ista pravila i ogranicenja kao i za imena numerickih varijabli. Razlika je jedino u tome sto iza imena string varijable mora stajati znak \$ (dolar). Na primjer:

```
A$
B1$
X7$      i slicno.
```

Kao i kod numerickih varijabli, string varijabla nije definirana tako dugo dok joj se ne dodijeli neki string, npr:

```
A$="ORAO"
C2$="skola"
80 G$="===="      i slicno.
```

Primjer:

```
R$="22*3=" <CR>
X=22*3    <CR>
PRINT R$:PRINT X <CR>
```

Na ekranu ce se pojaviti:

```
22*3=
66
```

2.4. KONTROLNE NAREDBE

Kontrolne naredbe nam služe za kontrolu izvođenja BASIC programa.

Najprije ćemo upoznati dvije OPERATIVNE NAREDBE, a to su RETURN i BREAK.

RETURN (CR - CARRIAGE RETURN)

U direktnom nacinu rada naredbom RETURN (tj. onda kad pritisnemo taster <CR>)javljamo racunalu da smo zavrsili s upisom i naredjujemo mu da izvrsti zadane naredbe.

U programskom nacinu rada je poslige formiranja programske linije potrebno da se ona smjesti u dio memorije u kojem se pamti program. To se takodjer postize naredbom RETURN (tj. onda kad pritisnemo taster <CR>).

BREAK

Naredbom BREAK (tj. onda kad pritisnemo <CTL> i <C>) prekida se izvođenje BASIC programa. Do prekida će doći kad se izvrsti tekuća naredba. Na ekranu će se pojaviti:

BREAK

i upitnik iza naredbe koja se je upravo izvršila.

RUN

Kad je program unesen u memoriju, njegovo izvršavanje otpocinje naredbom RUN. Program se pocinje izvršavati od linije koja ima najmanji broj. Na početku izvođenja programa sve varijable se automatski obrišu (numerickie varijable poprime vrijednost 0, a stringovi postaju "prazni").

Stavljanjem cijelog broja iza naredbe RUN, izvršavanje programa će zapoceti od programske linije s tim brojem, npr:

RUN 150 <CR> - program će se izvršiti počevši od linije broj 150

I u ovom slučaju na početku se brišu sve programske varijable. Efekat ove naredbe je isti kao u slučaju naredbe:

CLEAR:GOTO 150

STOP

Ova naredba zaustavlja izvođenje programa i ispisuje poruku BREAK i cijelu liniju u kojoj se nalazi STOP.

Primjer:

```
10 PRINT "MIKRORACUNALO"
20 SOUND 100,100:STOP:CLS
30 PRINT "ORAO"
```

RUN <CR>

```
MIKRORACUNALO
BREAK
20 SOUND 100,100:STOP?:CLS
```

CONT

Izvršavanje programa prekinutog s <CTL>, <C> ili naredbom STOP može se nastaviti naredbom CONT (engl. "continue" znači "nastaviti").

Unesite:

```
CONT <CR>
```

i nastavite s izvršavanje maloprije prekinutog programa:

```
ORAO
```

Potrebno je napomenuti da u slučaju prekida izvršavanja direktnih naredbi nije moguce nastaviti njihovo izvodjenje.

END

Naredba END određuje kraj programa, ali nije obavezna.

Primjer:

```
10 CLS
20 PRINT "VJEZBA"
30 END
```

NEW

Prije nego što počnemo unositi novi program u računalo, potrebno je pomocu naredbe NEW obrisati iz memorije računala stari program i vrijednosti programske varijable.

Dakle, naredbom NEW u direktnom nacinu brišemo BASIC program iz memorije računala.

REM

U cilju što bolje preglednosti dužih programa, naročito u toku njihovog razvijanja, pojedina mesta u programu potrebno je označiti i objasniti, odnosno iskomentirati.

Za to nam služi naredba REM. Iza nje se mogu napisati nasi komentari i objasnenja određenih mesta programa. Iza REM dozvoljeno je napisati bilo koje znakove, čak i druge naredbe, ali se one neće izvršavati jer će ih računalo

shvatiti kao nas komentar. Evo nekoliko primjera:

2540 REM 2550 do 2600 crtanje slike
740 REM *** skok na potprogram 2 ***

Ako napisemo:

1100 REM ispitivanje tastature:CLS

naredba CLS iza dvotocke se neće izvršiti jer je ona, kao i sve ostalo sto se nalazi iza REM, shvacena kao nas komentar programu.

U toku razvijanja programa preporučljivo je da si program sto bolje i jasnije iskomentirate, tj. da u programu imate sto vise kratkih i jasnih REM linija, odnosno komentara. Kada se razvijanje programa zavrsi, REM naredbe se mogu ukloniti iz programa u cilju dobijanja sto kraceg programa i njegovog sto brzeg izvršavanja.

CLS

Ovu smo naredbu dasad vec upoznali, ali da ponovimo:

Naredba CLS briše sadržaj ekranu, tj. briše sadržaj onog dijela memorije u kojem se pamti sve ono sto vidimo na ekranu (tzv. VIDEO memorije).

LIST

Naredbom LIST moguce je prikazivanje na ekranu prethodno unesenog programa ili samo zeljenog dijela programa (tzv. listanje). Naredba LIST može se koristiti na pet nacina:

1. LIST <CR>
 - lista cijeli program pocevsi od linije s najmanjim brojem
2. LIST 25 <CR>
 - lista samo liniju s brojem 25
3. LIST -90 <CR>
 - lista program do linije 90 uključujući i liniju 90
4. LIST 130- <CR>
 - lista program od linije 130 (uključujući i tu liniju) pa do kraja programa
5. LIST 245-320 <CR>
 - lista sve linije izmedju ove dviye uključujući i njih

Listanje programa može se prekinuti pomocu <CTL> i <C>.

EXIT

Za prebacivanje kontrole racunala iz BASIC-a u MONITOR koristi se naredba EXIT. Dakle, ako napisemo:

EXIT <CR>

<CR> A TINYB
CAS

pojaviti će se zvjezdica, znak MONITORA. U BASIC se vraćamo na jedan od dva moguća nacina. Prvi je taj da napisemo BC <CR> i tako izbrisemo program, a drugi je da napisemo BW <CR> i tako sacuvamo program u memoriji računala.

2.5 - ULAZNO-IZLAZNE NAREDBE

Ulažno-izlazne naredbe služe za kontrolu toka podataka između računala i ulazno-izlaznih jedinica. U osnovnom obliku, mikroracunalo Orao ima kao ULAZNU jedinicu tastaturu, a kao IZLAZNU jedinicu TV ekran ili video monitor.

Ovdje ćemo upoznati naredbe PRINT i INPUT.

PRINT

Ovom naredbom se na ekran ispisuju rezultati izracunavanja, tekst, skoro sve što se ispisuje na ekran ispisuje se pomoću naredbe PRINT. U dosadašnjim primjerima već smo ju donekle i upoznali, a sada ćemo o njoj reci sve ostalo.

Budući da je to naredba koja se najvise upotrebljava, umjesto PRINT može se napisati i znak "?" (upitnik). Na taj način program se pise brže. Znak "?" i riječ "PRINT" za računalo imaju potpuno isto značenje.

Postoji nekoliko načina upotrebe naredbe PRINT:

1. ISPIŠIVANJE BROJEVA, VRIJEDNOSTI NUMERICKIH VARIJABLJI I IZRAZA

Primjer: unesite sljedeći program koji će nam ispisati zbroj dva broja

```
10 A=245
20 B=76
30 PRINT A+B
```

RUN <CR>
321

U programskoj liniji 10 definirali smo numericku varijablu A tako da smo joj dodijelili vrijednost 245. U liniji 20 definirali smo drugu numericku varijablu, varijablu B i pridruzili joj vrijednost 76. U liniji 30 pomoću naredbe PRINT ispisujemo rezultat operacije ZBRAJANJA između ova dva broja, odnosno između varijabli A i B.

Ovaj program možemo napisati i nesto drugacije. Unesite ove dvije dodatne linije:

```
30 C=A+B
40 PRINT C
```

Sada cemo ispitati vrijednosti varijabli koje koristimo u ovom programu. Radi se o varijablama A, B i C :

PRINT A <CR>
245

-o PRINT B <CR>
76
?C <CR>
321

Vidimo da racunalo nakon izvršavanja programa pamti sve varijable (i numericke i string varijable), tako da ih možemo ispitivati i ispisivati posebno sve dok ne obrišemo program ili programske varijable ili dok ne napisemo neku novu programsku liniju. Napisite ponovo:

PRINT B <CR>
76

Sad unesite novu liniju:
50 END <CR>

i ponovo:

PRINT B <CR>

0

Isto tako, moguce je ispisivati brojeve na ovaj nacin:

PRINT B8 <CR>
88

2. ISPIŠIVANJE STRINGOVA, VRIJEDNOSTI STRING VARIJABLJI I IZRAZA SA STRINGOVIMA

String je, kao sto vec znamo, svaki (i bilo kakav) niz karaktera (slova, brojeva, razlicitih znakova ...). Ispred i iza stringa treba staviti navodnike.

Prije nego sto napisemo neki novi primjer, stari program izbrisite iz memorije pomocu naredbe NEW.

NEW
C=100
B=200
A=300
B+C=500
B*B=40000
A*B=60000
A*B+C=60001

Primjer 1:

```
10 PRINT "MIKRORACUNALO"  
20 PRINT "ORAO"  
RUN  
MIKRORACUNALO  
ORAO
```

Sada ćemo ovaj primjer napisati u malo drugacijem obliku.

Primjer 2:

```
10 A$="MIKRORACUNALO"  
20 B$="ORAO"  
30 PRINT A$:PRINT B$  
RUN  
MIKRORACUNALO ORAO
```

U ovaj smo primjer uveli STRING VARIABLE i dali im imena A\$ i B\$ (citaj "a string" i "be string"). A\$ definirali smo u liniji 10 pomocu stringa "MIKRORACUNALO", a B\$ u liniji 20 pridruzivši mu string "ORAO" (uvocite da se stringovi moraju nalaziti unutar navodnika). Na kraju, linija 30 sadrži naredbe koje ispisuju sadržaj ovih string varijabli.

3. KONTROLNI KARAKTERI KOJI SE KORISTE UZ PRINT

U jednoj naredbi PRINT možemo navesti više podataka koje treba ispisati. Ti se podaci međusobno odvajaju znakovima ";" ili ",".

Promijenite liniju 30 iz prethodnog primjera u:

```
30 PRINT A$;B$  
LIST  
10 A$="MIKRORACUNALO"  
20 B$="ORAO"  
30 PRINT A$;B$  
RUN  
MIKRORACUNALOORAO
```

Tocka-zarez određuje pocetno mjesto ispisivanja i to ODMAH IZA onoga što je prije toga ispisano.

Promijenite liniju 30 koristeci tastere za pomicanje kursora i kopi taster u:

```
30 PRINT "MIKRORACUNALO";  
i dodajte novu liniju:  
40 PRINT B$
```

RUN
MIKRORACUNALO ORAO

Dakle, tocka-zarez koristi se uz PRINT naredbu i sluzi nam zato da pomocu nje uredujujemo ispis na ekranu.

Slicnu ulogu ima i ZAREZ:
Izbrisite liniju 40:

40 {CR}
i promijenite liniju 30 u:
30 PRINT A\$,B\$

RUN
MIKRORACUNALO ORAO

Vidimo da su rijeci odvojene za 8 praznih mesta. Isto se moglo postici i s:

30 PRINT A\$,
40 PRINT B\$

Prije nego sto cete unijeti novi primjer u racunalo, obrisite ovaj pomocu NEW {CR}.

INPUT

Naredba INPUT omogucava nam unos podataka preko tastature pod programskom kontrolom. To znači da se varijablama pomocu naredbe INPUT pridružuju vrijednosti direktno preko tastature i to za vrijeme izvodjenja programa. Kad racunalo u programu nadjе na naredbu INPUT, na ekranu se pojavi upitnik, a racunalo ceka da korisnik unese podatak. Unos podatka zavrsava pritiskom na {CR}.

NUMERICKIM VARIJABLAMA vrijednost se može dodijeliti unesenjem broja, numericke varijable ili numerickog izraza.

STRING VARIJABLAMA vrijednost se pridružuje unesenjem nekog stringa.

Naredba INPUT ne može se koristiti u direktnom, nego samo u programskom nacinu rada.

Primjer:

```
10 INPUT X
20 INPUT A$
30 CLS
40 PRINT "X=";X
50 PRINT "A$=";A$
```

Naredba u liniji 10 znači: unesi jednu numericku varijablu kojoj smo vec u samoj INPUT naredbi pridruzili simbolicku oznaku (X). Znaci, ono sto unesemo s tastature bit će X.

RUN (CR)
Linija 20 znaci da je potrebno upisati, odnosno u racunalo unijeti jedan string koji će biti dodijeljen simbolickoj oznaci za tu varijablu - A\$.

Izvršite program:

```
RUN (CR)
?
racunalo ceka na upis podatka (linija 10)
Unesite neki broj, npr:
1988 (CR)
?
pojavio se i drugi upitnik, racunalo ceka da se upise i neki string. Upisite npr:
GODINA (CR) - nije potrebno pisati navodnike
```

Na ekranu će se sada pojaviti:

```
X=1988
A$=GODINA
```

Primjer 2:

Ovaj primjer pokazuje visestruki unos podataka i to istovremeni unos numerickih i string varijabli. Svaka varijabla u INPUT naredbi mora biti od druge varijable odvojena zarezom.

```
10 INPUT A,B$,C
20 PRINT A,B$,C
```

RUN

? ORAO (CR)

na ekranu se pojavljuje poruka:

? REDO FROM START

?

sto znaci da smo upisali pogresan tip varijable. Racunalo očekuje numericku varijablu, a mi smo joj upisali string. Međutim, nista strasnog se nije dogodilo. Ovaj upis nije prihvacen, pa možemo ponoviti upis, ovaj put pravilno.

Prvi put se unosi broj, onda string, pa onda opet broj.

Primjer:

Naredba INPUT se najčešće koristi tako da se prije unosa ispisuje zeljena poruka, odnosno pitanje. Evo primjera:

```
10 INPUT "TVOJE IME JE";I$
20 PRINT "ZDRAVO";I$
```

RUN <CR>

TVOJE IME JE? PERO <CR>
ZDRAVO PERO

2.6. PETLJE I POTPROGRAMI

Ovdje ćemo se upoznati sa sljedecim naredbama:

IF - THEN
GOTO
FOR - NEXT - STEP
GOSUB - RETURN.
ON

IF - THEN

Nijedan od primjera i programa s kojima smo se dosad susreli nije bio od neke narocite koristi. Od racunala u praksi očekujemo da sam donosi odluke i u skladu s njima reagira. Upravo to, ispitivanje istinitosti određenih uvjeta i grananje (preusmjeravanje) toka programa u zavisnosti od toga da li je određeni uvjet ispunjen ili ne, omogucava nam naredba IF-THEN.

Njen opći oblik je:

IF -uvjet- THEN -xxxxx-

-xxxxx- može biti broj programske linije, neka naredba ili vise naredbi. Ako je uvjet koji se ispituje ISTINIT, tada se izvrsava -xxxxx-.

Ako uvjet NIJE ISTINIT, prelazi se na izvršavanje prve programske linije koja se nalazi iza linije s IF-THEN naredbom.

Naredba IF koristi se u kombinaciji sa sljedecim operatorima:

= jednako
< manje
> veće
<= manje ili jednako
>= veće ili jednako
<> razlicito

Navedene operatore nazivamo operatori relacije (odnosa).

Primjer 1: IGRA POGADJANJA BROJEVA

Ovdje se radi o igri pogadjanja brojeva u kojoj sudjeluju dva igrača. Prvi zadaje bilo koji broj, a drugi ga pogadja. Prepisite sljedeći program vrlo pazljivo.

```

10 REM ----- IZVODJENJE PROGRAMA - I OSNOVNE PROGRAMSKE STRUKTURE
15 REM POGADJANJE BROJEVA
20 REM -----
22 REM ON OVOMO SE OBIJETAKU ZA VREDNOSTI UZIJE
25 CLS
30 INPUT "ZADAJ JEDAN BROJ";Z:CLS
40 INPUT "POGODI ZADANI BROJ";P
50 IF P=Z THEN PRINT "BRAVO, USPJELI STE!":SOUND 100,100:GOTO 90
60 IF P<Z THEN PRINT "PREMALO, POKUSAJ PONOVNO"
70 IF P>Z THEN PRINT "PREVISE, POKUSAJ PONOVNO"
80 GOTO 40
90 INPUT "DA LI ZELIS PONOVNO (DA/NE)";A$
95 IF A$= "DA" THEN 20
99 CLS:PRINT "ZDRAVO !":END

```

BT88

predmet

U liniji 30 unosimo zamisljeni broj Z. U liniji 40 pogodjamo taj broj, odnosno unosimo broj P. Kako se koristi naredba IF-THEN, vidi se iz linija 50, 60 i 70. Ako smo pogodili zamisljeni broj (ako je P=Z), onda se izvrsavaju sve naredbe iz linije 50. Zadnja naredba u liniji 50 je "GOTO 90" sto znači "idi na liniju 90", ili "nastavi s izvodjenjem programa od linije 90". Ako P nije jednako Z (ako broj nije pogodjen), izvrsava se ili linija 60 ili linija 70. Ako je P<Z tada se ispisuje poruka iz linije 60, a poruka iz linije 70 ne, i obrnuto. Nakon toga dolazi se do linije 80: GOTO 40, sto znači da se program bezuvjetno nastavlja od linije 40 u kojoj ponovo, nakon neuspjelog pokušaja, pogodjamo zadani broj Z. Ako smo ovaj puta pogodili broj (P=Z, uvjet iz linije 50 je ispunjen), tada se ispisuje poruka "BRAVO, USPJELI STE!", nakon toga se odsvira jedan ton (SOUND 100,100) i ide se na liniju 90 (naredba GOTO 90 u liniji 50). U liniji 90 odgovaramo na postavljeno pitanje s DA ili NE. Kad upisemo odgovor, on postaje string varijabla A\$ koja se ispituje u liniji 95. Ako je odgovor bio "DA", uvjet u liniji 95 je zadovoljen i program se nastavlja od linije 20, tj. od pocetka. Ako nas odgovor nije bio "DA", izvrsava se linija 99, odnosno zavrsava se izvodjenje programa.

Pomocu IF-THEN naredbe moguce je formirati PETLJU, koja (uz grananja i potprograme) spada u osnovne programske strukture.

Primjer 2: FORMIRANJE PETLJE pomocu IF-THEN naredbe
Napisat cemo program koji ispisuje kvadrate brojeva od 1 do 20:
 10 CLS
 20 N=1
 30 PRINT N;" - ";N*N
 40 N=N+1
 50 IF N<21 THEN 30
 60 END

Formirali smo, dakle, petlju u kojoj se N mijenja od pocetne vrijednosti 1, pa do konacne vrijednosti 20 (povecavamo ga u liniji 40). U liniji 50 provjeravamo da li je N manji od 21 i ako je manji, ide se ponovo na liniju 30, pa 40 i 50, i tako dvadeset puta, sve dok N ne postane veci od 20. Kada N dostigne vrijednost 21, uvjet u liniji 50 nece biti zadovoljen i preci ce se na liniju 60, na zavrsetak izvodjenja programa.

Za vjezbu pokusajte sami preureediti ovaj program tako da cete ispisivati samo kvadrate parnih, odnosno neparnih brojeva.

U primjeru 1 - POGADJANJE BROJEVA susreli smo se s jednom novom naredbom. To je naredba GOTO.

GOTO

Naredbom

GOTO n izvodi liniju n, bezuvjetno skreće na liniju s programskim brojem n. Nakon toga se program dalje izvodi pocevsi od linije na koju je skrenut naredbom GOTO.

Primjer:

```
10 CLS  
20 PRINT "mikroracunalo ORAO"  
30 GOTO 50  
40 PRINT "LINIJA 40 NECE BITI IZVRSENA"  
50 PRINT "PEL VARAZDIN"
```

Ako ste zaboravili kako se "uključuju" mala slova, da vas podsjetimo: pritisnite <PF1>. Isključuju se na isti nacin. Komentar ovom primjeru gotovo da i nije potreban. Ukratko, "GOTO 50" u liniji 30 skreće tok programa na liniju 50, tako da ce linija 40 ostati neizvrserna.

FOR - TO - STEP - NEXT

Petlja je, kao sto smo vec rekli, jedna od osnovnih programskih struktura. Koristi se kada jednu ili vise naredbi treba izvrsiti vise puta zaredom.

Dosad smo se vec upoznali s petljom - formirali smo ju pomocu naredbe IF-THEN. U BASIC-u mikroracunala Orao petlja se moze formirati na mnogo jednostavniji nacin. U tu svrhu koristimo naredbe:

```
FOR - TO - STEP - NEXT
```

```
DO  
NEXT I = sljedeci
```

FOR je prva naredba u petlji koja ima sljedeći oblik:

FOR N=a TO b STEP c

N je ime varijable koja se naziva indeks petlje. Ona se u svakom krugu izvršavanja povećava ili smanjuje onako kako je to definirano naredbama TO i STEP.

INDEKS PETLJE se sastoji od samo jednog (velikog) slova. "a" je pocetna, a "b" je krajnja vrijednost indeksa petlje. "c" je vrijednost koja označava korak promjene indeksa.

Ova tri broja (a,b,c) ne moraju biti cijela, a mogu biti i negativni. Isto tako mogu biti dana u obliku varijabli i numeričkih izraza.

Ako se izostavi "STEP c", korak promjene je 1 (jedan).

Primjer 1:

Pred nama je zadatak da napravimo program za ispis brojeva od 1 do 10 na ekranu. Pogledajmo nekoliko mogućih rješenja.

1. RJESENJE:

```
10 CLS
20 PRINT 1
25 PRINT 2
30 PRINT 3
35 PRINT 4
40 PRINT 5
45 PRINT 6
50 PRINT 7
55 PRINT 8
60 PRINT 9
65 PRINT 10
70 END
```

Složit ćete se ako kazemo da je ovo, vjerojatno, najlošije moguce rješenje. Zamislite da trebamo ispisati sve brojeve od 1 do 1000 !!! Mnogo bolje je:

2. RJESENJE

```
10 CLS
20 N=1
30 PRINT N
40 N=N+1
50 IF N<11 THEN 30
60 END
```

Slijedeće rješenje je, ipak, najlegantnije - uocite prednost upotrebe tzv. FOR-NEXT PETLJE:

3. RJESENJE

```
10 CLS
20 FOR N=1 TO 10
30 PRINT N
40 NEXT N
```

Kad bi morali ispisati brojeve od 10 do 1, program bi izgledao ovako:

```
10 FOR N=10 TO 1 STEP -1
```

```
20 PRINT N
```

```
30 NEXT N
```

Evo još dva primjera:

```
10 FOR N=1 TO 10 STEP 0.5
```

```
20 PRINT N;
```

```
30 NEXT N
```

```
10 FOR N=10 TO 1 STEP .5
```

```
20 PRINT N
```

```
30 NEXT N
```

Evo i dva primjera sa zvukom:

```
10 FOR N=10 TO 250 STEP 5
```

```
20 SOUND N,5
```

```
30 NEXT N
```

```
10 FOR N=250 TO 10 STEP -5
```

```
20 SOUND N,5
```

```
30 NEXT N
```

UPIT

INO Primjer 2:

PR4

Uzmimay, kao primjer, da moramo zbrajati nekakve dosadne
recene i te tako da uvijek zbrajamo po pet brojeva zajedno.
Posao bi si mogli riješiti slijedecim programom:

```
10 CLS
```

```
20 ZB=0
```

```
30 INPUT X1
```

```
40 INPUT X2
```

```
50 INPUT X3
```

```
60 INPUT X4
```

```
70 INPUT X5
```

```
80 ZB=X1+X2+X3+X4+X5
```

```
90 PRINT "ZBROJ=";ZB
```

Slijedecim programom posao zbrajanja nebi si nista
olaksali, ali bi si olaksali posao pisanja programa
(zamislite samo da moramo zbrajati 100 brojeva ...). Dakle:

```
10 CLS
```

```
20 ZB=0
```

```
30 BR=1:REM *** BR je brojac petlje ***
```

```
40 INPUT X
```

```
50 ZB=ZB+X
```

```
60 BR=BR+1
```

```
70 IF BR<=5 THEN 40
```

```
80 PRINT "ZBROJ=";ZB
```

Ovaj problem, kao i sve ostale probleme ovakvog tipa, najlakše ćemo riješiti upotrebom FOR-NEXT petlje:

```
10 CLS
20 ZB=0
30 FOR N=1 TO 5
40 INPUT X
50 ZB=ZB+X
60 NEXT N
70 PRINT "ZBROJ=";ZB
```

Vrlo jednostavnom izmjenom u liniji 30, program bi promijenili tako da zbraja 10,20 ili svejedno koliko brojeva.

Vrlo cesto ćemo unutar jednog programa koristiti nekoliko FOR-NEXT petlji. Međutim, pri upotrebi dvostrukih ili višestrukih petlji moramo biti vrlo oprezni. Petlje se NE SMIJU preklapati, nego moraju biti jedna unutar druge, tj. pocetak i kraj jedne petlje moraju biti izmedju pocetka i kraja druge.

Primjer:

```
10 CLS
20 FOR N=1 TO 10
30 FOR M=1 TO 5
40 PRINT N;
50 NEXT M
60 PRINT
70 NEXT N
```

Dakle, dvije ili više FOR-NEXT petlji moraju biti jedna unutar druge ili potpuna odvojene jedna od druge.

Postoji još jedna vrlo vazna stvar na koju moramo paziti kad radimo s FOR-NEXT petljama. Nije dozvoljeno skociti (npr. s GOTO) na neku od naredbi unutar FOR-NEXT petlje (ako se GOTO nalazi izvan petlje). Znaci, nije dozvoljeno pisati:

```
.....
.....
60 GOTO 80
65 REM
70 FOR X=1 TO 2 STEP .1
80 PRINT X
90 NEXT X
.....
.....
....
```

na ekranu će se pojaviti poruka o greski: NEXT bez FOR.

FOR-NEXT petlja može se koristiti i u direktnom nacinu:

```
FOR V=1 TO 255:SOUND V,5:NEXT V
```

GOSUB - RETURN

U programima se vrlo cesto javlja potreba da se odredjeni zadatak (jedna grupa naredbi) izvrsava više puta. Potreba za time može se javiti u razlicitim dijelovima programa, tako da otpada upotreba FOR-NEXT petlje. U tom slučaju nije potrebno svaki put pisati iste linije programa koje će se nalaziti na razlicitim mjestima u programu. Dovoljno je tu grupu programske linije napisati samo jedanput i izdvojiti ju iz glavnog programa. Takvu izdvojenu programsku cjelinu nazivamo POTPROGRAM. Tako napisan potprogram može se pozvati iz bilo kojeg dijela programa i on će se tada izvršiti. Očito da nema potrebe da u programu imamo nekoliko potpuno jednakih cjelina. Dovoljno je imati samo jednu takvu cjelinu, jedan potprogram, koji ćemo pozvati kad god nam zatreba.

Potprogrami se pozivaju pomoći naredbe:

GOSUB n

gdje "n" predstavlja broj linije na kojoj počinje potprogram. Posljednja naredba potprograma uvijek mora biti:

RETURN

Pomoći ove naredbe se vraćamo iz potprograma - program se nastavlja izvoditi od prve naredbe koja se nalazi iza GOSUB-a.

Primjer 1:

```

10 CLS
20 PRINT "PRVI PUTA":GOSUB 100
30 PRINT "DRUGI PUTA":GOSUB 100
40 PRINT "TRECI PUTA":GOSUB 100
50 END
100 REM *** POTPROGRAM ***
110 FOR N=1 TO 255
120 SOUND N,5
130 NEXT N
140 RETURN

```

Primjer 2:

Napisat ćemo ponovo igru POGADJANJE BROJEVA, s tim da ćemo u njoj koristiti POTPROGRAM.

```

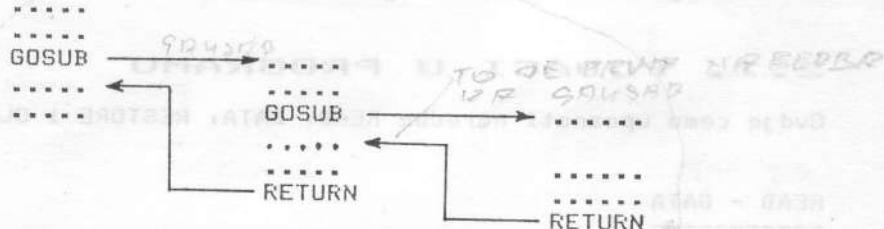
10 REM -----
15 REM POGADJANJE BROJAVA 2
20 REM -----
22 REM
25 CLS
30 INPUT "ZADAJ JEDAN BROJ"; Z:CLS
40 INPUT "POGODI ZADANI BROJ"; P
50 IF P=Z THEN PRINT "BRAVO, USPJELI STE !"
SOUND 100,100:GOTO 90
60 IF P<Z THEN GOSUB 100
70 IF P>Z THEN GOSUB 100
80 GOTO 40
90 INPUT "DA LI ZELIS PONOVO (DA/NE)"; A$
95 IF A$="DA" THEN 20
99 CLS:PRINT "ZDRAVO !":END
100 REM +++
110 PRINT "PROBAJ PONOVO"
120 RETURN

```

Naredba GOSUB ne moze se koristiti bez naredbe RETURN i obrnuto.

Naredba GOSUB se razlikuje od naredbe GOTO po tome sto GOSUB "pamti" broj linije i naredbu s koje se preslo na izvršavanje potprograma. Zbog toga je, nakon zavrsetka potprograma, moguc povratak (RETURN) na prvu sljedecu naredbu iza naredbe s kojom je potprogram bio pozvan (a to je prva naredba iza GOSUB).

Iz jednog potprograma mogu se pozivati drugi potprogrami:



Prilikom pisanja vecih programa, preporucljivo je da se oni podijele na manje cjeline - MODULE. To bi, zapravo, bile grupe potprograma koje bi na kraju povezali u zajednicku cjelinu, u konacan program. Time se postize veca preglednost programa, a mnogo lakse ga je i testirati.

Kao zakljucak navedimo dvije vazne znacajke potprograma:

- dijeljenje programa u module koji mogu biti testirani i izvodjeni odvojeno od glavnog tijela programa cime se povecava razumljivost i preglednost programa
- omogucava da se isti dio programa koristi za iste i/ili slicne probleme, a da se za svaki problem ne pise poseban program.

ON

16 REM
17 REM PROCEDURE GOSUB
18 REM
19 REM
20 REM

Naredba ON koristi se u kombinaciji s naredbama GOTO i GOSUB. Tako se omogucava grananje programa s jednog mesta (iz jedne programske linije) na vise mesta u programu.

ON I GOTO 100,200,300

ako je I=1 izvodi se GOTO 100
ako je I=2 izvodi se GOTO 200
ako je I=3 izvodi se GOTO 300

Isto bi bilo:

ON I GOSUB 100,200,300

Primjer:

```
10 CLS
20 INPUT "UNESI BROJ OD 1 DO 4":X
30 ON X GOSUB 100,200,300,400
40 GOTO 20
100 PRINT "PRVI POTPROGRAM":RETURN
200 PRINT "DRUGI POTPROGRAM":RETURN
300 PRINT "TRECI POTPROGRAM":RETURN
400 PRINT "CETVRTI POTPROGRAM":RETURN
```

Da vas podsjetimo, izvodjenje programa mozete prekinuti s <CTL> i <C>. Za vjezbu pokusajte preraediti ovaj primjer tako da u liniji 30 "GOSUB" zamijenite s "GOTO".

2.7. PODACI U PROGRAMU

Ovdje ćemo upoznati naredbe READ, DATA, RESTORE i CLEAR

READ - DATA

Na pocetku svakog programa obicno se postavljaju, tj. definiraju varijable koje koristimo u tom programu. Mozemo ih postavljati na uobičajeni nacin (A=1, B\$="ORAO"...). Međutim, ako imamo mnogo varijabli, to će bitno povecati duzinu samog programa. U tom slučaju koriste se naredbe READ i DATA.

Primjer:

```
10 A=1:B=2:C=3
20 PRINT A,B,C
```

110 001
Pomocu READ - DATA to isto izgledalo bi ovako:

```
10 READ A,B,C  
20 PRINT A,B,C  
30 DATA 1,2,3
```

Naredbama READ i DATA numerickim i string varijablama takodjer dojeljujemo vrijednosti.

Iza naredbe READ navode se simbolickne oznake varijabli kojima se dodjeljuje vrijednost. One su medjusobno odvojene zarezom.

Iza naredbe DATA nalaze se vrijednosti koje se pridruzuju varijablama. One su takodjer odvojene zarezom.

Prvoj varijabli koja je navedena iza READ (u ovom slucaju A) pridruzuje se prvi broj naveden iza DATA (ovdje 1), dakle to je isto kao da smo napisali A=1. Drugoj varijabli iza READ pridruzuje se drugi broj iza DATA, i tako redom.

Pri tome treba paziti na dvije stvari. PRVO, da se string varijablama pridruzuju stringovima, a numerickim varijablama brojevima. DRUGO, da broj imena varijabli iza READ bude jednak broju vrijednosti (za te variable) koje su navedene iza DATA.

Primjer:

```
10 READ A$,B$,C,D$  
20 PRINT A$;B$;C;D$  
30 DATA MIKRORACUNALO,'ORAO',32,K  
RUN  
MIKRORACUNALO 'ORAO' 32 K
```

Naredba DATA moze se nalaziti bilo gdje u programu. Ona se ignorira sve do izvrsavanja naredbe READ.

Pomocu jedne READ naredbe mogu se "citati" odgovarajuce vrijednosti iz vise DATA naredbi. Kad se procita jedna DATA naredba, prelazi se u drugu. Primjer:

```
10 DATA YUGO,45  
20 DATA AX  
30 READ A$,B,C$  
40 PRINT A$,B,C$  
RUN  
YUGO 45 AX
```

Isto tako, varijablama koje se nalaze iza vise naredbi READ, mogu se pridruzivati vrijednosti iz jedne te iste DATA naredbe. Primjer:

```
10 READ A$,B$  
20 PRINT A$;B$;  
30 READ C  
40 PRINT C  
50 DATA MIKRORACUNALO,ORAO,103
```

Podaci iz DATA naredbe (iz DATA linije) mogu se procitati i pomocu FOR-NEXT petlje. Primjer:

```
10 FOR N=1 TO 5  
20 READ A  
30 PRINT A  
40 NEXT N  
50 DATA 1,3,5,7,9
```

```
348,A BASE 01  
348,A PRINT 05  
E+5,I AT&D 05
```

Kada pokrenete ovaj program, lijepo cete vidjeti kako READ naredba prolazi kroz DATA liniju.

Citanje podataka iz DATA linije pomocu FOR-NEXT petlje koristi se kada zelimo potprogram napisan u strojnom jeziku koji je zapisan kao podaci u DATA liniji, smjestiti u odredjeni dio memorije. Ali, o tome cemo nesto vise reci kasnije i objasniti na prakticnom primjeru.

RESTORE

RESTORE naredba nam omogucava da vrijednosti navedene u DATA liniji, a koje su vec jedanput procitane i pridruzene varijablama, budu ponovo procitane i pridruzene nekim drugim varijablama. Dakle, RESTORE vraca podatkovnu "kazaljku" na pocetak DATA linije.

Primjer:

```
10 READ A,B,C,D  
20 PRINT A;B;C;D  
30 RESTORE  
40 READ X,Y  
50 PRINT X;Y  
60 DATA 1,2,3,4
```

```
348,A BASE 01
```

```
348,A PRINT 05
```

```
MUR
```

RUN
AT&D

1 2 3 4
1 2
KLR OBRISATI
CLEAR = OČISTI MEMRIJU

```
348,BU AT&D 01
```

```
X AT&D 05
```

```
348,B BASE 05
```

```
MUR
```

Napisali smo nekakav BASIC program. On se trenutno nalazi u memoriji racunala i zauzima odredjeni dio memorije, odnosno, kolicina slobodne memorije sada je manja nego sto je bila prije unosenja naseg programa. Kad program pokrenete naredbom RUN, sam program trosi jos i dodatnu kolicinu preostale slobodne memorije u koju, u toku izvrsavanja programa, spremi razne podatke, medjurezultate i pamti stanje i trenutne vrijednosti numerickih i string varijabli.

Naredbom CLEAR brišemo sve varijable i oslobođujemo dio memorije koji se troši za vrijeme izvođenja programa. To znači da se sve numericke varijable postavljaju na vrijednost 0, a sve string varijable na duljinu stringa 0.

Naredba CLEAR koristi se bez dodatnih parametara.

Ponekad se u programu može javiti potreba za brišanjem nekih variabli, ali vrlo rijetko se dogodjava da treba obrišati bus sve varijable. Zato budite oprezni, s upotrebom ove naredbe nemajte pretjerivati. I inace cete CLEAR koristiti vrlo rijetko.

2-8- ARITMETICKE OPERACIJE

BASIC mikroracunala Oroč omogućava izvođenje sljedećih aritmetičkih operacija:

OPERATOR	OPERACIJA
+	zbrajanje
-	oduzimanje
*	množenje
/	dijeljenje
^	potenciranje

Te se operacije izvode izmedju dva broja, a broj može biti zadan i u obliku variabla ili izraza.

Naravno, množenje i dijeljenje imaju visi prioritet, pa će u izrazima biti izvršeni prije zbrajanja i oduzimanja.

Slijedeći primjer racuna cetiri osnovne operacije i potenciranje, i to izmedju dva broja koja unosimo s tastature tako da nam u ovom slučaju Oroč može poslužiti kao kalkulator

```
10 REM =====
15 REM KALKULATOR
16 REM =====
18 REM
20 CLS

30 PRINT:PRINT
40 REM UNOS PODATAKA
50 INPUT "UNESI PRVI BROJ";A
60 INPUT "UNESI DRUGI BROJ";B
70 PRINT
80 PRINT "UNESI OZNAKU OPERACIJE:"
85 INPUT " (+,-,*,,/^)";O$
90 REM ISPITIVANJE I ISPIŠI
100 IF O$= "+" THEN PRINT A;O$;B;"=";A+B: GOTO 30
110 IF O$= "-" THEN PRINT A;O$;B;"=";A-B: GOTO 30
120 IF O$= "*" THEN PRINT A;O$;B;"=";A*B: GOTO 30
130 IF O$= "/" THEN PRINT A;O$;B;"=";A/B: GOTO 30
140 IF O$= "^" THEN PRINT A;O$;B;"=";A^B: GOTO 30
150 REM OPERACIJA NEPOZNATA
160 GOTO 80
```

Ovako bi izracunali koliko je $16*27.2$:

```
16 <CR>
27.2 <CR>
* <CR>
<CR>
```

Osim aritmetickih operacija, BASIC mikroracunala DRAO ima i mogucnost obrade logickih operacija (AND, OR, NOT). S logickim operacijama upoznat cemo se nesto kasnije.

Funkcije i operacije se obavljaju prema PRIORITETU, prvo one s visim, a zatim one s nizim prioritetom.

Prioritet:

- | | | |
|---------|---|--------|
| NAJVISI | 1. formiranje indeksa i podstringova
2. sve funkcije osim NOT i predznak minus
3. potenciranje
4. predznak minus
5. mnozenje i dijeljenje
6. zbrajanje i oduzimanje
7. relacije ($=, >, <, \geq, \leq, \neq$)
8. NOT
9. AND | 10. OR |
|---------|---|--------|

Ostale aritmeticko-logicke operacije tvorimo pomocu osnovnih, pod uvjetom da pazimo na prioritet izvodjenja osnovnih operacija jer u protivnom mozemo dobiti pogresan rezultat.

Za pravilno izracunavanje slozenijih izraza koristimo operatora za grupiranje:

```
( lijeva zagrada  
) desna zagrada
```

Sada cemo razmotriti kako se pojedine aritmeticke operacije i slozeniji izrazi KODIRAJU, odnosno zapisuju u BASIC-u.

Kako bi u BASIC-u zapisali ovaj izraz:

$$\frac{a+b}{c+2}$$

Zapisali bi ga, odnosno izracunali, na sljedeci nacin:

$$(A+B)/(C+2)$$

Pogledajmo jos nekoliko aritmetickih izraza zapisanih prvo na nama uobicajen nacin, a zatim i u obliku kojeg razumije BASIC.

IZRAZ

IZRAZ U BASIC-u

K+3,14159	PRINT 14+13	K+3.14159
a-2,54	PRINT (12+2)(14+3)	A-2.54
b-ca	PRINT (14+2)*(17+3)	B-C*A
xy	Naredba koja izmjenava	X*Y
i:j	Izmjenjivač	I/J
c ²	Brojni izraz	C 2
1	$\frac{4,5 + 14,2 \cdot 19,45}{2,1 - 12,4 / 16,22}$	
x ² + y ²	PRINT(3*(4.5+14.2*19.45)/(2.1-12.4/16.22)) (CR)	1/(X^2+Y^2)
$\sqrt{(a+b)c-1}d$		((A+B)*C-1)*D

✓ U slučaju da izraz u vanjskoj zagradi sadrži još i izraz u unutarnjoj zagradi, prvo će se izvršavati izraz u unutarnjoj zagradi. Broj lijevih zagrada mora biti jednak broju desnih zagrada.

Primjer 1:

Izraz $A^B+C/D-E*F$
je ekvivalentan izrazu $(A^B)+(C/D)-(E*F)$
jer operacije u zagradama imaju visi prioritet od zbrajanja i oduzimanja.

Primjer 2:

Ponekad je potrebno naglasiti redoslijed izvršavanja po prioritetu jednako vrijednih operacija:

PRINT 8/4*2 daje rezultat 4
PRINT 8/(4*2) daje rezultat 1

Prilikom pisanja izraza, morat ćemo se držati sljedećih pravila:

✓ 1. Dva aritmetička operatorka se ne smiju pojaviti jedan iza drugoga:

$A*/B$ nije dozvoljeno

✓ 2. Sve računske operacije moraju biti izrazene eksplicitno:

$(A+B)*(C+D)$ a ne $(A+B)(C+D)$

✓ 3. Izraz a^{bc} je dvomislen, zato izbjegavamo zapis A^B^C . Nedvomisleni zapisi su

$A^{(B^C)}$ sto znači $a^{(b^c)}$

ili

$(A^B)^C$ sto znači $(a^b)^c$

✓ Ova dva izraza nam ne daju jednaki rezultat što nam pokazuje i primjer:

PRINT $2^{(3^2)} = 512$
PRINT $(2^3)^2 = 64$

2.9. POLJA PODATAKA

Naredba DIM

Pretpostavimo da kod sebe imate neki spisak brojeva, npr. sest telefonskih brojeva svojih prijatelja. Da ih spremite u racunalo, morat cete definirati sest novih varijabli, tj. za svaki broj jednu varijablu (A,B,C,D,E,F). Slozit cete se da je to prilicno nespretno (zamislite da se radi o stotinjak brojeva - taj bi program bio prilicno dug i dosadan za upisivanje).

Lakse bi nam bilo kad bi mogli pisati ovako:

```
10 REM OVAJ PROGRAM NE RADI NISTA PAMETNOG
20 FOR B=1 TO 6
30 READ BR
40 NEXT B
50 DATA 4321, 5567, 3932, 4721, 6931, 3297
```

Ali ne mozemo, zato sto se ovi brojevi ne pamte (nakon sto se izvrsi ovaj program, BR ima vrijednost 3297, ali se prethodni brojevi nisu zapamtili).

Ipak je ovu ideju moguce realizirati i to upotrebom POLJA PODATAKA. Polje je skup varijabli (elemenata polja) koje imaju isto ime, a razlikuju se samo po broju unutar zagrade koji стоји uz ime varijable.

U nasem primjeru zajednicko ime za sve varijable neka bude B. Tih sest brojeva imat ce oznake:

- B(0) - prvi broj
- B(1) - drugi broj
- B(2) - treci broj
- B(3) - cetvrti broj
- B(4) - peti broj
- B(5) - sesti broj

Polje podataka mozemo koristiti tek nakon sto u memoriji racunala rezerviramo odredjeni prostor za to polje. To se postize naredbom DIM.

DIM B(5)

rezervirat ce 6 elemenata (od 0 do 5) za polje B. Ti elementi su varijable:

B(0), B(1), B(2), B(3), B(4), B(5)

Program, u kojem cemo sest telefonskih brojeva spremiti u polje podataka, izgledat ce ovako:

```
10 DIM B(5)
20 FOR N=0 TO 5
30 READ B(N)
40 NEXT N
50 DATA 4321,5567,3932,4721,6931,3297
```

Izvršite program i napisite (direktno):

```
FOR N=0 TO 5:PRINT B(N):NEXT N <CR>
4321
5567
3932
4721
6931
3297
```

Polje B u ovom primjeru naziva se JEDNODIMENZIONALNO POLJE. U BASIC-u mikroracunala Draq mogu se dimenzionirati polja s više od jedne dimenzije:

X(A) - JEDNODIMENZIONALNO POLJE
X(A,B) - DVODIMENZIONALNO POLJE
X(A,B,C) - TRODIMENZIONALNO POLJE

gdje su A,B i C INDEKSI POLJA koji određuju pojedini element polja. Elementi polja mogu biti numericke ili string varijable, što određujemo imenom polja (npr. A(5) ili B\$(4))

POLJE PODATAKA predstavlja SKUP INDEKSIRANIH VARIJABLJI koje nazivamo ELEMENTI POLJA.

Primjer 1: FORMIRANJE JEDNODIMENZIONALNOG POLJA X(9)

```
X(0)
X(1)
X(2)
...
      Polje ima (9+1) = 10 elemenata
...
X(9)
```

Primjer 2: FORMIRANJE DVODIMENZIONALNOG POLJA X(2,3)

```
X(0,0)  X(0,1)  X(0,2)  X(0,3)
X(1,0)  X(1,1)  X(1,2)  X(1,3)
X(2,0)  X(2,1)  X(2,2)  X(2,3)
```

Ukupan broj elemenata = (2+1)*(3+1) = 3*4 = 12

Vidimo da dvodimenzionalno polje formiramo prema brze rastucem indeksu 2 i sporije rastucem indeksu 3 (ovo je dvodimenzionalno polje s 2+1 redaka i 3+1 stupaca - zamislite da u ovakvo polje podatke u kojem retku i u kojem stupcu se nalazi određeno slovo na ekranu ...).

Primjer 3: FORMIRANJE TRODIMENZIONALNOG FOLJA X(2,2,2)

Zamislite da, uz podatke o retku i stupcu, trebamo znati i broj stranice.

X(0,0,0) X(0,1,0) X(0,2,0)
X(1,0,0) X(1,1,0) X(1,2,0)
X(2,0,0) X(2,1,0) X(2,2,0)

X(0,0,1) X(0,1,1) X(0,2,1)
X(1,0,1) X(1,1,1) X(1,2,1)
X(2,0,1) X(2,1,1) X(2,2,1)

X(0,0,2) X(0,1,2) X(0,2,2)
X(1,0,2) X(1,1,2) X(1,2,2)
X(2,0,2) X(2,1,2) X(2,2,2)

Ukupan broj elemenata polja X(2,2,2)=27
 $(2+1)*(2+1)*(2+1)=27$

X(0,0,0) - nulti red
nulti stupac
nulta stranica

X(2,1,0) - drugi red
prvi stupac
nulta stranica

X(2,1,2) - drugi red
prvi stupac
druga stranica

Mogu se dimenzionirati polja od jos vise dimenzija (cetiri, pet, sest ...). Međutim, organizacija podataka u takvima poljima je veliki problem, jer se barata s velikom kolicinom podataka, a sve sto ima vise od tri dimenzije, covjeku je tesko zamisliti.

Primjer:

Zadatak je sljedeći: za svaki mjesec u godini trebamo upamtiti dva podatka, srednju i maksimalnu temperaturu.

Podatke bi mogli organizirati na ovaj nacin:

MJESEC - SREDNJA TEMPERATURA - MAKSIMALNA TEMPERATURA
ili konkretno:

SIJECANJ - 1.4 - 6.3 - redak 1

VELJACA - 5.5 - 11.2 - redak 2

...

...

PROSINAC - 2.1 - 5.4 - redak 12

↑ ↑ ↑

stupac 1 stupac 2 stupac 3

Kada smo ovako predstavili problem, lakše je uociti da se radi o polju podataka koje ima 12 redaka (za svaki mjesec po jedan) i 3 stupca (u prvom stupcu zapisan je mjesec, u drugom srednja, a u trećem maksimalna temperatura).

Kako da rezerviramo potreban prostor u memoriji za polje od 12 redova i 3 stupca? Prisjetimo se onoga što smo malo prije naučili - radi se o naredbi DIM. U ovom slučaju radi se o dvodimenzionalnom polju (imamo redove i stupce; kad bi imali još i stranice, polje bi bilo trodimenzionalno). U to polje ćemo zapisivati i stringove (nazive mjeseca) i brojeve (temperature). Kad bi otvorili polje za numeričke varijable, u njega ne bi mogli zapisivati nazive mjeseca. Zato ćemo otvoriti polje za string varijable. Brojevi koje ćemo upisivati u polje racunalo će shvatiti kao stringove, a za nas to nije vazno. Vazno je da ih racunalo zapamti.

Polje ćemo dimenziorati pomocu naredbe DIM T\$(11,2). Zasto bas DIM T\$(11,2)?

T\$ je ime polja koje smo proizvoljno odabrali. U ovom slučaju vazno je da se u imenu nalazi znak "\$" koji označava da se radi o string varijablama.

"(11,2)" bi prokomentirali ovako: u polju se nalazi (11+1) redova i (2+1) stupaca (indeksi pocinju od 0 !!!).

Evo i konačnog programa. Podaci se prvo uzimaju iz DATA linija i smjestaju u dvodimenzionalno polje podataka. Nakon toga se iz tog polja citaju i uredno ispisuju na ekran.

```
5 REM MJESECNE TEMPERATURE
10 CLS
20 DIM T$(11,2)
25 REM CITANJE NAZIVA MJESECA
30 FOR N=0 TO 11
40 READ T$(N,0)
50 NEXT N
55 REM CITANJE SREDNJE TEMPERATURE
60 FOR N=0 TO 11
70 READ T$(N,1)
80 NEXT N
85 REM CITANJE MAKSIMALNE TEMPERATURE
90 FOR N=0 TO 11
100 READ T$(N,2)
110 NEXT N
120 REM ISPIS PODATAKA IZ FOLJA
130 PRINT "MJESEC", "SREDNJA ", "MAKSIMUM"
140 PRINT:PRINT
150 FOR N=0 TO 11
160 PRINT T$(N,0), T$(N,1), T$(N,2)
170 NEXT N
200 DATA SIJECANJ, VELJACA, OZUJAK, TRAVANJ, SVIBANJ, LIPANJ
210 DATA SRPANJ, KOLOVODZ, RUJAN, LISTOPAD, STUDENI, PROSINAC
220 DATA 1.4, 5.5, 12.3, 15.8, 18.6, 23.5, 28.1, 26.4, 21.3,
15.3, 6.2, 2.1
230 DATA 6.3, 11.2, 16.5, 19.1, 25.7, 30.4, 34.9, 30.3, 27.5,
20.2, 9.3, 6.4
```

Ako ste točno prepisali sve linije programa (pripazite na DATA linije), izvršite ga s RUN. Na ekranu će se pojaviti tri kolone: MJESEC, SREDNJA i MAKSIMUM s podacima.

Do linije 50 formiran je nulti stupac polja, tj. svih 12 redova polja sadrži podatak u nultom stupcu:

T\$(0,0) = SIJECANJ	- nulti redak, NULTI STUPAC
T\$(1,0) = VELJACA	- prvi redak , NULTI STUPAC
.....
.....
T\$(11,0)= PROSINAC	- dvanaesti redak, NULTI STUPAC

Linije 60,70 i 80 formiraju PRVI STUPAC POLJA T\$. Obratite pažnju na naredbu READ T\$(N,1) - N (redak) se mijenja od 0 do 11, a 1 označava prvi stupac). Polje zasada izgleda ovako:

T\$(0,0) = SIJECANJ	T\$(0,1) = 1.4
T\$(1,0) = VELJACA	T\$(1,1) = 5.5
....
....
T\$(11,0)= PROSINAC	T\$(11,1)= 2.1

Konacno, linijama 90,100 i 110 formiramo i TRECI STUPAC POLJA (READ T\$(N,2)).

Konacan izgled polja je ovakav:

T\$(0,0) = SIJECANJ	T\$(0,1)=1.4	T\$(0,2)=6.3
T\$(1,0) = VELJACA	T\$(1,1)=5.5	T\$(1,2)=11.2
....
....
T\$(11,0)= PROSINAC	T\$(11,1)=2.1	T\$(11,2)=6.4

Formirano je, dakle, cijelo polje T\$. Ono se sastoji od 12 redaka i tri stupca i ima $(11+1)*(2+1)=36$ podataka koji se nalaze u DATA linijama i koje smo nasim programom preselili u polje T\$ u kojem svaki od elemenata ima dva indeksa: indeks retka i indeks stupca. Pogledajmo koji se element nalazi u sedmom retku i drugom stupcu:

```
PRINT T$(6,2)  
28.1
```

Obratite pažnju na linije 140, 150 i 160. Jednom FOR-NEXT petljom ispisali smo na ekran sadržaj cijelog polja. Ostatak programa su, u ovom slučaju neophodne, DATA linije.

Ako vas zbunjuje kako polje T\$(11,2) ima 12 redova i 3 stupca, otvorite polje T\$(12,3) i nemojte koristiti indeks 0.

Probajte za vježbu rjesiti ovaj isti zadatak, ali tako da podatke smjestite u polje s obrnutim brojem redaka i stupaca, da otvorite polje T\$(2,11).

Kako cete sada citati podatke iz DATA linija? Za svaki stupac posebno, a ima ih 12, vrlo je nespretno. Kako bi, umjesto s 12 FOR-NEXT petlji, obavili posao citanja s 3 petlje? Probajte citanjem podataka iz DATA linija formirati redove polja.

Drugi zadatak bi mogao glasiti: izbacite iz programa sve DATA linije. Napisite isti program, ali tako da podatke unosite s tastature pomocu naredbe INPUT.

Ako ste pazljivo proučili sve što smo rekli o poljima podataka, vjerujemo da vam ovi zadaci neće predstavljati nerjesive probleme.

2.10. STRING FUNKCIJE

ASC, CHR\$, LEFT\$, RIGHT\$, MID\$, LEN, STR\$, VAL

Pojam "STRING" već smo dobro upoznali. Rekli smo da je string niz bilo kakvih znakova, koji mogu biti:

- ✓ - slova
- ✓ - znamenke
- ✓ - specijalni (posebni) znakovi

Stringovi se moraju nalaziti unutar navodnika. Maksimalna duljina stringa kod mikroracunala Orao iznosi 255 znakova.

Stringovi jedino ne mogu biti kontrolni znakovi, npr. <CTL>, <CTL> i <L>, <CR> ... String može biti uključen u kontekstu naredbi PRINT ili INPUT, dok string varijable moraju obavezno na kraju imena sadrzavati znak "\$". Znak "\$" predstavlja identifikacijski znak za stringove, a zovemo ga i "string operator".

Kao što s brojevima vrsimo razlike operacije, tako ih možemo vrsiti i sa stringovima. Ali, prije nego se u to upustimo, potrebno je da saznate na koji nacin mikroracunalo Orao pamti slova, brojeve i ostale znakove.

Računar
Svaki znak ima svoj KOD, odnosno BROJ kojeg racunalo pamti umjesto tog slova. Memoriju racunala zamislite si kao tisuće praznih ladica - u svaku od tih ladica može se smjestiti SAMO JEDAN broj, jedan kod. U memoriji mikroracunala Orao nalazi se 23534 praznih ladica, a svaka od njih predstavlja jednu memoriju lokaciju. Svaka lokacija ima svoj (kucni) broj, svoju ADRESU. Svi podaci koje unosimo u racunalo (slova, programi, znakovi ...), svi oni se u memoriji racunala nalaze spremljeni kao brojevi, razni kodovi. U jednu memoriju lokaciju (u jednu ladici) možemo staviti samo jedan broj, a taj broj može biti najmanje 0, a najviše 255. Kad bi htjeli u memoriju racunala spremiti broj 256 (ili bilo koji drugi veci od 255, a manji od 65536), taj bi nam broj zauzeo dvije memorije lokacije (dvije ladice), itd ... Rekli smo da svakom karakteru (slovu, broju ili drugom znaku) pripada jedan kod, u rasponu od 0 do 255. Kodovi od 0 do 31

rezervirani su za specijalne znakove (kontrolne kodove). Kodovi od 32 do 127 predstavljaju znakove, velika i mala slova, a kodovi od 128 do 159 rezervirani su za znakove koje korisnik može sam definirati. Kodovi od 160 do 255 su kodovi inverznih znakova, brojeva i slova.

BASIC mikroracunala Orao sadrži nekoliko naredbi (funkcija) koje vrše razne operacije sa stringovima (pretvaranje karaktera u kod i obrnuto, ispitivanje duzine stringova, njihovo usporedjivanje ...).

USPOREDJIVANJE STRINGOVA

4PLC

Nakon ovog kratkog uvoda vjerojatno ste shvatili kako Orao pamti znakove. Sada vam neće biti problem ni da shvatite kako se usporedjuju stringovi.

Ako napisemo "IF A\$>B\$" tada Orao neće usporedjivati duzine stringova A\$ i B\$, kako bi mnogi mogli pomisliti. Racunalo će uzeti prvi znak iz A\$ i prvi znak iz B\$, pa će usporediti njihove kodove. Ako su kodovi razliciti, ispituje se koji je od njih veci, nakon cega je usporedjivanje stringova A\$ i B\$ završeno. Ako su kodovi jednaki, uzima se slijedeci znak iz svakog stringa. Ako se ispitivanjem kodova prvih karaktera iz stringova A\$ i B\$ utvrdi da je kod iz B\$ veci, to ujedno znači da je B\$ veci od A\$.

Slova su, to svi znamo, poredana po abecedi. Tim su redom poredani i njihovi kodovi. Tako slovo A ima kod 65, B je 66, C je 67, D je 68 ...

Međutim, ovdje izrazi "veci" i "manji" mogu dovesti do zabune, pa se umjesto izraza "veci" koristi izraz "SLIJEDI", a umjesto "manji" izraz "PRETHODI". Tako da:

string "ANA" je manji od stringa "BETI", odnosno string "ANA" prethodi stringu "BETI".

Primjer 1:

```
10 A$= "AUTO"
20 B$= "MOTO"
30 C$= "AUTO"
40 IF A$=C$ THEN PRINT "A$=C$"
50 IF A$>B$ THEN PRINT "A$>B$"
60 PRINT "B$>A$"
```

Primjer 2: ZBRAJANJE STRINGOVA

```
10 A$= "MIKRORACUNALO"
20 B$= "ORAO"
30 PRINT A$+B$
40 C$= A$+" "+B$
50 PRINT C$
```

Na ovom jednostavnom primjeru vidimo da ZBRAJANJE STRINGOVA nije nista drugo nego njihovo spajanje, nadovezivanje.

ASC funkcija

Napisite :

PRINT ASC ("A") <CR>

Rezultat je 65, a to je KOD slova "A", odnosno broj koji predstavlja slovo "A".

Fokusajte sad:

A\$="A":PRINT ASC (A\$) <CR>

Rezultat je ponovo 65 jer se opet radi o kodu slova "A".

U praksi se funkcija ASC ne koristi cesto za ovakvo ispisivanje kodova pojedinih znakova. Cesce se koristi da nekoj numerickoj varijabli dodijelimo onu vrijednost koja predstavlja kod prvog znaka u stringu. Na primjer:

```

10 A=ASC("A")
20 B=ASC("B")
30 C=ASC("CIKLA")
40 PRINT A,B,C
RUN
65      66      67

```

Isti program mogli smo napisati i ovako:

```

NE
10 A$="A":A=ASC(A$)
20 B$="B":B=ASC(B$)
30 C$="CIKLA":C=ASC(C$)
40 PRINT A,B,C

```

Argument funkcije ASC mora biti u zagradi, npr A=ASC(A\$). Ako se radi o tekstu, on mora unutar zagrade biti u navodnicima, npr C=ASC("CIKLA"). Nije dozvoljeno traziti ASC od praznog stringa.

CHR\$ funkcija

Ova funkcija ima suprotan efekt od funkcije ASC.
Probajte:

PRINT CHR\$(65) <CR>

2 PRH Rezultat je "A". Dakle, funkcija CHR\$ pretvara decimalnu vrijednost koda nekog karaktera (u taj karakter, odnosno tvori string od jednog karaktera.) KOD toga znaka je argument funkcije CHR\$.

Osim sto pretvara kodove u stringove, ova funkcija na mikroracunalu DRAO ima neke mnogo značajnije primjene. Reklamiramo da znakovi ciji su kodovi od 0 do 31 imaju posebnu primjenu, a "ispisivanje" nekog od tih znakova može imati utjecaja na izvodjenje programa. Ove znakove (nazivaju se KONTROLNE FUNKCIJE) možemo unijeti direktno preko tastature istovremenim pritiskom na taster (CTL) i odredjeno slovo.

Ako zelimo upotrijebiti neki od ovih znakova unutar programa, posluzit ćemo se funkcijom CHR\$. Primjer: naredbu CLS vec poznajemo. Isto bi bilo kad bi napisali:

PRINT CHR\$(12)

Ovdje su napisani neki od najčešće upotrebljavanih kontrolnih znakova i njihovo značenje:

PRINT CHR\$(4)	- pomice kurzor na vrh ekrana pri čemu se sadržaj ekrana ne briše
PRINT CHR\$(5)	- briše red ispod kursora
PRINT CHR\$(6)	- briše ekran od kursora na dolje
PRINT CHR\$(7)	- proizvodi kratak zvučni signal
PRINT CHR\$(9)	- pomice kurzor za jedno mjesto UDESNO
PRINT CHR\$(10)	- pomice kurzor za jedno mjesto DOLJE
PRINT CHR\$(11)	- pomice kurzor za jedno mjesto GORE
PRINT CHR\$(12)	- briše ekran
PRINT CHR\$(13)	- vraća kurzor na početak reda (CR)
PRINT CHR\$(31)	- pomice kurzor za jedno mjesto LIJEVO

Sve znakove kojima raspolaže mikroracunalo Orao ispisat ćemo slijedecim programom:

```
10 FOR N=32 TO 255  
20 PRINT CHR$(N);  
30 NEXT N
```

Kontrolna varijabla N u FOR-NEXT petlji ima početnu vrijednost 32 zato sto znakovi od 0 do 31 služe za posebne funkcije.

LEN
====

Funkcijom LEN(X\$) nekoj numeričkoj varijabli pridružujemo broj koji je jednak broju znakova stringa X\$ (duzini stringa X\$).

Primjer:

```
10 A$="PROGRAM"  
20 X=LEN(A$)  
30 PRINT X  
RUN  
7
```

7 je broj znakova u stringu "PROGRAM" ne računajući navodnike.

Ako u zagradi navodimo tekst umjesto imena stringa, tada taj tekst mora biti unutar navodnika. Primjer:

```
10 X=LEN("PROGRAM")  
20 PRINT X  
RUN  
7
```

Funkcija LEN koristi se najčešće u kombinaciji s nekom od funkcija koje ćemo tek upoznati (LEFT\$, RIGHT\$, MID\$). To su funkcije koje imaju više argumentata, a sve tri se koriste za stvaranje PODSTRINGOVA od nekog GLAVNOG STRINGA. Ako imamo, na primjer, string "MIKROKOMPJUTER", njegovi bi podstringovi bili "MIKRO", "MIKROKOM", "KROKO", "OKO", "JUTER", i slično.

Ponekad je, dakle, potrebno iz glavnog stringa izdvajiti neki njegov dio, koji tada zovemo PODSTRING.

LEFT\$

=====

Primjer:

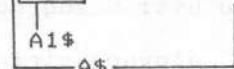
```
10 A$="MIKROKOMPJUTER"
20 A1$=LEFT$(A$,5):REM A1$ je PODSTRING od A$
30 PRINT A1$
```

RUN

MIKRO

Funkcijom LEFT\$(A\$,5) tvorimo podstring stringa A\$ duzine pet znakova i to počevši od LIJEVE strane.

MIKROKOMPJUTER



- A1\$=LEFT\$(A\$,5)

Ovaj program tvori podstring od prvih pet znakova iz glavnog stringa "MIKROKOMPJUTER". Ako navedemo duzinu veću od duzine glavnog stringa, funkcija LEFT\$ će uzeti cijeli glavni string:

```
10 PRINT LEFT$("ORAO",6)
```

RUN

ORAO

Vidimo da argument funkcije LEFT\$ može biti i tekst, ali ga onda trebamo pisati unutar navodnika.

RIGHT\$

=====

RIGHT\$ i LEFT\$ su vrlo slične funkcije i imaju istu sintaksu. Funkcija RIGHT\$ isto tako formira podstring glavnog stringa, samo što se kod funkcije RIGHT\$ znakovi uzimaju počevši od desne strane glavnog stringa. Primjer:

```
10 A$="MIKROKOMPJUTER"
20 A2$=RIGHT$(A$,9)
30 PRINT A2$
```

RUN

KOMPJUTER

RIGHT\$(A\$,9) uzima zadnjih devet slova iz rijeci "MIKROKOMPJUTER" i tvori podstring A2\$="KOMPJUTER".

Kad bi naveli duzinu vecu od duzine glavnog stringa, podstring bi bio jednak glavnom stringu.

U svim operacijama koje vrse operacije nad stringovima, duzina stringa ne smije biti veca od 255 znakova.

MID\$
====

Primjer:

10 B\$="ELEKTRONIKA"
20 C\$=MID\$(B\$,5,4)
30 PRINT C\$

RUN
TRON

Funkcija C\$=MID\$(B\$,5,4) tvori podstring glavnog stringa "ELEKTRONIKA" i to pocevsi od petog znaka i duzine cetiri znaka, znaci od T do N (TRON).

Funkcija MID\$ ima tri argumenta. Prvi je ime glavnog stringa (B\$), a druga dva su numericki argumenti. Prvi od njih označava od kojeg se po redu znaka pocinje tvoriti podstring, a koje ce duzine on biti, to odredjuje drugi numericki argument. Sva tri argumenta moraju biti u zagradi i medjusobno odvojeni zarezom.

Unutar zagrade mozemo, umjesto imena glavnog stringa, navesti i tekst, ali obavezno unutar navodnika:

PRINT MID\$("TELEVIZIJA",3,4)

LEVI

Prvi numericki argument ne smije biti manji od 1, a kad bi bio veci od duzine glavnog stringa, kao rezultat bi dobili prazan string.

Dруги numericki argument ne smije biti manji od nule. Ako je nula, rezultat ce biti prazan string, a ako je veci od duzine preostalog dijela glavnog stringa, rezultat ce biti desni dio glavnog stringa pocevsi od znaka kojeg je odredio prvi argument.

Funkciju MID\$ mozemo korisno upotrijebiti da pretrazimo neki string i ispitamo da li se u njemu nalazi neki znak ili neki drugi string. Sljedeci primjer ispituje da li se u stringu M\$="TELEVIZIJA" nalazi string N\$="VIZ":

```

10 M$="TELEVIZIJA"
20 N$="VIZ"
30 N=LEN(N$): M=LEN(M$)
40 FOR X=1 TO M-N
50 T$=MID$(M$,X,N)
60 IF T$=N$ THEN PRINT X
70 NEXT X
RUN
5
sto znači da se string "VIZ" nalazi unutar stringa
"TELEVIZIJA" i da pocinje od petog znaka.

```

STR\$

====

Vise puta javlja se potreba da se neki broj, koji predstavlja vrijednost neke numericke varijable, pretvori u string. U tu svrhu koristimo funkciju STR\$:

```

10 INPUT X: REM UPISI BROJ
20 X$=STR$(X)
30 PRINT "X=";X
40 PRINT "X$=";X$
RUN
? 25
X= 25
X$= 25

```

Funkcija STR\$ koristi se i kad zelimo na neki string nadovezati broj (broj pretvorimo u string i stringove zbrojimo, odnosno spojimo), ili kad zelimo urediti ispis brojeva na ekranu, zbog pravilnog potpisivanja brojeva. Ako bi htjeli pravilno potpisivati brojeve, broj trebamo pretvoriti u string, "izmjeriti" mu duzinu i, ovisno od toga kolika je duzina "najduzeg" broja, dodati mu na pocetku odredjen broj praznih mjesto:

```

10 CLS
20 FOR N=5 TO 150 STEP 5
30 N$=STR$(N)
40 IF LEN(N$)<5 THEN N$="- "+N$:GOTO 40
50 PRINT N$
60 NEXT N

```

U liniji 40 ispituje se duzina stringa N\$; ako je manja od 5 dodaje mu se toliko puta jedno prazno mjesto dok duzina stringa ne bude popunjena do 5 s praznim mjestima (a zbog potpisivanja je vazno s koje strane mu se dodaju prazna mesta). Na ovaj nacin smo uredili ispis brojeva.

VAL funkcija

Suprotni efekat od funkcije STR\$ postize se upotrebom funkcije VAL. Ona pretvara karaktere stringa u odgovarajuću numericku vrijednost. Podrazumijeva se da sadržaj stringa mora biti numerički - dozvoljava se upotreba znamenki od 0 do 9 i decimalne tocke. Na primjer:

```
10 A$="123.5"
20 A=VAL(A$)

30 PRINT A
RUN
123.5

10 A$="99.17"
20 PRINT A
RUN
99.17

10 A$="ORAO"
20 A=VAL(A$)
30 PRINT A
RUN
0

10 X$="32. UDARNA"
20 X=VAL(X$)
30 PRINT X
RUN
32
- uzima se u obzir kompletan dio numerickog dijela
stringa do prvog nenumerickog podatka

10 Q$="ORAO 103"
20 PRINT VAL(Q$)
RUN
0
- kad se nađe prvo na nenumericki podatak, rezultat je
nula (0).
```

2.1.1. LOGIČKE OPERACIJE

Naredba IF-THEN vec nam je dobro poznata. U njoj smo ispitivali da li je neki uvjet zadovoljen (istinit) ili nije i na temelju toga donosili odluke i vrstili grananja u nasem programu. Susreli smo se vec i s operatorima odnosa - relacijskim operatorima ($<$, $>$, $=$, $<=$, $>=$) koji su usporedjivali dva broja ili stringa. Međutim, BASIC nam omogućava povezivanje nekoliko uvjeta koji međusobno moraju biti povezani LOGICKIM OPERATORIMA, a to su operatori AND, OR i NOT (I, ILI i NE).

Ako imamo uvjet IF X=3 THEN CLS

da,
jasno je će se naredba CLS izvršiti samo onda ako X ima vrijednost 3. Ako je X razlicit od 3, naredba CLS se ne izvodi, vec program skace na prvu sljedecu programsku liniju. Ali kad bi napisali:

IF X=3 AND Y=4 THEN CLS

naredba CLS bi se izvršila samo u slučaju da je X=3 i Y=4, dakle samo onda ako su zadovoljena oba dva uvjeta. Proucite sljedeći primjer: od tri broja koja mu na pocetku upisemo, racunalo će ispisati najvećeg od njih:

```
10 INPUT A,B,C  
20 IF A>B AND A>C THEN PRINT A;  
30 IF B>A AND B>C THEN PRINT B;  
40 IF C>A AND C>B THEN PRINT C;  
50 PRINT " JE NAJVECI"
```

Pomoći AND možemo povezati i više uvjeta, a osim brojeva možemo usporedjivati i stringove, npr.:

IF A\$="DA" AND B=9 AND Y>=100 THEN ...

Povodom li dva ili više uvjeta logičkim operatorom OR (OR=ILI), dovoljno je da samo jedan od njih bude istinit pa da konacni uvjet bude zadovoljen, tj. da se izvrši naredba

iza THEN, npr.:

```
IF A$="DA" OR X=1 THEN CLS
```

Da bi se izvršila naredba CLS, dovoljno je da samo jedan od uvjeta bude zadovoljen - ili da je A\$="DA" ili da X bude 1, uz napomenu da će se CLS izvršiti i ako su oba dva uvjeta ispunjena. Zapamtiti treba da je, u slučaju kad koristimo logički operator OR, dovoljno da BAR JEDAN od uvjeta bude zadovoljen pa da konacan uvjet bude zadovoljen.

Posljednji logički operator je NOT (NOT=NE). Objasnimo ga na primjeru:

```
10 X=2  
20 IF NOT X=1 THEN PRINT "UVJET JE ISPUNJEN"  
RUN  
UVJET JE ISPUNJEN
```

Liniju 20 procitali bi: ako X nije 1, onda ispisi ... Ucinak operatora NOT na pojedine operatore odnosa je ovakav:

OPERATOR ODNOSA	LOGICKI OPERATOR NOT
=	<>
>	<=
<	>=
>=	<
<=	>
<>	=

Dozvoljeno je i ovo:

```
IF X>0 AND Y<10 OR Z$="NE" THEN PRINT "UVJET ISPUNJEN"
```

Najvisi prioritet pri izvršavanju logičkih operacija ima NOT. Iza njega je AND i na kraju OR. To znači da bi se prethodni primjer:

```
IF X>0 AND Y<10 OR Z$="NE" THEN PRINT "UVJET ISPUNJEN"
```

izvršavao ovim redom:

najprije će se izvesti AND izmedju $X > 0$ i $Y < 10$. Nakon toga izvodi se OR izmedju prethodnog zakljucka i uvjeta da li je Z = "NE"$. Dakle, ovaj slozeni uvjet bit će zadovoljen ako je $X > 0$ I $Y < 10$ ILI je Z = "NE"$.

Redoslijed izvodjenja logičkih operacija možemo izmijeniti upotrebom ZAGRADA. Njih koristimo kod kreiranja slozenih logičkih uvjeta:

```
IF X>0 AND (Y<10 OR Z$="NE") THEN PRINT "UVJET ISPUNJEN"
```

Tekst će se ispisati samo ako je $X > 0$ i ako je barem jedan od uvjeta u zagradi istinit.

Osim što služe za povezivanje uvjeta, logički operatori se, doduse vrlo rijetko, koriste i za izvodjenje logičkih operacija izmedju brojeva. Radi se o BINARNIM LOGICKIM OPERACIJAMA. One se izvode s brojevima prikazanim u binarnom obliku.

Tabela istinitosti 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 1 - KONJUNKCIJA

```
10 A=175:B=250
```

```
20 Z=A AND B
```

```
30 PRINT Z
```

```
RUN
```

```
170
```

ili A = 10101111 = 175 decimalno
 B = 11111010 = 250 decimalno

A AND B = 10101010 = 170 decimalno

Primjer 2 - DISJUNKCIJA

```
10 A=65:B=128
20 Z=A OR B
30 PRINT Z
RUN
193
ili      A = 01000001      = 65 decimalno
          B = 10000000      = 128 decimalno
-----
          A OR B = 11000001      = 193 decimalno
```

Primjer 3 - NEGACIJA

```
10 A=254
20 Z=NOT A
30 PRINT Z
RUN
1
-255
ili      A = 11111110      = 254 decimalno
-----
NOT A = 00000001      = 1 decimalno
```

Slijedecim primjerom cemo pokazati kako je moguce alfanumericki znak programski prikazati na ekranu u invertiranom obliku:

Primjer 4 - inverzija znaka

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

2.12. ORGANIZACIJA EKRANA

Vjerujemo da ste se s naredbom PRINT dosad vec dobro upoznali. U svakom od primjera koje smo dosad upoznali nalazi se bar jedan PRINT ... Dosad smo naucili sto se dogadja ako iza PRINT stoji zarez ili tocka-zarez, ili pak dvotocka. Međutim, jednu vrlo vaznu stvar jos nismo naucili. Radi se o tome kako je organiziran prikaz teksta (karaktera) na ekranu, od koliko se redova sastoji slika na ekranu i koliko slova mozemo napisati u jednom (koliko ima kolona na ekranu).

Napisite:

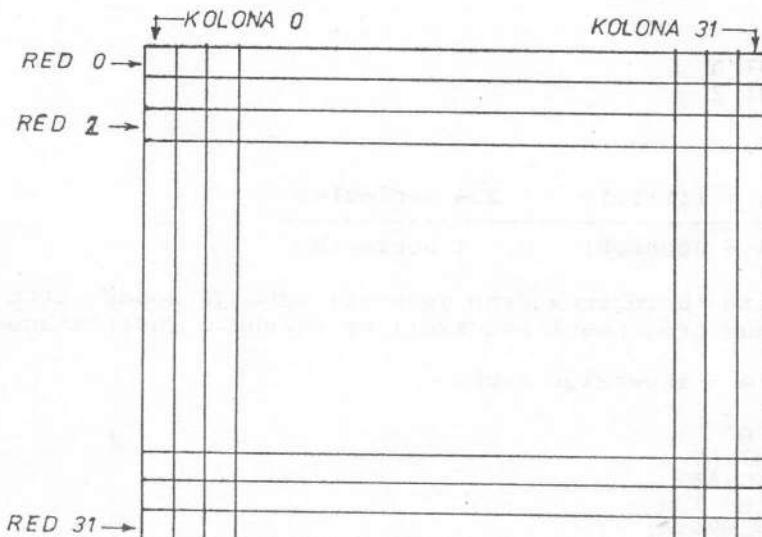
```
FOR X=1 TO 35:PRINT"X";:NEXT X <CR>
```

Jedan cijeli red ekranu se ispunio sa slovom X, a u drugom redu imamo samo tri slova X. Iz toga mozemo zaključiti da u jednom redu mozemo napisati najvise 32 znaka. Kazemo da

postoje 32 KOLONE na ekranu. Zanima nas još samo koliko redova može stati na ekran. Odgovor je - isto 32.

Možemo, dale, zaključiti: mikroracunalo Orao na ekranu istovremeno može prikazati 32 alfanumerička znaka u svakom od, ukupno, 32 reda, tj. da ima prikaz teksta u 32 reda i 32 kolone. Najgornji red na ekranu je NULTI red, ispod njega je prvi red, a posljednji red je 31. red. Isto je tako i s kolonama: prvi lijevi znak na ekranu nalazi se u NULTOJ koloni, slijedeći je u prvoj, a posljednji u 31. koloni.

Ekran mikroracunala Orao možete zamisliti kao koordinatni sistem s ishodistem u gornjem lijevom uglu i rasponom vrijednosti 32 po osi X i 32 po osi Y. To izgleda ovako:



SVEKA POGLAVJE
4 OBILIKU PRIPRAVE
SRADITI I ND
KRATKE VJEŽBE
- MRKO
- OSRADO
- VJEŽBE
- ZADACI

Poznavanje organizacije ekrana je vrlo vazna stvar i to znanje cete obilno primjenjivati u vashim programima. Poznavajući ekran Orla, lako cete uređiti sva potpisivanja brojeva, riječi, poruka, centrirati naslove u sredinu reda. Pisat cete po ekranu kao olovkom po papiru, kako i gdje cete htjeti, a sve sa ciljem sto efektnijeg izgleda rezultata vaseg programa

Predjimo sada na konkretnе primjere. U ovom dijelu priručnika cete se upoznati sa OSAM novih naredbi koje nam služe da pomocu njih uređujemo ispisne na ekran.

Jednu od ovih osam naredbi vec ste upoznali. To je naredba CLS. Pomocu nje se briše sadržaj ekranu i kursor se postavlja na vrh ekranu. Isto se postize i sa <CTL> i <L>, samo sto se CLS može koristiti u programu, a <CTL>+<L> ne.

CUR

====

Naredba CUR sluzi za pozicioniranje kursora na ekranu.

Time odredjujemo novu poziciju kursora, a to je ujedno mjesto na ekranu na kojem ce se pojaviti tekst ispisani naredbom PRINT ili INPUT.

Probajte napisati:

CUR 15,5: PRINT "ORAO" <CR>

Ovime smo napisali rijec "ORAO" u petom redu, pocevsi od petnaeste kolone. Dakle :

CUR K,R

postavlja kurzor u K-tu kolonu u R-tom redu ekrana.

Probajte:

10 CLS
20 CUR 5,10: INPUT "KAKO SE ZOVES";A\$:CLS
30 CUR 5,10: PRINT "ZDRAVO ";A\$

ili

10 CLS
20 FOR N=0 TO 20
30 CUR N,N
40 PRINT "ORAO"
50 NEXT N
60 FOR X=20 TO 0 STEP -1
70 CUR X,20-X
80 PRINT "ORAO"
90 NEXT X

TAB funkcija

=====

Funkcija TAB (X) postavlja kurzor na X-tu kolonu brojeci od pocetka reda, odnosno odredjuje mjesto na kojem ce se ispisati prvi ispis pocevsi od pocetka reda. Na primjer:

50 PRINT TAB(10); "ORAO"

ce ispisati "ORAO" tako da ce prvo slovo "O" biti ispisano na desetom mjestu od pocetka reda.

Prvi znak u redu nalazi se u nultoj koloni, pa je 0 najmanji broj koji mozemo navesti uz TAB. Najveci dozvoljeni broj je 255.

U slucaju da je trenutna pozicija kursora veca od one koju smo mi naveli, ispis ce se nastaviti u prvoj sljedećoj koloni. Evo primjera:

10 PRINT TAB(10); "MIKRORACUNALO"; TAB(5); "ORAO"

Ispis pocinje od 10. kolone s "MIKRORACUNALO", a na kraju tog ispisa kursor se nalazi u 23. koloni, sto je vece od vrijednosti koju smo naveli u slijedecoj TAB funkciji, tako da ce se ispis nastaviti od 23. kolone s "ORAO".

Funkcija TAB koristi se iskljucivo s PRINT naredbom, a upotrebljava se uglavnom za tabuliranje ispisa na ekranu, pa otuda i njezino ime.

Primjer: na ekranu cemo ispisati tri kolone. U prvoj ce biti brojevi od 1 do 20, u drugoj njihovi kvadrati, a u trecoj isti brojevi na treću potenciju:

```
10 CLS
20 PRINT TAB(5); "X"; TAB(15); "X^2"; TAB(25); "X^3"
30 PRINT
40 FOR X=1 TO 20
50 PRINT TAB(4); X; TAB(15); X*X; TAB(25); X*X*X
60 NEXT X
```

SPC funkcija

=====

Funkcija SPC (X) ubacuje X praznih mesta u ispisu, tj. pomice kursor prilikom ispisa za X pozicija u desno, s obzirom na trenutnu poziciju (engl."space"=razmak).

Ako u zagradi navedemo broj veci od 32, ispis ce preci u novi red, pa cemo dobiti jedan red prazan. Najveci broj koji smijemo navesti u zagradi uz funkciju SPC je 255, a najmanji je 0.

Funkciju SPC mozemo korisno upotrijebiti za razdvajanje ispisa na ekranu, npr:

```
10 CLS
20 A$="MIKRORACUNALO"
30 B$="ORAO"
40 PRINT A$;SPC(5);B$
```

Funkcijom SPC sluzimo se iskljucivo unutar naredbe PRINT

POS funkcija

=====

POS (X) daje položaj kursora na ekranu, odnosno daje broj kolone u kojoj se kursor nalazi nakon ispisa. Na primjer:

```
10 PRINT "ORAO";
20 PRINT POS(X)
RUN
ORAO 4
```

jer se kursor nakon ispisa nalazio u 4. koloni.

Vazno je uociti znacaj tocke-zareza iza PRINT naredbe - pozicija kurzora se zadrzava na onom mjestu gdje smo stali u prethodnom ispisu. Probajte ovo:

```
10 PRINT "ORAO"  
20 Y=POS(X)  
30 PRINT Y  
RUN  
ORAO  
0
```

samo zato sto smo izbacili znak ";".

INV
==

Vrlo cesto javlja se potreba da se na ekranu neke rjeci ili recenice ISTAKNU, a to se postize tako da se rijec napisu u inverznom obliku. Normalan nacin ispisivanja za mikroracunalo Orao je bijeli znak na crnoj podlozi. Kad ukljucimo INVERZNI nacin pisanja, ispisuju se crni znakovi na bijeloj podlozi. Treceg nacina nema.

Naredbom

INV 1

uklucujemo inverzni nacin pisanja, dok pomocu

INV 0

inverzni nacin iskljucujemo, odnosno vracamo se u normalan nacin pisanja.

Primjer 1:

```
10 PRINT "MIKRORACUNALO ";  
20 INV1: PRINT "(ORAO)";: INV0  
30 PRINT " PEL VARAZDIN"
```

Primjer 2:

```
10 PRINT "ORAO";  
20 INV1: PRINT " - PEL"
```

Probajte nakon izvrsavanja primjera 2 izlistati program. Listing programa se pojavljuje u inverznom obliku. Iskljuciti ga mozete na dva nacina - pritiskom na **<PF3>** ili ako napisete direktno **INV0 <CR>**. Razlika izmedju upotrebe naredbe INV i tastera **<PF3>** je ta sto se INV moze koristiti u programima i u zeljenim momentima ukljucivati i iskljucivati inverzni nacin ispisa, sto je pomocu tastera **<PF3>** nemoguce.

VDU

====

Naredbom VDU definiramo velicinu aktivnog dijela prikaza teksta na ekranu, tj. postavljamo prozor za tekst.

U normalnim okolnostima, na ekranu se slika prikazuje u 32 reda i 32 kolone. Pomocu naredbe VDU (uz koju se navode dodatni parametri) prikaz teksta na ekranu moze se ograniciti na, na primjer, 15 redova i 10 kolona, ili 20 redova i 23 kolone. Takav dio ekrana tada se naziva PROZOR.

Naredba VDU ima cetiri parametra i izgleda ovako:

VDU LI, DE, GO, DO

LI - prva (Lijeva) kolona	- 0 <= LI < 31
DE - posljednja (Desna) kolona	- 0 < DE <= 31
GO - prvi (Gornji) red	- 0 <= GO < 31
DO - posljednji (Donji) red	- 0 < DO <= 31

Primjer - VDU

Prepisite i izvrsite ovaj program:

```
X 10 CLS
20 A$= "MIKRORACUNALO"
30 FOR X=1 TO 18
40 CUR 6,20
50 PRINT RIGHT$(A$,X)
60 SOUND X*10,30
70 NEXT X
80 VDU 0,31,24,31
90 CLS
```

Kad ste ga izvrsili, izlistajte ga:
LIST <CR>

Prve linije programa su se izgubile, sto se je dogodilo?
Probajte sad obrisati ekran sa CLS:

CLS <CR>

Tekst "MIKRORACUNALO ORAO" i dalje stoji na istom mjestu. O cemu se zapravo radi?

U liniji 80:

80 VDU 0,31,24,31

Ogranicili smo prikaz na ekranu, odnosno "otvorili" prozor koji je sirine 32 kolone (od 0 do 31) i visine 8 redova (od 24. do 31. reda). Sadrzaj ekrana od nultog do 23. reda ostaje sacuvan, ne brise ga CLS, vec se brise samo sadrzaj prozora na dnu ekrana.

Otvorite sada novi prozor:

VDU 2,13,10,19 <CR>

i obrisite ga pomocu CLS ili <CTL>i<L>.

Izlistajte program: LIST <CR>

Tako je otvoren novi prozor ...

Povratak na format ekrana od 32 reda i 32 kolone
ostvarili bi pomocu naredbe

VDU 0,31,0,31

Na taj nacin sadrzaj ekrana ne bi bio izbrisani. Ako nam
sadrzaj ekrana nije vayan, mozemo pisati samo

VDU <CR>

Naredba VDU bez dodatnih parametara prvo postavlja
prozor 0,31,0,31 tj. postavlja prozor za tekst preko cijelog
ekrana, a nakon toga i brije ekran (CLS). Ponekad se
naredbom VDU bez ostalih parametara mozemo sluziti umjesto
naredbe CLS (narocito na pocetku svakog programa).

SCREEN\$ funkcija
=====

Funkcija SCREEN\$ sluzi za "citanje" znaka s ekrana.
(engl."SCREEN"=ekran)

Koristi se u obliku:

CUR A,B:A\$=SCREEN\$

tako da se string varijabli pridruzuje znak koji se na
ekranu nalazi u koloni A i retku B.

Primjer 1 - zelimo procitati znak na ekranu koji se na-
lazi u 20. redu i 10. koloni

10 CUR 10,20
20 A\$=SCREEN\$
30 PRINT A\$

Primjer 2 - zelimo procitati koji se znak nalazi u petom
redu i 17. koloni ekrana

10 VDU
20 CUR 15,5
30 PRINT "ORAO"
40 CUR 17,5
50 B\$=SCREEN\$
60 CUR 16,16: PRINT B\$

Primjer 3:

Najprije cemo ovrisati ekran i negdje na ekranu ispisati
rijec "PEL". Nakon toga cemo na ekranu traziti znak "E"
pocevsi od nultog reda i nulte kolone, dakle od samog pocetka
ekrana. Na svakoj poziciji gdje se ne nalazi znak "E" ispisat
cemo TOCKU, a kad na ekranu pronadjemo "E", cuti cemo zvucni
signal i dobiti podatke na kojoj poziciji na ekranu se znak
"E" nalazi.

```

10 CLS
20 CUR 20,15: PRINT "PEL"
25 T$="E"
30 R=0 :REM OD NULTOG REDA
40 K=0 :REM OD NULTE KOLONE
50 CUR K,R: A$=SCREEN$
60 IF A$=T$ THEN 110
70 CUR K,R: PRINT "."
80 K=K+1: IF K<32 THEN 50
90 R=R+1: IF R>31 THEN 40
100 PRINT "NA EKRANU NEMA ZNAKA ";T$:END
110 PRINT: SOUND 50,50
120 PRINT "ZNAK <";T$;"> NALAZI SE U REDU";R;
    " I KOLONI";K
130 END

```

Da malo prokomentiramo ovaj program. U njemu smo definirali dvije varijable, R i K. R nam označava retke, a K kolone. Za svaku pojedinacnu vrijednost varijable R, K se promjeni 32 puta, poprimajući vrijednosti od 0 do 31 (u svakom redu imamo 32 kolone). U linijama 50 i 60 ispitujemo znak koji smo upravo procitali s ekrana. Ako je to znak kojeg trazimo, program nastavlja izvodjenje od linije 110, a ako nije, nastavlja dalje od linije 70 gdje se ispisuje točka na onom mjestu ekrana za koje smo malo prije utvrdili da ne sadrži traženi znak. Probajte promijeniti vrijednost za T\$ u liniji 25 (npr. "S"). Pogledajte što će se dogoditi. U liniji 90 povećala se vrijednost za R do 31 – znači da smo dosli do kraja ekrana. Ispisuje se poruka iz linije 100, gdje se na kraju nalazi obavezna END naredba, znak računalu da mora stati s izvođenjem programa.

2.13. NUMERIČKE FUNKCIJE

Numerička funkcija je definirana u obliku:

$$y=f(x)$$

odnosno, varijabli "y" pridružujemo funkciju vrijednost "f" od argumenta "x". Svakoj numeričkoj funkciji slijedi jedan podatak naveden u zagradi (to je ARGUMENT funkcije), a funkcija daje određeni rezultat (to je REZULTAT funkcije). Sve funkcije koje imaju numerički argument i daju numerički rezultat zovu se NUMERIČKE FUNKCIJE.

Argument svake funkcije mora stajati unutar zagrade, inace nas računalo neće razumjeti.

U ovom dijelu upoznat ćemo sve numeričke funkcije s kojima raspolaze BASIC mikroracunala DRAO. To su funkcije:

ABS, INT, RND, SGN, SQR, LOG, EXP, SIN, COS, TAN, ATN

ABS funkcija

=====

Funkcija ABS (X) izracunava APSOLUTNU VRIJEDNOST argumenta (X). Jednostavnije receno, funkcija ABS (X) pretvara predznak broja X u PLUS, bio on negativan ili pozitivan.

```
ABS (-4)=4  
ABS (4) =4
```

```
PRINT ABS(-2.73) <CR>
```

2.73

```
PRINT ABS(2.73) <CR>
```

2.73

Ako je argument funkcije ABS negativan broj, dobit ćemo isti taj broj, ali pozitivan. Na pozitivne brojeve funkcija ABS nema nikakav učinak.

Funkciju ABS(X) najčešće koristimo pri radu s kvadratnim korijenima ili logaritmima. Ponekad se javi potreba da upotrijebimo neku funkciju koja ne dopušta negativne argumente. Pomocu funkcije ABS(X) tada sve argumente možemo učiniti pozitivnima, tj. prihvatljivim za tu funkciju.

INT funkcija

=====

Funkcijom INT (X) dobivamo PRVU MANJU cijelobrojnu vrijednost decimalnog argumenta (X).

Jednostavnije bi to rekli ovako: funkcija INT (X) služi nam za dobijanje CIJELOG BROJA i to tako da se broj u zagradi zaokrugi na PRVI MANJI broj.

```
PRINT INT (7.8) <CR>
```

7

```
PRINT INT (-4.2) <CR>
```

-5

Rezultat funkcije INT (X) za POZITIVAN BROJ bit će cijelobrojni dio broja X, odnosno ono što se nalazi s lijeve strane decimalne točke.

S NEGATIVNIM BROJEVIMA je malo "drugacije". Rezultat INT(X) za negativan decimalni broj X bit će prvi manji cijeli broj ispod broja X, na primjer:

```
PRINT INT(-2.001) <CR>  
-3
```

```
PRINT INT(-7.999) <CR>
```

* -8

Ako zelimo pravilno izvoditi zaokruzivanje brojeva uz INT (X) funkciju, broju X moramo prvo pribrojiti vrijednost 0.5 (sto slijedi iz pravila o zaokruzivanju brojeva). Ovo vrijedi i za pozitivne i za negativne brojeve, argumente funkcije INT (X).

Funkciju INT (X) mozemo upotrijebiti i onda kad zelimo provjeriti da li je neki broj cijeli broj. To se postize na ovaj nacin:

```
10 INPUT A  
20 IF A=INT (A) THEN PRINT A;" JE CIJELI BROJ":END  
30 PRINT A;" NIJE CIJELI BROJ"
```

Funkciju INT (X) mozemo korisno upotrijebiti i zato da ispitamo da li je neki broj dijeljiv s drugim brojem. Pogledajte slijedeci koji se zove DJELJITELJI BROJA:

```
10 REM djeljitelji broja  
20 INPUT N  
30 FOR K=2 TO N/2  
40 IF N/K=INT(N/K) THEN PRINT K  
50 NEXT K
```

RND funkcija
=====

Ova funkcija omogucuje mikroracunalu da "zamisli" neki SLUCAJNI BROJ, odnosno generira slucajni broj u intervalu izmedju 0 i 1. Ukoliko se, kao argument RND (X) funkcije, u zagradi nalazi broj 0, racunalo ce generirati uvijek isti broj, a ako je u zagradi bilo koji veci broj od 0, racunalo ce svaki puta "zamisliti" drugi broj. Unesite ovaj primjer:

```
10 FOR X=1 TO 15  
20 PRINT RND (7)  
30 NEXT X
```

Na ekranu vidimo ispis od 15 slucajnih brojeva. Svaki od njih vec je od 0, a manji od 1. Slucajni broj ovako generiran moze biti 0, ali ne moze biti 1.

Ova se funkcija moze vrlo korisno primjeniti kod testiranja matematickih i statistickih programa, kod igara je gotovo nezaobilazna ...

Rijetko se javlja potreba za slucajnim brojem izmedju 0 i 1. Potrebno je generirati slucajni brj koji je pogodan za dalju primjenu - crtanje, racunanje ... Pomocu slijedeceg primjera ispisat cemo 20 zvjezdica na slucajno odabranim mjestima na ekranu. Da bi to mogli napraviti, bit ce potrebno generirati brojeve izmedju 0 i 31 i zaokruziti ih na cijele, radi pozicioniranja kursora na ekranu:

```
10 CLS:FOR X=1 TO 20  
20 A=RND(7)*32  
30 B=RND(7)*32  
40 K= INT(A)  
50 R= INT(B)  
60 CUR K,R: PRINT "*"  
70 NEXT X
```

Brojevi generirani u linijama 20 i 30 ce uvijek biti manji od 32, a INT(X) ce ih zaokruziti na prvu manju vrijednost, tako da niti jedna zvjezdica nece "ispasti" iz ekrana.

Ako trebamo slucajne brojeve u rasponu od 0 do nekog broja, u ovom slucaju do 31, potrebno je slucajni broj iz intervala 0 do 1 pomnoziti s brojem koji je za jedan VECI od GORNJE granice intervala, a veci je zbroj funkcije INT (X) koja ce zaokruziti taj broj na prvu manju cijelobrojnu vrijednost. Ako nam je potrebno odrediti slucajni broj u nekom drugom intervalu (ne od nule), npr. od 5 do 8, tada cemo to izvesti ovako:

```
INT ((8-5+1)*RND(7)+5)
```

Slijedecim primjerom ispisat cemo deset brojeva iz intervala od 5 do 8:

```
10 FOR X=1 TO 10
20 N=INT((8-5+1)*RND(5)+5)
30 PRINT N
40 NEXT N
```

Jednostavnije bi liniju 20 bilo napisati ovako:

```
20 N=INT(4*RND(5)+5)
```

Ova funkcija nam sluzi da ispitamo kakav PREDZNAK ima neki broj. Ako je argument ove funkcije (broj u zagradi) pozitivan broj, dobit cemo rezultat 1. Ako je argument negativan, rezultat funkcije bit ce -1. Za nulu je rezultat nula. Dakle:

```
Y=SGN(X)
```

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

Primjer:

```
10 A=2.4
20 B=0
30 C= -4.73
40 PRINT SGN(A), SGN(B), SGN(C)
```

Isto bi mogli napisati i u samo jednoj liniji:

```
PRINT SGN(2.4),SGN(0),SGN(-4.73)
```

Funkciju SGN koristimo u ispitivanjima predznaka brojeva. Na primjer, nije moguce racunati drugi korijen iz negativnih brojeva. Da bi sprijecili pojavu greske, a time i prekid izvodjenja programa, mozemo upotrijebiti oву funkciju da ispitamo da li je broj negativan.

SQR funkcija

Jedna od cestih racunskih operacija je racunanje drugog (kvadratnog) korijena. Zato nam sluzi funkcija SQR (X).

Funkcija

$$Y=SQR(X)$$

dodjeljuje varijabli Y kvadratni korijen argumenta X.
Na primjer:

PRINT SQR(9) <CR>

3

Y=SQR(16):PRINT Y <CR>

4

Argument funkcije SQR(X) NE SMIJE biti negativan broj.

LOG funkcija

Funkcija LOG (X) daje nam prirodni logaritam argumenta X (prirodni logaritam je logaritam broja s bazom $e=2.71828$). Argument ove funkcije mora biti pozitivan broj veci od nule. Za dobivanje logaritma po bazi b od nekog broja, koristimo jednadzbu

$$\log_b x = \log x / \log b$$

Primjer - odredi dekadski logaritam od broja 2:

10 B=10

20 Y=(LOG(2))/(LOG(10))

30 PRINT Y

EXP funkcija

Iz prirodnog logaritma mozemo izracunati antilogaritam pomocu funkcije EXP (X).

Funkcija

$$Y=EXP (X)$$

dodjeljuje varijabli Y potenciju broja "e" ($e=2.71828$). Vrijednost argumenta definira potenciju broja "e".

Primjer:

operacija Y=2.71828^X

isto je sto i Y=EXP(X)

Vrijednost broja "e" za mikroracunalo Orao izgleda ovako:

upisite PRINT EXP(1) <CR>

Od TRIGONOMETRIJSKIH FUNKCIJA Basic mikroracunalna Orao posjeduje funkcije SIN, COS, TAN i ATN.

SIN funkcija =====

Funkcija $Y=\text{SIN}(X)$ dodjeljuje varijabli Y sinus od argumenta u zagradi, izrazen u radijanima. Ako zelimo radijane pretvoriti u stupnjeve, podijelit ćemo vrijednost u radijanima s PI i pomnoziti sa 180:

$$\text{kut u radijanima}/\text{PI}*180 = \text{KUT U STUPNJEVIMA}$$

Ako pak imamo vrijednost izrazenu u stupnjevima, preračunat ćemo ju u radijane tako da vrijednost u stupnjevima podijelimo sa 180 i pomnozimo s PI:

$$\text{kut u stupnjevima}/180*\text{PI} = \text{KUT U RADIJANIMA}$$

COS funkcija =====

Funkcija $Y=\text{COS}(X)$ dodjeljuje varijabli Y kosinus od argumenta X izrazen u radijanima.

TAN funkcija =====

Funkcija $Y=\text{TAN}(X)$ dodjeljuje varijabli Y vrijednost tangens od argumenta X izrazen u radijanima.

ATN funkcija =====

Funkcija $Y=\text{ATN}(X)$ dodjeljuje varijabli Y vrijednost arkus tangens.

Na kraju navedimo da BASIC mikroracunalna Orao ne posjeduje funkcije kao:

$$y=\text{arcsin}(x) \text{ i } y=\text{arccos}(x)$$

pa stoga moramo koristiti sljedeće jednadzbe:

$$\text{a)} \quad \text{arcsin } (x)= \text{atn} \quad \text{za } |x|<1$$

$$\text{b)} \quad \text{arccos } (x)= \text{atn} \quad \text{za } 0 < x <=1$$

2.14. FUNKCIJE DEFINIRANE OD STRANE KORISNIKA

U programu se neki matematicki izraz moze ponavljati na vise mesta. U tom slucaju pogodno je definirati funkciju kojoj se dodjeljuje vrijednost tog izraza.

Ako se nam u programu, npr, cesto javlja potreba za izracunavanjem kvadrata brojeva, nije potrebno svaki puta posebno vrsiti kvadriranje. Definirat cemo si funkciju A koja izracunava kvadrat broja B i to na slijedeci nacin:

10 DEF FN A(B)=B*B

i svaki puta kad nam zatreba kvadrat nekog broja, pozvat cemo funkciju A. Uzmimo da nam treba izracunati i ispisati koliko je 7 na kvadrat:

20 PRINT FN A(7)

Definirati mozemo samo numericke funkcije koje imaju SAMO JEDAN argument (to je B u zagradi u liniji 10).

Rezultat korisnicki definirane funkcije je takodjer broj. Svakoj funkciji moramo dati IME, kako bi ju kasnije mogli pozivati (to je A u liniji 10). Ime sadrzi samo jedan znak (veliko slovo abecede).

Kad pozivamo ovaku definiranu funkciju, kao argument unutar zagrade mozemo navesti:

- BROJ
npr: 30 PRINT FN A(B)
- VARIJABLJU
npr: 40 PRINT FN A(C)
- MATEMATICKI IZRAZ
npr: 50 Z=FN(B+3):PRINT Z

Primjer:

Definirat cemo funkciju koja izracunava povrsinu kruga:

```
10 PI=3.141593
20 DEF FNP(R)=R^2*PI
30 INPUT W
40 PRINT FNP(W)
RUN
?2
12.566372
```

Pogledajte liniju 20:
20 DEF FNP(R)=R^2*PI

Kod definiranja funkcije, iza imena funkcije navodimo ime jedne varijable (R). TO ISTO IME moramo upotrijebiti s desne strane znaka jednakosti, dakle prilikom samog definiranja funkcije. Kad cemo funkciju pozvati (linija 40), navest cemo u zagradi neki drugi podatak (broj, varijablu ili

matematicki izraz). Ovdje je to varijabla W ciju smo vrijednost prethodno unijeli preko tastature. Kod poziva funkcije treba paziti jer trebanavesti točno ime funkcije.

I još nesto - prije nego cemo pozivati odredjenu funkciju, ona mora biti već definirana. Zbog toga se u praksi naredbe DEF FN nalaze na samom početku programa.

2.15. PISANJE PROGRAMA I PRINCIPI PROGRAMIRANJA

Iako još niste naučili sve BASIC naredbe mikroracunala DRAO, možda će netko od vas pozeljeti da već sada napise svoj prvi program sluzeci se naredbama koje smo dosad upoznali. Ali prije nego što se upustite u taj posao, bilo bi vrlo korisno da pazljivo pročitate i proučite ovo poglavlje u kojem ćete saznati kako se pisu programi - bilo da se radi o programima koje cemo pisati u BASIC-u ili u nekom drugom programskom jeziku.

Svako računalo služi nam kao pomocno sredstvo pri obradi raznih podataka. Sposobnosti i mogućnosti računala su vrlo velike i raznovrsne, a mogu se "aktivirati" odgovarajućim programom. Prema tome, mogli bi reci da program usmjerava univerzalne sposobnosti i mogućnosti računala na rješenja određenog problema.

Svaki program može se predstaviti pomoći tri osnovne programske strukture, a to su:

- sekvenca
- selekcija
- iteracija

SEKVENCA (linijska struktura) je niz naredbi koje se izvršavaju jedna za drugom.

SELEKCIJA (razgranata struktura) je mjesto grananja u programu. U BASIC-u se to ostvaruje pomoći naredbi IF-THEN.

ITERACIJA (petlja) je struktura u kojoj se jedna ili više naredbi izvršavaju više puta za redom, sve dok nije zadovoljen uvjet za izlazak iz petlje. Petlju formiramo pomoći naredbi FOR-NEXT, ili IF-THEN uz pomoći GOTO.

Proces pisanja programa (programiranja) obuhvaca više faz, a zapocinje DEFINIRANJEM I ANALIZOM PROBLEMA (zadatka) kojeg namjeravamo rjesiti pomoći računala. Analizom zadatka problem postaje konkretniji i detaljnije razradjen. Zadatak se razbija na manje cjeline (MODULE), a svaka od njih se obradjuje posebno.

Nakon toga potrebno je napraviti PLAN TIJEKA PODATAKA i utvrditi potrebnu ORGANIZACIJU PODATAKA u programu, nakon čega možemo utvrditi TISK ODVIJANJA PROGRAMA koji se može vrlo pogodno predstaviti graficki, u obliku DIJAGRAMA TOKA (podataka, odnosno programa).

Pisanje programa u jeziku koje racunalo razumije najcesce se u literaturi naziva PROGRAMIRANJE, sto nije pravilno jer se pojam "programiranje" odnosi iskljucivo na apstrakciju problema koji zelimo rjesiti racunalom. Problem svodimo na kreiranje ALGORITMA tj. postupka ili kljуча koji ce nas najkracim putem dovesti do zeljenog rjesenja. Stoga pisanje programa u BASIC-u ili nekom drugom programskom jeziku nije "programiranje", vec KODIRANJE programa koji je izrazen opisno dijagramom toka.

Znaci, poslije faze izrade dijagrama toka, slijedi kodiranje programa u odgovarajuci programske jezik. Skup

ovako dobivenih instrukcija naziva se IZVORNI PROGRAM. Testiranjem ovako napisanog programa utvrdjuje se njegova ispravnost. Sve greske koje su se pojavile prilikom testiranja potrebno je otkloniti da dobijemo 100% ispravan program.

Na kraju je potrebno izraditi DOKUMENTACIJU za konacnu verziju programa, a program SNIMITI (na kazetu, disketu...) radi kasnije upotrebe.

Da biste sve ovo lakse shvatili, proc i cemo zajedno cijeli ovaj proces dva puta, u dva primjera. Napisat cemo zajedno dva programa.

PRIMJER 1:

Fred nema je zadatak da napisemo program koji ce na ekranu ispisati 100 brojeva, odnosno brojeve od 0 do 99.

Ovaj problem trebamo tako pripremiti da ga Orao moze obraditi nizom svojih osnovnih naredbi.

Dakle, zadatak nam je poznat i vrlo je jednostavan. U glavi mozda vec imate gotovo rjesenje - ipak, bolje ce biti da ga korak po korak, sasvim detaljno, prikazemo na papiru. Kako? Postoje dva nacina:

1. opisno
2. dijagramom toka

Prikazimo ovaj problem OPISNO:

KORAK 1 - postavi vrijednost varijable N na nulu

KORAK 2 - ispisi vrijednost varijable N

KORAK 3 - uvecaj vrijednost varijable N za 1

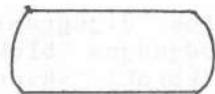
KORAK 4 - ispitaj da li je vrijednost varijable N veca od 99:
- ako NIJE veca idi na KORAK 2
- ako JE VEECA idi na KORAK 5

KORAK 5 - prekid izvodjenja programa (KRAJ)

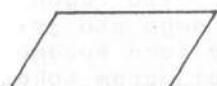
Ovakav prikaz programa naziva se SKICA programa.

Skicu programa mozemo prikazati i GRAFICKI, a to je najcesce i najbolji put da se prikaze i izradi točna skica. Graficki izradjenu skicu programa nazivamo DIJAGRAM TOKA PROGRAMA ili BLOK DIJAGRAM.

Prilikom izrade ovih dijagrama sluzimo se standardnim grafickim simbolima koji odgovaraju pojedinim operacijama u nasem programu. To su slijedeci simboli:



- definiranje pocetka i kraja problema
(dolazi na pocetku i na kraju)



- ulazno-izlazne operacije



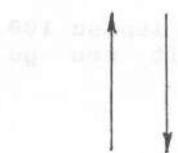
- definiranje procesa u problemu



- simbol odluke

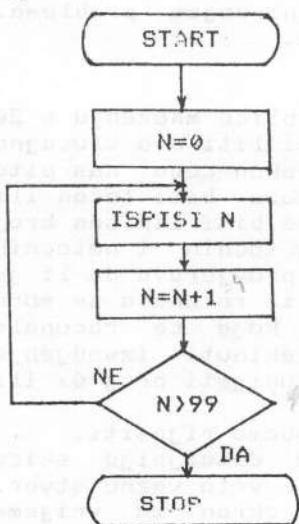


- veznik



- oznacavanje toka programa
strelicama povezujemo ostale graficke simbole u dijagramu toka, a smjer strelice označava tok pojedinih informacija iz blok u blok

Nacrtajmo sada dijagram toka za problem ispisa 100 brojeva:



Prilikom pisanja programa preporučljivo je da se najprije izradi opisna skica programa, a nakon nje i detaljni graficki dijagram toku (blok dijagram).

Sada, kad je gotova skica programa i blok dijagram, mozemo poceti s kodiranjem, odnosno prevodjenjem blok dijagrama u BASIC program. Svaki elemenat (korak) skice programa (i bloka dijagrama) potrebno je prevesti u jednu ili vise BASIC naredbi. Problem ispisa 100 brojeva je vrlo lagan-izradjena skica programa ne moze biti drugacija nego sto je, a ona je toliko detaljna da svaki korak dijagrama toku mozemo prevesti u jednu naredbu. Evo kako izgleda nas dijagram toku kodiran u BASIC-u:

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

Program kodiran u BASIC-u sastoji se od programske linija koje sadrze jednu ili vise naredbi. Duljina linije je najvise 72 znaka. Svaka programska linija obavezno mora biti numerirana brojem izmedju 0 i 63999, s time da svaka sljedeca linija mora imati broj veci od prethodne.

Ovime smo rjesili nas prvi zadatak, problem ispisa 100 brojeva. Drugi je nesto tezi i slozeniji. Rjesit ceo ga takodjer zajedno i to detaljno, korak po korak.

PRIMJER 2:

Pred nama je ovaj puta zadatak da napisemo program koji ce nam postavljati zadatke iz tablice mnozenja i to s 20 brojeva, znaci do 20×20 .

Posao cemo zapocet analizom i definiranjem problema. Zapisat cemo sve sto znamo o samom zadatku.

OPIS ZADATKA:

Racunalo nam postavlja pitanje iz tablice mnozenja s 20 brojeva (do 20×20). Dakle, racunalo ce zamisliti dva slucajna broja A i B u intervalu od 1 do 20, nakon cega nas pita koliko je $A \times B$. Mi unosimo odgovor koji moze biti tocan ili netocan. U svakom trenutku na ekranu treba biti ispisani broj ukupno postavljenih pitanja, kao i broj tocnih i netocnih odgovora. Kada unesemo odgovor, racunalo provjerava da li je on tocan ili ne. Za svaki tocan odgovor iz racunala se mora cuti kratki zvucni signal. Broj pitanja koje ce racunalo postaviti nije ogranicen. Kad zelimo prekinuti izvodjenje programa, potrebno je, umjesto odgovora, unijeti broj 0, ili jednostavno pritisnuti <CR>.

To je, dakle, glavni zadatak koji trebamo rjesiti.

Sada bi vec mogli preci na izradu detaljnije skice programa, međutim smo zaboravili na jednu vrlo vaznu stvar. Nije svejedno kako ce nam izgledati ekran za vrijeme

izvodjenja. Trebamo, znaci, isplanirati kako cemo urediti IZGLED EKRANA. To je takodjer vrlo vazan dio svakog programa. Cim nam ekran ljepse izgleda, imat cemo osjecaj da je nas program bolji, sto je u neku ruku i točno. Kad uredjujemo izgled ekrana, to znaci da odredjujemo kako će se odvijanje programa moci pratiti na ekranu. Nas je cilj da taj prikaz uredimo sve ljepse, da na ekranu uvijek budu svi potrebni podaci, naslovi da idu u sredinu reda, koristit cemo inverzni nacin pisanja, otvarati prozore za tekst, izbjegavati nepotrebno gomilanje raznih podataka, pitanja i odgovora na ekranu, paziti da nam naslov ne "ispadne" iz ekrana zato sto smo napunili cijeli ekran, ... Kod uredjivanja ekrana valja pripaziti na bezbroj sitnica, a njih cete najbolje upoznati kroz praksu i samostalno programiranje.

Mi cemo si ekran urediti ovako:

Na vrhu ekrana inverzno cemo ispisati naslov programa. Ispod naslova ostavit cemo par redova praznih i u sredini slijedeceg ispisivati broj ukupno postavljenih pitanja. Ispod toga jedan red prazan, a onda u lijevom kutu ide brojac tocnih, a u desnom kutu brojac netocnih odgovora. Kad cemo odgovarati na pitanja, pazit cemo da se na ekranu ne gomilaju pitanja jedno ispod drugog, vec da na ekranu imamo samo jedno pitanje uz trenutni rezultat odgovaranja na vrhu ekrana.

Pocnimo s izradom opisne skice naseg problema:

Rekli smo vec da se svaki problem moze rasclaniti na nekoliko manjih i zaokruzenih cjelina za koje je karakteristично da imaju odredjeni stupanj samostalnosti. Takve logische cjeline nazivaju se MODULI.

Nasu tablicu mnozenja razbit ćemo na nekoliko cjelina, a nakon toga svaku cjelinu obraditi posebno.

Dijelovi problema:

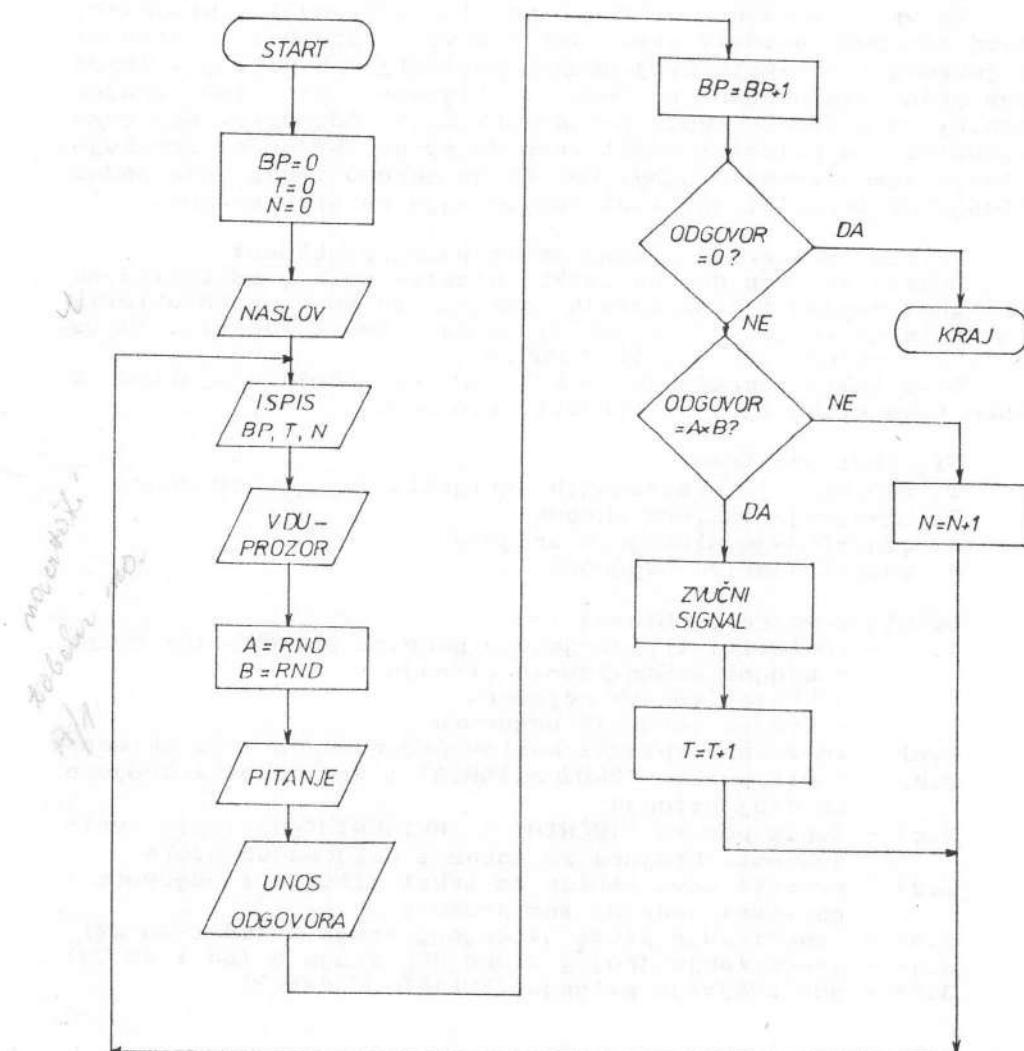
1. definiranje programskih varijabli (cvije brojaca)
2. uredjenje izgleda ekrana
3. generiranje slucajnih brojeva
4. unos i analiza odgovora

Detaljna skica problema:

1. - postaviti tri brojaca u pocetni položaj (na nulu)
 - brojac postavljenih pitanja
 - brojac tocnih odgovora
 - brojac netocnih odgovora
- 2.a) - inverzno ispisati naslov programa na vrhu ekrana
- 2.b) - ispis poruke "BROJ PITANJA" i vrijednosti brojaca za broj pitanja
- 2.c) - ispis poruke "TOCNIH" i "NETOCNIH" uz ispis vrijednosti brojaca za tocne i netocne odgovore
- 2.d) - otvoriti novi prozor za tekst pitanja i odgovora i obrisati sadrzaj tog prozora
- 3.a) - generiranje prvog slucajnog broja A (od 1 do 20)
- 3.b) - generiranje drugog slucajnog broja B (od 1 do 20)
- 3.c) - postavljanje pitanja "KOLIKO JE A*B ?"

- 4.a) - unos odgovora
 4.b) - povecati za jedan brojac postavljenih pitanja
 4.c) - provjeriti da li je odgovor NULA; ako je nula
 idu na kraj programa
 4.d) - analiza odgovora:
 - ako je odgovor TOCAN napraviti slijedeće:
 - generirati zvucni signal
 - povecati za jedan brojac točnih odgovora
 - nastaviti s izvodnjem od 2.b)
 - ako je odgovor NETOCAN napraviti slijedeće:
 - povecati za jedan brojac netočnih odgovora
 - nastaviti s izvodnjem programa od 2.b)
 5. - KRAJ programa

Prikazimo graficki ovu skicu pomocu BLOKA DIJAGRAMA:



Na kraju cemo dijagram toku kodirati u BASIC. Evo kako izgleda konacni program:

```
5 REM =====
6 REM TABLICA MNOZENJA
7 REM =====
10 BP=0:T=0:N=0
15 VDU
20 INV1
25 PRINT " *** TABLICA MNOZENJA ***"
30 INV0
35 CUR 7,3: PRINT "BROJ PITANJA=";BP
40 CUR 0,5: PRINT "TOCNIH=";T
45 CUR 17,5: PRINT "NETOCNIH=";N
50 VDU 0,31,9,31:CLS
55 A=INT (RND(7)*20)+1
60 B=INT (RND(7)*20)+1
65 PRINT "KOLIKO JE ";A;"*";B;
70 INPUT R
75 BP=BP+1
80 IF R=0 THEN VDU: END
85 IF R=A*B THEN SOUND 50,50: T=T+1: GOTO 35
90 N=N+1: GOTO 35
```

Prokomentirajmo ukratko zajedno ovaj program:

U liniji 10 definirali smo tri brojaca, brojac pitanja (BP), brojac tocnih (T) i netocnih (N) odgovora.

Naredbe u linijama 15 do 45 vjerojatno su vam jasne. Da pogledamo liniju 50. U njoj otvaramo prozor za tekst koji zauzima mjesto na ekranu od 0. do 31. kolone i od 9. do 31. reda ekrana. Naredba CLS u liniji 50 brise samo sadrzaj tog prozora, ali ne brise sadrzaj ekrana od 0. do 8. reda gdje smo smjestili naslov i ostale ispise. Zbog ove naredbe CLS cursor se automatski pomice na vrh PROZORA (ne ekrana !) gdje ce se pojaviti pitanje. Obratite paznju na nacin na koji smo generirali slucajne brojeve. Njihova vrijednost ne smije biti manja od 1, niti veca od 20. Nakon sto smo unesli i ispitali tocnost odgovora, povecali sadrzae odredjenih brojaca (osim ako nije kraj), skacemo na liniju 35. Linije 35, 40 i 45 ispisuju sadrzaje brojaca i to u djelu ekrana koji se NE NALAZI u prije otvorenom prozoru. To je moguce samo ako si pozicioniramo cursor u taj dio ekrana (naredbom CUR).

Ponovno dolazimo u liniju 50 koja ce ponovo otvoriti i obrisati ISTI prozor a to znaci da se prethodno pitanje BRISE s ekrana. Time smo postigli vecu preglednost na ekranu - u svakom trenutku na ekranu se nalazi samo jedno pitanje.

Kad cete pisati vlastite programe, najbolje je da skicu programa prevedete u BASIC program prvo NA PAPIRU, a tek onda da unesete program u racunalo. Na taj nacin cete sigurno izbjegci mnoge greske koje najcesce nastaju prilikom direktnog unosenja programa u racunalo.

Program "tablica mnozenja" nema u sebi niti jednu REM naredbu, ni jednog komentara. Mogli bi dodati liniju:

52 REM GENERIRANJE SLUCAJNIH BROJEVA

i slicne komentare.

DOKUMENTACIJA PROGRAMA sastoji se od svega sto ste zabiljezili na papir za vrijeme izrade programa. Ako imate priliku, napravite listing programa na stampacu. Listingu je potrebno dodati sto vise komentara i opisa. Vec sutra cete pisati nove programe i vrlo brzo zaboraviti na detalje programa kojeg ste danas zavrsili. A u njemu cete, mozda, trebati kasnije nesto promijeniti, modifcirati. Bez uredne dokumentacije to ce vam biti vrlo tesko. Najcesce je lakse poceti pisati program iznova, nego vrsiti izmjene u programu za koji ne postoji uredna dokumentacija.

Prije nego sto zakljucimo izradu dokumentacije, program treba istestirati, tj. provjeriti njegovu ispravnost rada. Potrebno je ukloniti eventualne greske iz programa, a njih ce u svakom programu sigurno biti. Tek kad je program potpuno ispravan i ispitani, on je spremjan za upotrebu.

2-16- ČUVANJE PROGRAMA - RAD S KAZETOFONOM

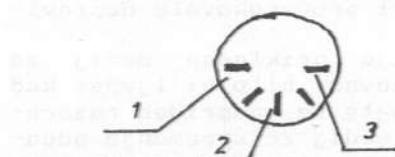
U prethodnom poglavlju napisali smo jedan kratak program, tablicu mnozenja. Racunalo nam je postavilo dvadesetak pitanja, nakon cega nam je bilo dosta matematike. Sto da uradimo s programom koji se nalazi u memoriji racunala? Ako racunalo ugasimo, program ce se izbrisati iz memorije i kad bi ponovo htjel vjezbatи tablicu mnozenja, program bi trebali opet prepisivati u racunalo. Zamislite samo da se radi o nekom duzem programu! Kad bi svaki puta trebali iznova unositi program u racunalo tako da ga prepisujemo s papira, od racunala ne bi imali velike koristi. Kod prepisivanja bi izgubili mnogo vremena i jos vise zivaca. Upravo zato racunalo ima mogucnost da program koji se nalazi u njegovoj memoriji SNIMI pomocu obicnog kucnog kazetofona na najobicniju muzicku kazetu. Kad program snimimo na kazetu, racunalo mozemo ugasiti, a sutradan uzeti kazetu sa snimljenim programom, ukljuciti racunalo i "ucitati" program s kazete u racunalo za nekoliko sekundi.

POVEZIVANJE MIKRORACUNALA ORAO S KAZETOFONOM

Mikroracunalo Orao povezuje se s kazetofonom pomocu specijalnog kabla. To i nije bas tako specijalan kabel. Moze ga izraditi svatko tko ima lemilicu, a zica i konektori mogu se kupiti u svakoj trgovini s tehnickom robom.

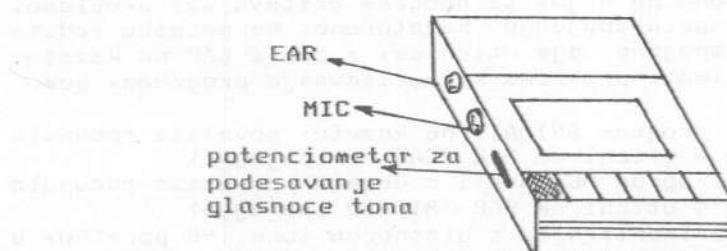
Odgovarajući kraj kabla spoji se na priključak za kazetofon koji se nalazi na stranjoj strani racunala. Drugi kraj kabla usteča se u kazetofon.

Priključak za kazetofon na mikroracunalu DRAO izgleda ovako:



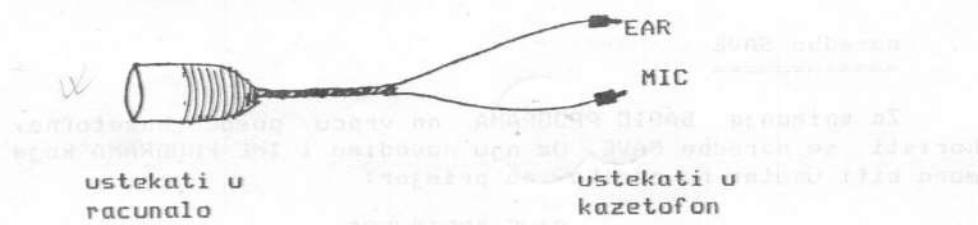
- 1 - na ulaz kazetofona (MIC na kazetofonu)
- 2 - masa
- 3 - na izlaz kazetofona (EAR ili PHONO na kazetofonu)

Na kazetofonu to najčešće izgleda ovako:



Da nebi slučajno pogrijesili, pogledajte sto pise ispod svake uticnice na vašem kazetofonu.

Kabel kojim ćemo povezati racunalo i kazetofon ovog tipa, izgledat će ovako:



Nije obavezno da imate kazetofon ovog tipa (ako su ovakvi mali MONO kazetofoni najpouzdaniji i najjeftiniji). Ako imate odgovarajuci kabel, moguce je SVAKI kazetofon povezati s racunalom.

Kad smo povezali racunalo i kazetofon, potrebno je na kazetofonu podesiti glasnoci i boju tona. Sto se tice glasnoce tona, za pocetak namjestite potenciometar negdje oko sredine ili 2/3 maksimuma, a potenciometrom za boju tona (ako postoji) pojedajte visoke tonove do kraja. Potrebna nam je jos samo kazeta, najobicnija vrpca namijenjena za snimanje muzike koja se moze kupiti u svakoj trgovini. Najbolje ce biti da uzmete NOVU vrpcu. Tako cete biti sigurni da na njoj ne postoje nikakva ostecenja, sto je od izuzetne vaznosti. Kad bi vrpcu bila makar malo ostecena, dio programa koji bi bio snimljen na tom dijelu vrpcu nebi se kasnije mogao ucitati, ili bi se ucitao pogresno, a to bi prouzrokovalo nepravilan rad programa.

Mnogi smatraju da kazetofon nije prikladan medij za pohranjivanje i cuvanje programa. Naravno, bilo bi ljepse kad bi imali disketu jedinicu. No, nemajte se unaprijed razocarati. Kazetofon je izuzetno pouzdan medij za spremanje podataka. Pouzdan, praktican i jeftin, a njime se zaista lako upravlja.

Mozda vam prvi pokusaji snimanja i ucitavanja programa nece uspijevati. Nemajte se odmah predati. Potrebno je samo vise eksperimentirati s DVIJE stvari na kazetofonu:

Prva je glasnoce tona. Probajte snimljeni program ucitavati jednom glasnije, jednom tise, sve dok ne pronadjete nivo glasnoce tona na kojem se program ucitava bez problema. Druga stvar je nacin spajanja kazetofona. Na pocetku radite tako da su vam spojene obje uticnice, i MIC i EAR na kazetofonu. Ako cete imati problema kod ucitavanja programa, pokušajte slijedeće:

- kad cete program SNIMATI na kazetu, povezite racunalo i kazetofon samo s uticnicom MIC (EAR ne spajajte)
- kad cete program UCITAVATI s kazete, povezite racunalo i kazetofon samo s uticnicom EAR (MIC ne spajajte)

Usput eksperimentirajte s glasnocom tona. Na pocetku, u prvih 10-15 minuta, mozda cete se malo namuciti s nekoliko neuspjesnih pokusaja snimanja i ucitavanja programa. Ali, kad uspijete snimljeni program ucitati u racunalo, vase muke su zavrseene. ZAPAMTITE kako treba spojiti racunalo i kazetofon kod snimanja, odnosno kod ucitavanja. Zapamtite OBVEZNO i to u kojem je položaju bio potenciometar za glasnoci tona.

naredba SAVE : POSPREMITI
=====

Za snimanje BASIC PROGRAMA na vrpcu pomocu kazetofona, koristi se naredba SAVE. Uz nju navodimo i IME PROGRAMA koje mora biti unutar navodnika, na primjer:

SAVE "PRIMJER"

Ime programa odabiremo sami, potpuno slobodno, po svojoj volji. Moramo paziti jedino na to da ime programa ne bude duze od DESET znakova. Svakom programu dajemo drukcije ime zato sto ce se s vremenom na istoj vrpci naci nekoliko programa.

Ako se prethodni primjer (tablica mnozenja) jos uvijek nalazi u memoriji racunala, mozemo ga snimiti na kazetu. Pripremite kazetofon i napisite:

SAVE "T.MNOZENJA"

Prije nego sto pritisnete <CR>, ukljucite snimanje na kazetofonu. Ako na kazetofonu imate i brojac okretaja, postavite ga na nulu. Sada pritisnite <CR>. Iz racunala cete zacuti zvuk koji označava da se program snima (snima se u obliku tog zvuka). Ako je program koji snimate duzi od 255 bajtova, cuti cete vise odvojenih BLOKOVA zvuka. Svaki BASIC program snima se u blokovima, tj. u komadima duzine 256 bajtova, a poslije svakog od njih postoji kratka pauza (GAP). Tek kad se na ekranu pojavi cursor, znaci da je snimanje završeno. Ovako snimljen program mozemo ucitati u racunalo na slijedeci nacin: (najprije napisite NEW)

naredba LOAD = PUNI

=====

Pravilno spojite racunalo i kazetofon, a vrpcu premotajte na pocetak programa. Napisite:

LOAD "T.MNOZENJA"

i pritisnite <CR>. Racunalo ce ispisati:

SEARCHING "T.MNOZENJA"

sto znaci da racunalo "trazi" na vrpci program pod imenom "T.MNOZENJA".

Kad je racunalo pronaslo traženi program na vrpci, ispisati ce poruku:

LOADING "T.MNOZENJA" B 0

"B" znaci da se ucitava BASIC program, a
"0" da se trenutno ucitava NULTI blok programa.

Ako program ima vise blokova, racunalo ce za svaki blok ponovo ispisati ime programa, "B" i BROJ BLOKA koji se trenutno ucitava.

Tek kad se na ekranu pojavi cursor, program je ucitan i spremam za rad. Mozemo ga izvrsiti na uobičajen nacin (RUN).

Ako vam, za vrijeme ucitavanja programa, racunalo ispise poruku:

LOADING ERROR

Znaci da je doslo do pojave greske kod ucitavanja. Probajte podesiti potenciometar za glasnocu reprodukcije. Provjerite da li ste pravilno spojili kazetofon i racunalo, a nakon toga ponovite postupak ucitavanja.

Osim ovakve mogucnosti snimanja programa, moguce je snimiti i SAMO DIO PROGRAMA, odnosno snimiti program od linije XXX do linije YYY, na primjer:

SAVE "PROBA" 500-2400

ce spremiti na vrpcu dio programa od linije 500 do linije 2400.

Program moze biti snimljen i na ovaj nacin:

```
10 A$= "TEST"  
20 SAVE A$
```

Dakle, kao ime programa moze doci bilo koji BASIC STRING

Kod naredbe LOAD, ime programa (koje zadajemo preko tastature) se usporedjuje s imenom programa na kazeti na toliko znakova koliko smo ih naveli uz LOAD naredbu, npr:

LOAD "A"

ce ucitati s kazete prvi program cije ime pocinje s "A".

Tako cemo pomocu:

LOAD ""

moci ucitati prvi program koji naidje.

Ako program s datim imenom nije na pocetku, racunalo ce na na to upozoriti porukom:

REWIND TAPE

Tada se kazeta moze premotati na pocetak programa. Nakon toga se pritisne <CR> za nastavak ucitavanja.

Ako u memoriji racunala vec postoji neki program, a mi ucitamo drugi, napravit ce se MERGE, odnosno ti ce se programi spojiti u jednu cjelinu.

TGAP naredba

=====

Zapis BASIC programa na kazeti je podijeljen u blokove. Izmedju njih postoji kratka pauza (GAP), koja omogucuje racunalu da obradi procitani blok prije nego sto pristigne novi. To je potrebno zato jer racunalo ne moze upravljati kazetofonom (START-STOP). Duzina pauze se inicializacijom BASIC-a postavlja na 3 sekunde, a moze se mijenjati naredbom:

TGAP X

kod cega je X broj koji predstavlja duzinu pauze (10 odgovara jednoj sekundi). Ako je pauza prekratka, moze doci do greske kod ucitavanja programa.

Za BASIC programe koji se sastoje od 40 i vise blokova, pauzu je potrebno PRODUZITI (npr. TGAP 40). Takav program snimili bi ovako:

```
TGAP 40 <CR>
SAVE "PROGRAM" <CR>
```

Ovako snimljen program ucitava se obicnom naredbom LOAD, dakle:

```
LOAD "PROGRAM" ili
LOAD ""
```

2.17. ISPITIVANJE TASTATURE

Potreba za ispitivanjem tastature javlja se gotovo u svakom programu. U tu svrhu sluzi nam naredba:

INKEY A\$

Prilikom izvrsavanja ove naredbe racunalo utvrdjuje koji je taster U TOM TRENTUKU bio pritisnut. Rezultat je string koji sadrzi znak pritisnutog tastera ili prazan string ako nije pritisnut niti jedan taster.

Razlika izmedju naredbi INPUT i INKEY je slijedeca: INPUT zaustavlja izvodjenje programa i racunalo ceka na upis podatka preko tastature. Cim pritisnemo <CR> nastavlja se s izvodjenjem programa.

Kad program dodje do naredbe INKEY B\$, u tom trenutku (koji traje vrlo kratko) se ispita cijela tastatura, tj. provjeri se da li je i (ako je) koji je taster bio pritisnut. Nakon toga nastavi se s izvodjenjem programa. Ako smo bas u tom trenutku pritisnuli taster "R", tada ce B\$ poprimiti vrijednost "R", odnosno bit ce B\$="R". Ako pak u tom trenutku nije bio pritisnut ni jedan taster, B\$ ce biti prazan string, ili B\$="" . Nije obavezno da uz naredbu INKEY stoji bas A\$ ili B\$. To moze biti i:

INKEY S\$, INKEY X\$, INKEY CD\$...

ZA PRODJEVIE SVIH ZNAKOMA

Prepisite ovaj primjer:

```
TO  
10 INKEY A$; IF A$="" THEN 10  
20 PRINT ASC(A$)  
30 GOTO 10
```

RUN (CZ)

Izvodjenje programa prekinite s **<CTL> <C>**.
Ovaj program ispisuje KOD pritisnutog tastera. Na ovom kratkom primjeru naučit ćete DVIJE vazne stvari vezane uz naredbu INKEY. Prva je:

- obratite pažnju na liniju 10:

```
10 INKEY A$; IF A$="" THEN 10
```

Ona se sastoji od dve naredbe odvojene dvotockom. U prvoj ispitujemo testaturu, a u drugoj provjeravamo da li je koji taster uopće bio pritisnut. Ako nije, tada je A\$="" pa program vracamo ponovo na pocetak linije 10. Znaci, linija 10 u ovakvom obliku "zaustavlja" izvodjenje programa. Tek kad pritisnemo BILO KOJI taster, program izlazi iz linije 10.

U mnogim programima korisno je formirati sekvencu sličnu ovoj:

```
...  
...  
250 PRINT "PRITISNI BILO KOJI TASTER ZA NASTAVAK"  
260 INKEY Q$; IF Q$="" THEN 260  
270 REM NASTAVAK PROGRAMA  
...
```

Ovako se zadržava neki ispis na ekranu tako dugo koliko to zeli onaj tko sjedi za računalom.

DRUGA vrlo vazna stvar je PREPOZNAVANJE PRITISNUTOG TASTERA. Prepisite slijedeci primjer:

```
10 INKEY A$; IF A$="" THEN 10  
20 B=ASC(A$)  
30 PRINT CHR$(B)  
40 GOTO 10
```

Pokrenite program. Pritisnite "A". Na ekranu se pojавilo "a". Pritisak na "F" daje "f". Prisjetite se znacenja funkcija ASC i CHR\$ i sami proučite program.

* Prepisite slijedeci primjer:

```
10 CLS  
20 INKEY A$  
30 IF A$="a" THEN PRINT "*";: GOTO 20  
40 PRINT ".";  
50 GOTO 20
```

Pokrenite program. Na ekranu se ispisuju samo točkice.

Pritisnite "A" i - na ekranu se ispisuju zvjezdice. Svaki pritisak na "A" ispisuje jednu zvjezdicu, inace se ispisuju točkice. Kad bi u liniji 30 umjesto "a" stavili "A", program ne bi prepoznavao taster A. Uvjerite se u to!

Kad smo, u jednom od prethodnih primjera, ispisivali kod pritisnutog tastera, pritisak na "A" dao je rezultat (kod) 97. Međutim, KOD velikog slova "A" nije 97, nego 65 ! Ne radi se o nikakvoj greski. 65 je KOD malog slova "a".

Dakle, kad u nasem programu vrsimo ispitivanje i kontrolu tastature pomocu naredbe INKEY A\$, pritisak na taster s nekim slovom pridružuje stringu A\$ vrijednost tog istog, ali MALOG slova. Zato trebamo paziti u linijama s IF A\$="..." u kojima provjeravamo koji je taster bio pritisnut - unutar navodnika treba stajati MALO slovo.

Naredba INKEY nalazi primjenu u gotovo svim programima, a sluzi, cemu bi drugo, za ispitivanje tastature. U svim igramu napisanim u BASIC-u je nezaobilazna. Ovu cemo naredbu od ovog trenutka cesto susretati u nasim primjerima.

2.18. GRAFIKA I ZVUK

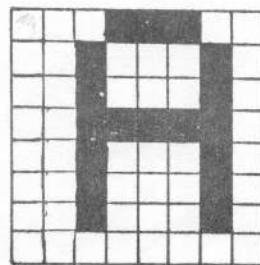
U ovom interesantnom poglavlju upoznat ćemo se s nekoliko novih naredbi:

MOVE, PLOT, DRAW, CIR, MODE, DOT, LETTER, SOUND

Najprije ćemo se upoznati s BASIC naredbama mikroracunala Orao koje služe za rad s GRAFIKOM. Naucit ćemo kako da crtamo razne slike, grafikone, krivulje ... Ali prije nego što pocnemo o grafici, moramo se prisjetiti poglavlja 2.12. u kojem smo se upoznali s formatom ekrana i PRINT funkcijama.

Orao može na ekranu prikazati sliku u 32 reda od kojih svaki ima 32 kolone. Na ekran može stati u jednom trenutku ukupno $32 \times 32 = 1024$ znaka. Sada nas zanima - od cega se sastoji jedan znak ?

Svaki znak se sastoji od osam redova tocaka, a svaki red ima opet osam kolona tocaka. Zato kazemo da je matrica jednog znaka 8×8 tocaka. Kad bi slovo "A" pogledali "pod povečalom", vidjeli bi da ono izgleda ovako:

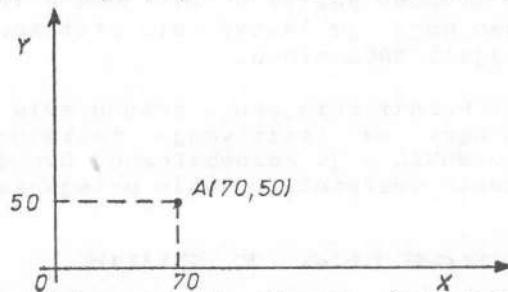


Ekran se sastoji od 32 reda znakova, a znak se sastoji od 8 redova tocaka. Iz ovoga nam nije teško izracunati koliko redova tocaka se ukupno nalazi na ekranu: $32 \times 8 = 256$.

Isto tako ćemo izracunati koliko kolona tocaka ima ekran: 32 kolone znakova, svaka po 8 kolona tocaka, što daje

ukupno $32 \times 8 = 256$. Da zaključimo: sliku na mikroracunalu Orao cini mreža od 256×256 točaka. To je grafika prilicno visoke rezolucije koja nam omogućuje da crtamo vrlo kvalitetne slike (kazemo da Orao ima grafiku visoke rezolucije od 256×256 točaka). Orao ne može crtati slike u bojama, grafika je crno-bijela (crna je pozadina na kojoj se iscrtavaju bijele točice).

Kad radimo s grafikom, ekran nam je prvi kvadrant Kartezijevog koordinatnog sustava, a svaka točka ekrana određena je pomocu dva broja koji se zovu KOORDINATE te tocke.



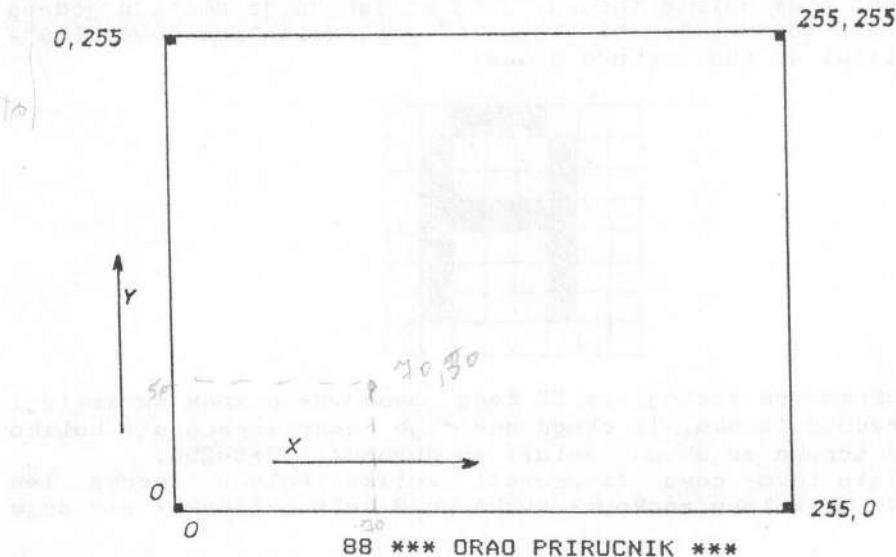
Najdonja lijeva točka ekrana ima koordinate $(0,0)$, a najdonja desna točka $(255,0)$.

Najgornja lijeva točka ekrana ima koordinate $(0,255)$, a najgornja desna točka je točka s koordinatama $(255,255)$.

Kad radimo s grafikom, zanima nas izgled ekrana u GRAFICKOM MODU, a pri radu s tekstom govorimo o ALFANUMERICKOM MODU.

Rezolucija ekrana u grafickom modu je 256×256 (točaka), dok je rezolucija ekrana u alfanumerickom modu 32×32 (znaka). "Prebacivanje" iz jednog u drugi mod nije potrebno - kad na ekranu želimo crtati, koristimo naredbe za rad s grafikom i rezoluciju 256×256 . Kad radimo s tekstom, rezolucija je ona "stara" - 32×32 . Naravno, moguce je istovremeno i jednostavno mijesanje slika i teksta.

Evo kako izgleda ekran u GRAFICKOM MODU:



Kao sto postoji cursor za tekst (odredjuje na kojem ce se mjestu na ekranu prikazati slijedeci znak), tako postoji i GRAFICKI KURSOR. Kad ispisujemo znakove na ekranu, cursor za tekst se automatski pomice. Isto vrijedi i za graficki cursor - svaka graficka naredba koju cemo izvrsiti pomaknut ce graficki cursor na odredjeno mjesto ekrana. Kad ukljucimo racunalo, graficki cursor se nalazi u donjem lijevom kutu ekrana, na koordinatama (0,0).

MOVE

====

Naredba MOVE sluzi za pomicanje grafickog cursora na zeljeno mjesto ekrana. Na primjer:

NE MOVE 100,150

ONA ce pomaknuti graficki cursor od POSETNE POZICIJE grafickog cursora (0,0) za 100 tocaka udesno i za 150 tocaka prema gore.

Dakle, naredba MOVE ima opci oblik:

MOVE X,Y

gdje prvi broj (X) označava koordinatu X, a drugi broj koordinatu Y. Koordinata X predstavlja broj koji nam govori za koliko tocaka UDESNO OD NULE pomicemo graficki cursor, a koordinata Y je broj koji predstavlja broj tocaka za koji pomicemo graficki cursor prema GORE od tocke u nultoj koloni X-tog reda.

Izvršenjem naredbe MOVE na ekranu se nije dogodilo nista vidljivo. Zbog toga se ona nikad ne koristi sama za sebe, nego prethodi ostalim grafickim naredbama.

Iza naredbe MOVE moraju se navesti dva broja odvojena zarezom koji označavaju koordinate jedne tocke ekrana. Za oba zarezom koji označavaju koordinate jedne tocke ekrana. Za oba dva broja vrijedi da ne smiju biti manji od 0, niti veci od 255. U protivnom, racunalo ce prijaviti gresku i stati s izvodjenjem programa.

PLOT

====

10 PLOT i MOVE su vrlo slicne naredbe. Pomocu naredbe:

PLOT X,Y

MOVE X,Y

se na mjestu ekrana s koordinatama X,Y crta TOCKA.

✓ Primjeri:

✓ PLOT 100,150 - crta tocku na koordinatama X=100, Y=150

✓ PLOT 0,0 - crta tocku u donjem lijevom kutu ekrana

✓ PLOT 255,0 - crta tocku u donjem desnom kutu ekrana
 ✓ PLOT 255,255 - crta tocku u gornjem desnom kutu ekrana
 ✓ PLOT 0,255 - crta tocku u gornjem lijevom kutu ekrana
 ✓ PLOT 127,127 - crta tocku u sredini ekrana

Primjer 1:
Nacrtat cemo 200 tocka sa slucajno odabranim koordinatama:

```

10 CLS
20 FOR N=1 TO 200
30 X= INT(RND(7)*256)
40 Y= INT(RND(7)*256)
50 PLOT X,Y
60 NEXT N
  
```

Primjer int (A.S) (CR)

Primjer 2:
Nacrtat cemo liniju (crtu) duz cijelog ekrana (duzine 255 tocka) crtajući tocku po tocku koji će imati koordinate X= od 0 do 255 i Y=100:

```

10 CLS
20 FOR X=0 TO 255
30 PLOT X,100
40 NEXT X
  
```

Primjer 3:
Nacrtat cemo graf funkcije SINUS (sinusoidu) za vrijednosti izmedju 0 i 2PI:

```

10 CLS
20 PI=3.14159
30 FOR N=0 TO 255
40 PLOT N,(100+80*SIN(N/128*PI))
50 NEXT N
  
```

Primjer 4:
Crtanje eksponencijalne funkcije naredbom PLOT:

```

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

2. Naredba

DRAW

=====

Pomoću naredbe

DRAW X,Y

povlacićemo LINIJU od trenutne pozicije grafickog kursora do tocke cije smo koordinate odredili brojevima X i Y iza naredbe DRAW.

Pocetna tocka linije određuje se pozicijom grafickog kursora, a poziciju grafickog kursora određujemo naredbom

MOVE 0,0; DRAW 0,32; DRAW 255,32; DRAW 255,0
MOVE 0,0; DRAW 255,0

MOVE. Vrijednosti koje navodimo iza naredbe DRAW predstavljaju zavrsnu tocku linije.

Primjer 1:

Nacrtat cemo diagonalu ekrana od donjeg lijevog do gornjeg desnog kuta ekrana:

10 MOVE 0,0 *menidi se tocka ali slavi da se*
20 DRAW 255,255 *ad me itma nije spuka*

Linija 10 postavit ce graficki cursor u donji lijevi kut ekrana, tj. u tocku s koordinatama (0,0), a linija 20 ce povuci crtu s pocetkom u tocki (0,0) i zavrsetkom u tocki (255,255). Nakon zavrsetka izvodjenja naredbe DRAW, pozicija grafickog cursora nalazi se upravo u tocki koju smo zadnju nacrtali, a to je tocka (255,255).

Primjer 2:

Sada cemo, osim dijagonale, nacrtati i kvadrat (okvir ekrana):

10 CLS
20 MOVE 0,0
30 DRAW 255,255
40 DRAW 0,255: DRAW 0,0
50 DRAW 255,0: DRAW 255,255



Primjecujete, vjerojatno, da smo pocetnu poziciju grafickog cursora pomocu naredbe MOVE odredili samo jedanput, na pocetku programa.

Unutar jedne naredbe DRAW moze se navesti i vise parametara, a njihov broj MORI BITI PARAN (2,4,6,...). Broj parametara koje navodimo unutar jedne naredbe DRAW ogranicen je samo najvecem dozvoljenom duzinom programske linije (72 znaka).

Koristeci naredbu DRAW na ovaj nacin, mnogo lakse cemo nacrtati sliku iz prethodnog primjera:

Primjer 3:

10 CLS
20 MOVE 0,0
30 DRAW 255,255,0,255,0,0,255,0,255,255

Argumenti naredbe DRAW mogu takodjer biti dani u obliku varijabli ili izraza. Evo jos nekoliko primjera upotrebe naredbe DRAW:

Primjer 4:

2 PRH
Nacrtat cemo kvadratnu mrezu na ekranu. Promjenom vrijednosti iza naredbe STEP u liniji 20 mozete dobiti mrezu razlicite gustoce.

10 CLS
20 FOR N=0 TO 255 STEP 5
30 MOVE N,0
40 DRAW N,255
50 MOVE 0,N
60 DRAW 255,N
70 NEXT N

Primjer 5:

Crtanje eksponencijalne funkcije naredbama MOVE i DRAW:

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

Primjer 6:

Ovaj program crta CASU:

5 REM CASA
6 REM ====
7 REM
10 CLS
20 S=128: V=200
30 A=50: B=25
40 K=0.1
50 FOR T=0 TO 6.2 STEP K
60 X=A*COS(T)
70 Y=B*SIN(T)
80 MOVE X+S,Y+B+5
90 DRAW X/4+S,Y/4+B+15
100 DRAW X*1.7+S,Y+V
110 NEXT T

Pogledajte kako izgleda casa kad se promijeni vrijednost za K u liniji 40 (probajte npr. K=0.03, K=0.2, ...).

CIR

==

Naredbom CIR crtamo KRUZNICU (engl. "CIRCLE"=krug).
Opci oblik naredbe CIR je slijedeci:

CIR X,Y,R

Iza CIR dolaze tri parametra, tri broja. Prva dva određuju KOORDINATE SREDISTA kruznice, a treći određuje POLUMJER kruznice.

Ti se parametri (argumenti naredbe CIR) također mogu dati u obliku varijabli ili izraza.

70 Primjer 1:
Nacrtajmo kruznici s koordinatama X=150, Y=120 i
polumjerom R=50:

70 CIR 150,120,50 <CR>

70 Primjer 2:
Najveća kruznica izgleda ovako:

70 CIR 127,127,127

70 Probajte, npr.:

70 CIR 127,127,200

Ako je treci broj (polumjer) veci od 128, tada se kao polumjer kruznice uzima razlika izmedju broja 256 i zadanog polumjera. U ovo slučaju nacrtani krug ima polumjer $(256-200)=56$.

Primjer 3:

Nacrtajmo nekoliko koncentričnih kruznica (to su kruznice koje imaju zajednicko srediste):

70 10 FOR R=10 TO 100 STEP 10

70 20 CIR 127,127,R

70 30 NEXT R

70 MODE

70 =====

Mikroracunalo Orao posjeduje mogućnost nekoliko načina crtanja. Izbor zeljenog načina crtanja vrši se pomoći naredbe MODE. Ali, sto je to "način crtanja"?

Vidjeli smo dosad da Orao crta bijelom bojom na crnoj pozadini. Kad bi htjeli obrisati nacrtanu (bijelu) sliku, ali ne pomoći naredbe CLS, morat ćemo istu sliku ponovo nacrtati, ali crnom bojom na crnoj podlozi, što će izgledati kao brišanje slike.

Naredbom:

70 MODE 1

70 prebacujemo se u ovaj način crtanja, odnosno crtati ćemo crnom bojom na crnoj podlozi.

Povratak u osnovni način crtanja (bijela boja na crnoj pozadini) vrši se naredbom:

70 MODE 0

Osim ova dva, postoji još jedan način crtanja, inverzni. To znači da će se crtati bijelom bojom na crnoj podlozi, a crnom bojom na bijeloj podlozi. U inverzni način crtanja prebacujemo se naredbom

70 MODE 128

Ukratko, izbor nacina crtanja vrsi se pomocu naredbe:

MODE X

X=0 - crta se bijelo na crnom
0<X<128 - crta se crno na bijelom
X>127 - crta se inverzno

Primjer 1:

Crtanje i brisanje kvadrata:

```
10 MODE 0: GOSUB 50
20 MODE 1: GOSUB 50
30 GOTO 10
50 MOVE 50,50: DRAW 100,50,100,100,50,100,50,50
60 RETURN
```

Primjer 2:

Crtanje i brisanje koncentricnih kruznica:

```
10 MODE 0: GOSUB 50
20 MODE 1: GOSUB 50
30 GOTO 10
50 FOR R=5 TO 50 STEP 5
60 CIR 127,127,R
70 NEXT R
80 RETURN
```

Primjer 3:

Nacrtat cemo ponovo kvadratnu mrezu na ekranu koristeci inverzni nacin crtanja:

```
10 MODE 0
20 FOR N=0 TO 255 STEP 8
30 MOVE N,N: DRAW N,255
40 NEXT N
50 MODE 200: REM INVERZNO
60 FOR N=0 TO 255 STEP 8
70 MOVE 0,N: DRAW 255,N
80 NEXT N
90 MODE 0
```

Prije nego sto pokrenete ovaj program, izlistajte ga na ekranu. Sad ga izvrsite. Vidjet cete da se svako slovo nalazi unutar polja od 8*8 tocaka (naredba STEP 8 u linijama 20 i 60). Obratite paznju na sjeciste linija. Izvrsite ovaj program nekoliko puta za redom bez brisanja ekrana.

NP DOT funkcija
=====

Osim grafickih naredbi koje smo dosad upoznali, postoji i jedna funkcija koja nam sluzi za rad s grafikom. To je funkcija DOT. Po ulozi je slicna funkciji SCREEN\$ koja cita znak s odredjene pozicije na ekranu.

Funkciju DOT koristimo za ispitivanje OSVIJETLJENOSTI neke tocke ekrana. Na tocku koju zelimo ispitati, postavimo graficki cursor naredbom MOVE. Ako je tocka koju ispitujemo osvijetljena, dobit cemo rezultat 1. Ako nije osvijetljena rezultat ce biti 0.

Primjer :

Ispitajmo stanje tocke (150,200):
10 MOVE 150,200
20 A=DOT
30 PRINT A

Ako je tocka (150,200) osvijetljena (bijela), A ce poprimiti vrijednost 1. Inace je A=0.

Slijedeci primjer mogao bi biti osnova vaseg vlastitog programa za crtanje. Ovo je program pomocu kojeg mozemo crtati i brisati sliku po cijelom ekranu. Crtanje i brisanje se izvodi tocku po tocku na poziciji odredjenoj pozicijom grafickog cursora.

Komande za rad s programom su:

Q - gore
A - dolje
O - lijevo
P - desno
B - uključuje se brisanje
C - povratak na crtanje

X - kraj programa

```
10 REM PROGRAM ZA CRTANJE
20 VDU: MODE 0
30 X=127: Y=127
40 PLOT X,Y
50 INKEY A$
60 IF A$="q" THEN Y=Y+1: GOTO 200
65 IF A$="a" THEN Y=Y-1: GOTO 200
70 IF A$="o" THEN X=X-1: GOTO 200
75 IF A$="p" THEN X=X+1: GOTO 200
80 IF A$="b" THEN MODE 1
85 IF A$="c" THEN MODE 0
90 IF A$="x" THEN PRINT "KRAJ": MODE 0: END
99 GOTO 50
200 IF X>255 THEN X=255
210 IF Y>255 THEN Y=255
220 IF X<0 THEN X=0
230 IF Y<0 THEN Y=0
240 GOTO 40
```

ANIMIRANA GRAFIKA

Pojam animirane grafike se odnosi na crtanje objekata koji se pokreću na grafickom zaslonu, tj. ekranu. Da bi se objekti mogli pokretati, postupamo na sljedeći nacin:

- KORAK 1 - Nacrtamo zeljeni objekt u prvoj fazi pokreta i prikazujemo ga T₁ vremena
- KORAK 2 - Brisemo nacrtani objekt djelimično ili potpuno
- KORAK 3 - Crtamo objekt u drugoj fazi pokreta na istoj ili promijenjenoj poziciji i prikazujemo ga T₂ vremena
- KORAK 4 - Da li su sve faze pokreta prikazane?
 - ako NISU idi na KORAK 2
 - ako JESU idi na korak 5
- KORAK 5 - kraj animacije

Brisanje objekta na ekranu omogućava nam promjena grafičkog moda, odnosno promjena nacina crtanja (naredba MODE).

Primjer :

Gibanje radij vektora po kružnom luku:

```
10 REM
20 REM ANIMACIJA REDIJ VEKTORA
30 REM
40 CLS
45 K=2*3.141593/256
50 FOR T=0 TO 63 STEP 5
55 Y=100*SIN(K*T)
60 X=127*COS(K*T)
65 MODE 0
70 GOSUB 200
75 MODE 1
80 GOSUB 300
85 GOSUB 200
90 NEXT T
95 MODE 0
97 GOSUB 200
99 END
200 MOVE 0,0: DRAW X,Y
210 RETURN
300 FOR A=1 TO 400: NEXT A
310 RETURN
```

-10-

ZVUK - naredba SOUND

Ovu naredbu dosad smo već nekoliko puta upotrijebili u dosadašnjim primjerima. Naredba SOUND proizvodi zvučni signal iz zvučnika ugradjenog u mikroracunalu. To je ton kojem odbiremo visinu i duljinu trajanja.

Naredba SOUND izgleda ovako:

SOUND V,D

Prvi parametar (V) predstavlja VISINU tona, a drugi (D) duzinu tona. Dozvoljene vrijednosti za oba dva parametra mogu biti izmedju 0 i 255. Dakle:

SOUND V,D

V - visina tona $0 \leq V \leq 255$
D - duzina tona $0 \leq D \leq 255$

Naredbom SOUND mogu se dobiti razliciti i vrlo interesantni zvucni efekti koji su nezaobilazni dio gotovo svakog dobrog programa. Naredbu SOUND mozemo koristiti pri pisanju igara, kao zvucni signalizator da li je pritisnut neki taster, mozemo generirati razne zvukove u slucaju da je odgovor na postavljeno pitanje bio tocan (ili netocan)...

Isprobajte nakoliko slijedeci zvucnih efekata:

Primjer 1:

10 FOR V=10 TO 200 STEP 5
20 SOUND V,10
30 NEXT V

Primjer 2:

10 FOR V=150 TO 50 STEP -1
20 SOUND V,8
30 SOUND 30,8
40 NEXT V

Primjer 3:

10 FOR X=1 TO 3
20 FOR V=160 TO 60 STEP -5
30 SOUND 0,7
40 SOUND V,7
50 SOUND 25,7
60 NEXT V
70 NEXT X

Pomocu naredbe SOUND mozemo komponirati lakse melodije. Da bi to mogli, moramo znati potrebne visine tonova za odredjene note. Da bi vam olaksali posao komponiranja, napisat cemo vrijednosti koje treba navoditi u naredbi SOUND kao parametar visine tona:

242	215		181	161	144	121	108		
0	228	203	192	171	152	136	128	114	102

Primjer:
Odsvirat cemo C-dur ljestvicu (duzina tona moze se odabratи):

```
10 REM MUZIKA  
15 REM -----  
20 INPUT "DUZINA TONA (1-255)";D  
30 IF D>255 THEN 20  
40 FOR X=1 TO 16  
50 READ V  
60 SOUND V,D  
70 NEXT X  
80 DATA 0,228,203,192,171,152,136,128  
90 DATA 128,136,152,171,192,203,228,0
```

LETTER

U programima se cesto javlja potreba da se dio teksta na ekranu posebno istakne. Jedan od nacina da se to postigne je da taj dio teksta napisemo inverzno, pomocu naredbe INV.

Zeljeni dio teksta mozemo ispisati i UVECANIM SLOVIMA, a u tu svrhu koristimo naredbu LETTER.

Opci oblik te naredbe je:

LETTER A\$,X,Y

- A\$ - tekst koji se zeli ispisati
- X - VELICINA znakova izrazena visekratnikom standardne velicine (X moze biti od 1 do 32)
 - npr. ✓ X=1 - normalna velicina znaka
 - ✓ X=2 - dvostruka velicina znaka
 - ✓ X=32 - jedno slovo preko cijelog ekrana
- Y - ORIJENTACIJA teksta koji se ispisuje na ekranu
 - Y=0 - ispis s lijeva na desno
 - Y=1 - ispis odozdo prema gore
 - Y=2 - ispis s desna na lijevo
 - Y=3 - ispis odozgo prema dolje

Pocetno mjesto ispisivanja zeljenog teksta definira se prethodno pomocu grafickog kursora. Ustvari, tekst koji navodimo uz naredbu LETTER se ne ispisuje, nego se iscrtava. Zato pocetak "ispisivanja" ne odredujemo pozicijom kursora za tekst, nego pozicijom grafickog cursora.

Naredbom MOVE X,Y postavit cemo graficki cursor na mjesto pocetka ispisa i to tako da je pocetna tocka ispisa uvjek DONJA LIJEVA tocka prvog znaka u stringu koju bi znak imao u otijentaciji 0.

Primer 1:

Ispisat cemo rijec ORAO cetiri puta uvecanim slovima i to s lijeva na desno od tocke 100,100:
10 MOVE 100,100
20 LETTER "ORAO",4,0

Isto bi bilo kad bi napisali:

10 A\$="ORAO"
20 MOVE 100,100
30 LETTER A\$,4,0

Primer 2:

10 MOVE 200,50
20 LETTER "ORAO",3,0

Pazite da vam se u vasim programima ne dogadjaju ovakve stvari. Vrlo lako moze se izracunati od koje je tocke potrebno zapoceti ispisivanje teksta uvecanim slovima. Kako?

Uzmimo bas ovaj primer 2. Kako da rijec ORAO napisemo u sredini reda, tri puta uvecanim slovima? Znamo da je i visina i sirina znaka normalne velicine 8 tocaka, a znamo i to da jedan red ekrana ima 256 tocaka (od 0 do 255). Naredbom LETTER uvecali smo sva slova TRI PUTA - znaci da nam je jedno slovo sirine (i visine) $8 \times 3 = 24$ tocke. Sva cetiri slova (ORAO) zajedno zauzimaju $24 \times 4 = 96$ tocaka. Ako oduzmemos 256-96 dobijamo broj koliko ostaje "praznih" tocaka u tom redu. Ovdje je to 160 tocaka. Ako taj broj podijelimo s dva, dobijamo tocku (koordinatu X) od koje treba poceti ispisivati "ORAO" tako da se ispis nadje u sredini reda. To je tocka $160 : 2 = 80$, odnosno X koordinata tocke u naredbi MOVE X,Y.

Probajte sad:

10 MOVE 80,50
20 LETTER "ORAO",3,0

Da bi se rijec "ORAO" nasla, osim u sredini reda, i u sredini ekrana, potrebno je jos izracunati koordinatu Y. Ona bi bila $(256-24) : 2 = 116$. Zasto 256-24? Zato sto je 24 visina jednog znaka uvecanog tri puta. Provjerite sad:

10 MOVE 80,116
20 LETTER "ORAO",3,0

Primjer 3:

Evo vrlo zgodnog primjera kako da u svom programu istaknete naslov:

```
10 CLS  
20 A$="NASLOV"  
30 MOVE 32,200  
40 LETTER A$,4,0  
50 MODE 128  
60 FOR N=196 TO 236  
70 MOVE 0,N  
80 DRAW 255,N  
90 NEXT N  
95 MODE 0
```

Primjer 4:

Ispis odozgo prema gore:

```
10 MOVE 200,64  
20 LETTER "DRAO",4,1
```

Ispis s desna u lijevo:

```
10 MOVE 208,100  
20 LETTER "DRAO",5,2
```

Ispis odozgo prema dolje:

```
10 MOVE 150,150  
20 LETTER "DRAO"
```

Jedno slovo preko cijelog ekrana:

```
10 MOVE 0,0  
20 LETTER "A",32,0
```

2-19. RAD S MEMORIJOM

U ovom poglavlju naucit ćete devet novih, vrlo interesantnih, BASIC naredbi mikroracunala Draqo koje služe za rad s memorijom. To su slijedeće naredbe:

FRE, POKE, PEEK, CHAR, SMOVE, LNK, USR, DMEM, LMEM

Da bi mogli razumjeti čemu koja naredba sluzi, bit će potrebno da se najprije ukratko upoznate s nekim osnovnim pojmovima kao sto su: brojevni sustavi, prevodjenja brojeva iz jednog u drugi brojevni sustav, memorijska lokacija, memorijska mapa mikroracunala Draqo ...

BROJEVNI SUSTAVI

Osnovni zahtjevi koje racunalo treba ispuniti jesu mogućnost prikazivanja i pohranjivanja brojeva, te izvršavanje aritmetickih i logickih operacija s tim brojevima. Brojevi

mogu biti prikazani u razlicitim brojevnim sustavima. Mi smo nauceni upotrebljavati decimalni brojevni sustav, vjerojatno zbog nasih deset prstiju.

DECIMALNI brojevni sustav

Osnova (BAZA) decimalnog brojevnog sustava je DESET, jer se upotrebljava deset znamenaka (0,1,2,3,4,5,6,7,8,9). Dajuci znamenkama odredjene polozae, odnosno TEZINE, mogu se izraziti i brojevi veci od deset.

Baza deset znaci da se pri brojanju poslije svakih deset jedinica vrsti prijenos u visi tezinski razred. U decimalnom brojevnom sustavu tezine pojedinih razreda su 1,10,100,1000... i one ustvari predstavljaju potencije baze deset: $10^0, 10^1, 10^2, \dots$

Broj 128 se u decimalnom brojevnom sustavu prikazuje kao $1*10^2 + 2*10^1 + 8*10^0 = 100+20+8 = 128$

BINARNI brojevni sustav

Racunala mogu biti izgradjena na osnovi bilo kojeg brojevnog sustava. Međutim, sva suvremena racunala zasnivaju se na binarnom brojevnom sustavu koji ima BAZU 2. Zasto se upotrebljava bas taj brojevni sustav? Zato sto je lakse razlikovati samo dve pojave umjesto deset. Vecina fizikalnih pojava razlikuje samo dva stanja, kao: zarulja svjetli ili ne, sklopka je uklopljena ili iskllopjena, materijal je namagnetiziran ili nije, struja je pozitivna ili negativna... Lakse je izraditi, a i pouzdaniji su sklopovi koji razlikuju samo dva stanja (binarno 0 i binarno 1), nego npr. deset stanja.

Dakle, binarni brojevni sustav zasnovan je na bazi dva. Jedine moguce znamenke su 0 i 1.

decimalno binarno

0	0
1	1
2	10
3	11

4 100

Poslije dvije jedinice vrsti se prijenos u visi tezinski razred. Tezine pojedinih razreda su potencije baze dva: $2^0, 2^1, 2^2, 2^3, \dots$ a to je $1, 2, 4, 8, 16, \dots$

decimalno binarno

0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001

itd.

1 : 2 = 0
2 : 2 = 0
1 : 2 = 1
3 : 2 = 1
4 : 2 = 1
1 : 2 = 1
2 : 2 = 0
1 : 2 = 0
2 : 2 = 0
1 : 2 = 1

101 *** ODAO PRIRUCNIK ***

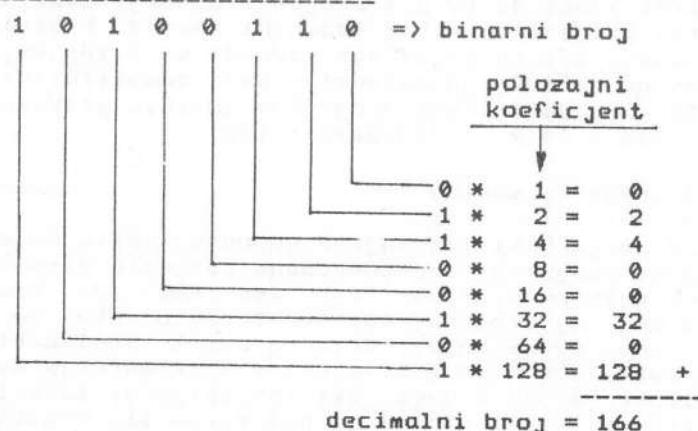
$$\begin{array}{l} 9 : 2 = 1 \\ 7 : 2 = 0 \\ 2 : 2 = 0 \\ 1 : 2 = 1 \end{array}$$

X Uzastopna brojevna mesta imaju tezine:

$$\dots 2^7, 2^6, 2^5, 2^4, 2^3, 2^2, 2^1, 2^0$$

Ova "tablica" tezina upotrebljava se za pretvaranje binarnih brojeva u, nama blizi, decimalni sustav. Na primjer, pronadjimo decimalni broj koji je jednak binarnom broju 10100110:

$$128 \ 64 \ 32 \ 16 \ 8 \ 4 \ 2 \ 1 \\ 2^7 \ 2^6 \ 2^5 \ 2^4 \ 2^3 \ 2^2 \ 2^1 \ 2^0 \Rightarrow \text{tezinski razredi}$$



Tako smo binarni broj 10100110 pretvorili u odgovarajući decimalni broj (166).

A kako da decimalni broj pretvorimo u binarni? Vrlo lako. Sada ćemo decimalni broj 166 pretvoriti u ponovo u binarni:

OSTATAK			
166 : 2 = 83	--->	0	
83 : 2 = 41	--->	1	
41 : 2 = 20	--->	1	smjer u kojem citamo binarni broj
20 : 2 = 10	--->	0	
10 : 2 = 5	--->	0	
5 : 2 = 2	--->	1	
2 : 2 = 1	--->	0	
1 : 2 = 0	--->	1	

binarni broj = 10100110

X HEKSADECIMALNI brojevni sustav

✓ Heksadecimalni brojevni sustav zasnovan je na bazi 16. Znamenke heksadecimalnog vrojevnog sustava su 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F, dakle 16 znamenaka.

$$\begin{array}{ll} \text{decimalno} & - \quad 0 \ 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9 \ 10 \ 11 \ 12 \ 13 \ 14 \ 15 \\ \text{heksadecimalno} & - \quad 0 \ 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9 \ A \ B \ C \ D \ E \ F \end{array}$$

Poslije 16 jedinica izvrsit će se prijenos u visi tezinski razred:

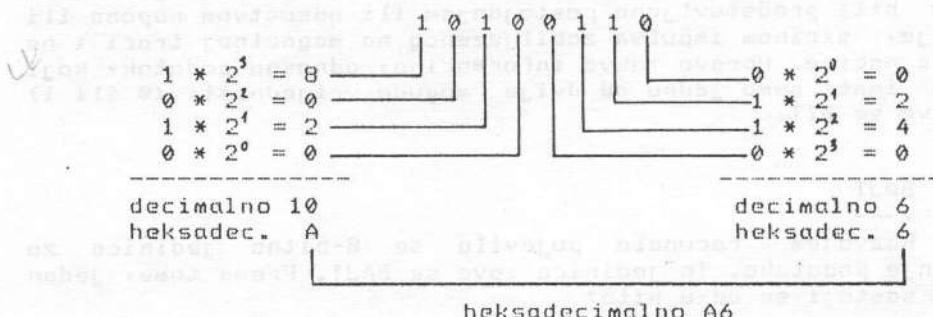
decimalno heksadecimalno

1	1
10	A
15	F
16	10
31	1F
32	20
itd...	

Tezine pojedinih razreda su $1, 16, 256, \dots$ i one predstavljaju potencije baze 16 ($16^0, 16^1, 16^2, \dots$). $16 \cdot 16 = 256$

Sada ćemo decimalni broj 166 prikazati u heksadecimalnom brojevnog sustavu:

$$166 \text{ decimalno} = 10100110 \text{ binarno}$$



$$\underline{\text{decimalno } 166 = \text{binarno } 10100110 = \text{heksadecimalno A6}}$$

Kako da heksadecimalni broj A6 prikazemo decimalno?
Prvi nacin, pomocu binarnog brojevnog sustava:

$$\begin{aligned} A6 &\rightarrow A = 10 = 1010 \\ &\rightarrow 6 = 6 = 0110 \end{aligned}$$

$$\text{Prema tome, } A6 = 10100110$$

Sada ćemo binarni broj 10100110 vrlo lako pretvoriti u decimalni broj na ranije opisan nacin.

Drugi nacin, direktno iz heksadecimalnog u decimalni sustav, je mnogo jednostavniji:

$$\begin{aligned} A6 &= 10 * 16^1 + 6 * 16^0 \\ &= 10 * 16 + 6 * 1 \\ &= 160 + 6 \\ &= 166 \end{aligned}$$

Da bi mogli razumjeti BASIC naredbe mikroracunala Orao koje služe za rad s memorijom, dovoljno će biti da (dobro) poznajete samo binarni brojevni sustav (uz decimalni, naravno). Neće biti uzalud ako naučite i heksadecimalni. Kad počnete programirati na strojnem jeziku, morat ćete zaboraviti decimalni sustav i savršeno savladati heksadecimalni.

BINARNI BROJ, BIT i BAJT

Unutar racunala svi podaci su predstavljeni samo brojevima koji se sastoje od nula i jedinica, odnosno predstavljeni su binarnim brojevima. Sva slova, znakovi, brojevi i naredbe se predstavljaju, odnosno kodiraju, pomocu binarnih brojeva. U takvom obliku racunalo ih pamti i obradjuje.

BIT

Jedna binarna znamenka obично се назива BIT. Тако се број 1011 сматра 4-битним бројем, а број 10100110 8-битним бројем. Крајњи леви бит бинарног броја назива се НАЈЗНАЧАЈНИЈИ BIT и њему припада највећа вредност. Крајњи десни бит зове се НАЈМАЊЕ ЗНАЧАЈАН BIT и припада му најмања вредност.

Fizicki gledano, za racunalo je informacija data jednim od dva moguca stanja. Stanja oznacena nulom ili jedinicom mogu biti predstavljena postojanjem ili odsustvom napona ili struje, sirinom impulsa zabiljezenog na magnetnoj traci i na druge nacine. Upravo takva informacija, odnosno podatak, koji moze imati samo jednu od dvije moguce vrijednosti (0 ili 1) naziva se BIT.

BAJT

Razvojem racunala pojavila se 8-bitna jedinica za cuvanje podataka. Ta jedinica zove se BAJT. Prema tome, jedan BAJT sastoji se od 8 bita:

bit7								bit0
1	0	1	0	0	0	1	1	0
bit najveće tezine							bit najmanje tezine	

Sadrzaj jednog bajta je 8-bitni broj koji moze biti:
- najmanje 00000000 = 0 decimalno
- najvise 11111111 = 255 decimalno
- svi brojevi izmedju 0 i 255

MIKROPROCESOR I MEMORIJA

Mikroracunalo se sastoje od integriranih krugova koji razlikuju samo dvije razlike vrijednosti napona, jedna odgovara nuli, a druga jedinici. Svaki od tih integriranih krugova sastoji se od desetak pa do nekoliko tisuća tranzistora "zapakiranih" u plasticno ili keramicko kućište. Na taj nacin dobija se cip.

MIKROPROCESOR upravlja radom cijelog računala. To je tzv. centralna procesorska jedinica, zapakirana u samo jedan integrirani krug. On iz memorije redom cita i izvršava naredbu po naredbi. Mikroprocesor mikroracunala Orao ima

oznaku "6502" i obradjuje istovremeno 8 bita (1 bajt) informacija, pa se zbog toga naziva 8-bitni mikroprocesor.

MEMORIJA sluzi za pohranjivanje programa i podataka. Sastoje se od integriranih krugova koji sadrze veliki broj memorijskih celija. U svaku memorijsku celiju moze se pohraniti jedan bit, a osam celija cini jednu cjelinu koja se zove MEMORIJSKA LOKACIJA. Svaka lokacija ima svoj "kucni" broj koji se zove ADRESA.

Najvaznije vrste memorija koje se susrecu u svakom mikroracunalu su RAM i ROM memorije.

RAM (Random Access Memory - memorija sa slobodnim pristupom) je ona memorija kod koje se moze slobodno pristupiti bilo kojoj grupi memorijskih celija, odnosno bilo kojoj memorijskoj lokaciji. U bilo koju lokaciju RAM-a podatak se moze upisati ili iz nje pročitati. S nestankom napajanja (struje) svi podaci koji se nalaze u RAM-u se nepovratno

brisu. Mikroracunalo Orao ima nesto vise od 32000 lokacija RAM memorije (tocnije 32×1024 ili 32 klobajta). U svaku od tih 32000 lokacija mozemo slobodno upisivati podatke (i citati ih). Ova kolicina RAM memorije je sasvim dovoljna da se napisu i vrlo ozbiljni i kvalitetni programi. 32469

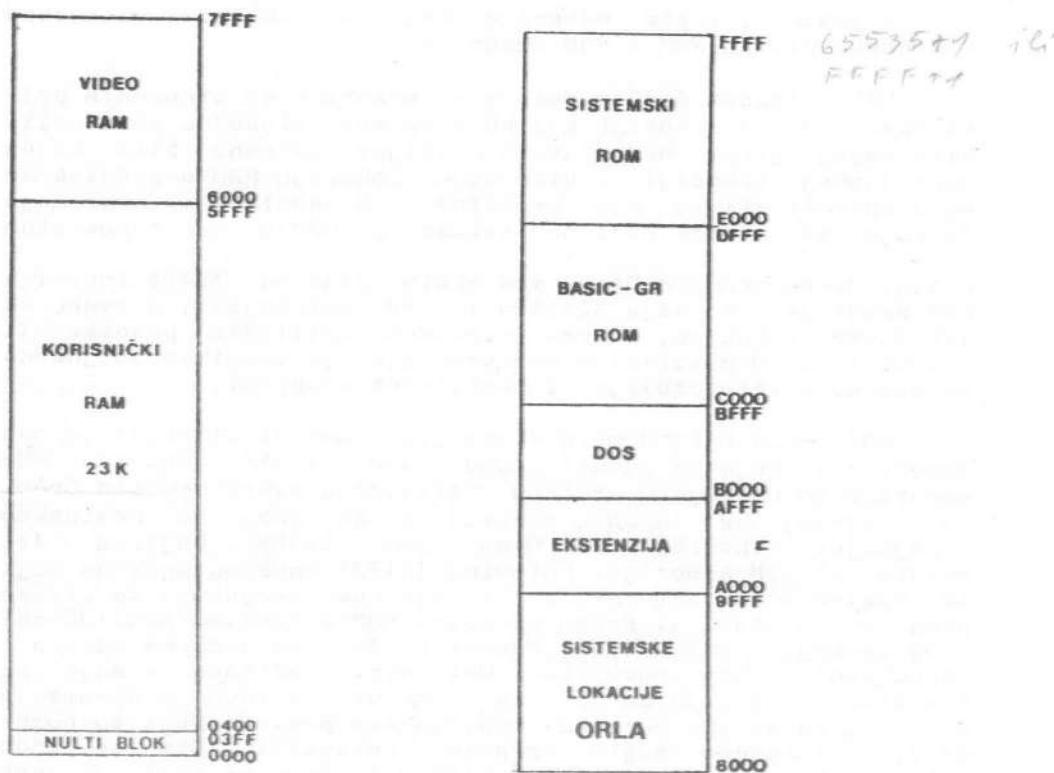
ROM (Read Only Memory = memorija samo za citanje) je ona memorija iz koje se podaci mogu samo citati. Sadrzaj ROM memorije se upisuje u procesu proizvodnje mikroracunala Orao. Taj sadrzaj je TRAJNO zapisan i ne gubi se nastankom napajanja. Mikroracunalo Orao ima 16384 bajtova (16 kilobajta) ROM memorije. Polovicu (8192) zauzima program koji se naziva BASIC interpreter i koji nam omogucuje da pisemo programe u BASIC-u. Drugu polovicu ROM-a zauzima MONITOR-ski program koji "daje život" racunalu. Taj se program naziva i operativni sistem racunala. On, npr., ucitava stanje na tastaturi, ispisuje poruke na ekranu, ucitava programe s vrpce i snima ih, ukratko, obavlja sve poslove koji su nuzni da bi se racunalo moglo normalno koristiti. Cim se Orao ukljuci, kontrolu preuzima monitorski program ciji je znak zvjezdica (*) koja se pojavi na ekranu odmah nakon uključivanja ili resetiranja racunala.

MEMORIJSKA MAPA MIKRORACUNALA ORAO

Za lakse razumijevanje BASIC naredbi koje se koriste za rad s memorijom (a i kasnije, kad ćemo upoznavati MONITORski program), moramo poznavati memorijsku mapu racunala koja nam pokazuje kakva je organiziranost racunala s obzirom na koristene memorijske lokacije.

MEMORIJSKA MAPA MIKROVACUNALA ORAO (organizacija
memorijskih lokacija unutar mikrovacunala Orao):

6502



Mikroprocesor 6502 moze direktno adresirati najvise 65535+1 memorijskih lokacija ili FFFF+1 u heksadecimalnoj notaciji.

NULTI BLOK (0000-03FF / 0-1023) je predvidjen za korištenje u sklopu sistemske memorije racunala koja je rezervirana za omogucavanje rada mikroprocesora 6502 i BASIC-a. Prilikom rada u BASIC-u, svi programi pocinju, odnosno nalaze se u memorijском prostoru iznad nultog bloka, a brigu oko rasподјеле i potrošnje memorije kod pisanja programa u BASIC-u vodi operativni sistem. 23532

KORISNICKI RAM (0400-5FFF / 1024-24575) sastoji se od 23552 slobodnih memorijskih lokacija, ili 23552 bajtova memorije. Prvi bajt korisnicke RAM memorije ima adresu 1024, a posljednji 24575. Svi programi napisani u BASIC-u automatski se smjestaju u ovaj dio memorije.

VIDEO RAM (6000-7FFF / 24576-32767) sastoji se od 8192 memorijskih lokacija (8 kilobajta) u kojima se nalazi trenutni sadržaj ekrana.

Dio memorije od 8000-9FFF predvidjen je za sistemske lokacije mikroracunala Orao namijenjene za komunikaciju s periferijskim jedinicama.

Područje od C000 do DFFF zauzima BASIC interpreter, a dio memorije od E000 do FFFF zauzima sistemske software (MONITOR).

FRE naredba

=====

Prva u nizu naredbi koje služe za rad s memorijom, je naredba FRE. Pomocu nje saznajemo kolicinu slobodnog memorijskog prostora u području korisnickog RAM-a.

Ako se nalazite u BASIC-u, napisite:

PRINT FRE(X) <CR>

i na ekranu će se ispisati broj koji predstavlja broj SLOBODNIH memorijskih lokacija (bajtova) koji imamo na raspolaganju u memoriji Orla.

Izbrisite sadržaj cijele memorije (*BC). Provjerite kolicinu slobodne memorije. Sada napisite nekoliko linija BASIC programa (makar nekoliko REM naredbi), a nakon toga opet ispitajte kolicinu slobodne memorije. Broj slobodnih memorijskih lokacija sada je manji, što znači da se sa svakom unesenom programskom linijom kolicina slobodne memorije smanjuje. No, ne brinite. Memorije ima sasvim dovoljno. Kad napišete neki "vrlo dugacak" program, vidjet ćete da je veći dio memorije još uvijek slobodan.

Naredba FRE koristi se samo u direktnom nacinu rada. To znači da se ona uglavnom koristi za vrijeme pisanja, odnosno razvijanja programa, kao kontrola utrosene memorije.

Kad unesete neki program u racunalo, ispitajte kolicinu slobodne memorije prije nego sto pokrenete program. Upamtite

taj broj. Nakon toga pokrenite program i u jednom trenutku ga prekinite (ili pricekajte da se sam završi). Ispitajte sada kolicinu slobodne memorije. Broj slobodnih bajtova je manji, što znači da se i u toku izvršavanja programa "potrosi" dodatna kolicina memorije.

Argument unutar zagrade koji se navodi uz naredbu FRE može biti bilo koje (veliko) slovo, npr:

PRINT FRE(U), PRINT FRE(W),...

POKE

=====

Memorija svakog računala sastoji se od određenog broja memorijskih lokacija (bajtova memorije). Kod Orla njih ima 65536. U neke od tih lokacija slobodno možemo upisivati svoje podatke, u neke ne smijemo upisivati nista, a u neke ne možemo upisati nista i kad bi htjeli.

Budući da se sadržaj svake memorijске lokacije sastoji od 8 bitova, znači da najveći broj koji može u nju "stati" u binarnom brojevnom sustavu izgleda ovako:

1 1 1 1 1 1 1 1

a to je decimalni broj 255. Najmanji broj je, naravno NULA (00000000).

Kada želimo promijeniti sadržaj neke memorijске lokacije, odnosno u neku lokaciju upisati broj koji želimo, koristimo naredbu POKE. Njen opći oblik je:

POKE M,B

tj. u memorijsku lokaciju s decimalnom adresom M upisujemo dekadski sadržaj (broj) B. Pri tome je:

0 <= M <= 65535
0 <= B <= 255

Primjer 1:

U memorijsku lokaciju s adresom 2407 zapisat ćemo broj 197:

10 M=2407
20 B=197
30 POKE M,B

ili jednostavnije:

POKE 2407,197 <CR>

Nista se vidljivog nije dogodilo. Promijenili smo samo sadržaj jedne lokacije, a to nije utjecalo niti na nas program, a ni na izgled ekrana.

Na memorijskoj mapi vidi se da postoji dio memorijskog prostora koji se zove VIDEO RAM. To je 8 kilobajta RAM-a u kojem se pamti trenutni sadrzaj ekrana. Kada bi naredbom POKE zapisali neki broj u bilo koju od tih 8192 lokacija, to bi se vidjelo i na ekranu. Probajte:

POKE 25500,255 <CR>

i opazit cete criticu u gornjem desnom kutu ekrana. Zasto bas criticu? Pa zato sto broj 255 binarno izgleda ovako:

1 1 1 1 1 1 1 1

a svaka jedinica predstavlja bijelu tockicu (nula je crna tockica).

Napisite sad:

POKE 27900,170 <CR>

Ugledat cete cetiri tockice na ekranu, zato sto broj 170 binarno izgleda ovako:

1 0 1 0 1 0 1 0

Primjer 2:

Popunit cemo cijelo područje VIDEO RAM-a s brojem 255. Kako ce to izgledati na ekranu, saznajte sami.

```
10 FOR N=24576 TO 32767
20 POKE N,255
30 NEXT N
```

Postoje memorijske lokacije u koje NE MOZEMO (pa i da hocemo) upisati nista. To su lokacije koje pripadaju ROM memoriji racunala.

Postoje i memorijske lokacije kojima je "opasno" mijenjati njihov sadrzaj naredbom POKE. Ne postoji opasnost da pokvarimo racunalo, ali je vrlo vjerojatno da cemo pokvariti pravilno izvodjenje BASIC programa koji se trenutno nalazi u memoriji. Memorijske lokacije s adresama od 0 do 1023 bolje je NE DIRATI (jedino ako znate koja cemu sluzi, a to je opet velika prednost !). Sadrzaj tih lokacija je vrlo bitan za pravilno izvodjenje BASIC programa. Iznad adrese 1024 smjestava se nas BASIC program. Mijenjanje sadrzaja lokacija u kojima je smjesten BASIC program izazvati ce promjene u samom programu, a to ce, najvjerojatnije, rezultirati pojavom greske za vrijeme izvodjenja programa.

Ako bas zelite isprobavati naredbu POKE, bit ce najbolje da mijenjate sadrzaje lokacija koje se nalaze u sredini ili pri vrhu korisnickog RAM-a.

PEEK

====

Ovom naredbom mozete procitati sadrzaj zeljene memorij-ske lokacije. Naredbom:

PEEK (M)

procitat cemo sadrzaj memorij-ske lokacije na adresi M. M je decimalni broj koji predstavlja adresu zeljene memorij-ske lokacije i moze imati vrijednost od 0 do 65535.

Dakle, naredbom PEEK mozemo citati sadrzaj cijelog memo-rijskog prostora mikroracunala Draqo. Pri tome ne postoji nikakva ogranicenja ni opasnosti kao kod naredbe POKE.

Naredba:

PRINT PEEK (16247)

ispisat ce sadrzaj memorij-ske lokacije na adresi 16247.

Primjer 1:

U memorijsku lokaciju 2500 prvo cemo upisati broj 99, a onda cemo pomocu naredbe PEEK procitati njen sadrzaj. Procitanu vrijednost pridruzit cemo varijabli A:

10 POKE 2500,99
20 A= PEEK(2500)
30 PRINT A

Primjer 2:

Pomocu slijedeceg primjera vidjet cete gdje i kako je u memoriji zapamcen program koji je upisan u racunalo:

1 REM POCETAK
10 FOR N=1040 TO 1140
20 M= PEEK(N)
30 IF M<33 THEN 50
40 PRINT N,M,CHR\$(M)
50 NEXT N
60 REM KRAJ

Ovaj program "cita sam sebe" iz memorije. Kad cete ga pokrenuti, na ekranu ce se pojaviti tri kolone znakova. U prvoj su brojevi od 1040 do 1140 koji predstavljaju adrese korisnicke RAM memorije u kojima je smjesten taj program i ciji sadrzaj citamo. U drugoj koloni su brojevi do 255 koji predstavljaju sadrzaj te memorij-ske lokacije (adrese se u prvoj koloni). U trecoj koloni su te vrijednosti pretvorene u karaktere, u tekst. Prepoznaju se samo rjeci "POCETAK" i "KRAJ" i neki brojevi. Ostali podaci predstavljaju kodove upotrebljenih BASIC naredbi, brojeve programske linija, kontrolne znakove...

DEFINIRANJE VLASTITOG SETA ZNAKOVA - naredba CHAR

=====

U dosadašnjem radu na mikroracunalu Orao, sigurno ste se vec naviknuli na oblik i izgled znakova, brojeva i slova kakve nam Orao prikazuje na ekranu. Oni svi zajedno cine SKUP ZNAKOVA mikroracunala Orao (SET ZNAKOVA, KARAKTER SET).

Oblik i izgled svakog karaktera (znaka, broja ili slova) definiran je na odredjenom mjestu u ROM memoriji racunala. Za definiranje izgleda svakog karaktera potroseno je 8 bajtova ROM-a, zato sto je oblik (izgled) svakog karaktera definiran unutar polja od 8*8 tocaka. Slovo "B" izgleda ovako:

U djelu memorije od adrese 57344 (ili E000 hex.) navi-se, nalazi se "KARAKTER GENERATOR" - dio memorijskog prostora ROM-a u kojem je definiran oblik svakog karaktera posebno. Tako se, npr., u lokacijama od 57344 do 57351 nalaze brojevi koji određuju izgled razmaka (praznog mesta), a to je osam nula. Uvjerite se u to:

FOR N=57344 TO 57351: PRINT PEEK(N):NEXT N

Procitajte vrijednosti sljedećih osam lokacija:

FOR N=57352 TO 57359: PRINT PEEK(N):NEXT N

To su brojevi 8,8,8,8,8,0,8,0. Njih cemo "ucrtati" u mrezu od 8*8 tocaka uz NAJOMENU:

- tezine bitova u karakter generatoru su INVERZNE tezinama bitova binarnog koda (2 ima tezinu 2 i slicno).

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

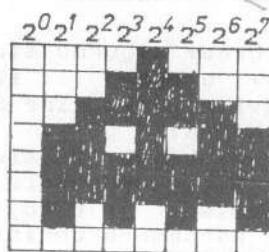
1* 2^3 = 8
1* 2^3 = 8
1* 2^3 = 8
1* 2^3 = 8
1* 2^3 = 8
0
1* 2^3 = 8
0

Tako smo dobili usklicnik.

Mikroracunalo Orao, osim standardnog seta znakova koji su definirani u ROM-u, korisniku pruza mogucnost da definira svoj vlastiti set znakova. Takva mogucnost postaje vrlo zanimljiva kada zelimo, na primjer, umjesto latinice koristiti cirilicu, ili trebamo definirati neke matematicke simbole, ili pak zelimo definirati sasvim nove znakove koji ce nam biti potrebni u nasoj novoj igri (svemirci, duhovi, ...).

Postupak definiranja vlastitog seta znakova vrlo je jednostavan. Potrebno je uzeti list papira i nacrtati praznu mrezu od 8*8 polja:

U takvu mrežu ucrtat ćemo znak ili lik koji zelimo. Neka to bude jedan duh:



$$\begin{array}{l} 2^4 \\ 2^3 + 2^4 + 2^5 = 16 \\ 2^2 + 2^3 + 2^4 + 2^5 + 2^6 = 56 \\ 2^1 + 2^2 + 2^3 + 2^4 + 2^5 + 2^6 + 2^7 = 124 \\ 2^0 + 2^1 + 2^2 + 2^3 + 2^4 + 2^5 + 2^6 + 2^7 = 214 \\ 2^1 + 2^2 + 2^3 + 2^4 + 2^5 + 2^6 + 2^7 = 254 \\ 2^1 + 2^2 + 2^3 + 2^4 + 2^5 + 2^6 + 2^7 = 254 \\ 2^1 + 2^3 + 2^5 + 2^7 = 170 \\ 0 = 0 \end{array}$$

Iz crteza smo izracunali vrijednosti koje sadinjavaju naseg duha. To su brojevi 16, 56, 124, 214, 254, 254, 170, 0.

Rekli smo vec da korisnicki RAM pocinje od lokacije 1024 i prostire se prema gore, sve do VIDEO RAM-a. Lokacije od 0 do 1023 koristi racunalo i nije preporucljivo da napamet mijenjamo njihove sadrzaje. Unutar tog dijela, od 0 do 1023 nalaze se razni podaci o BASIC programu koji se trenutno nalazi u memoriji, a osim toga tu su i mnogi drugi podaci od vrlo velike vaznosti za racunalo. U ovom trenutku nama su narocito interesantne TRI lokacije koje se nalaze u tom dijelu, nultom bloku RAM-a. To su lokacije s adresama: 513, 514 i 515. Njihov trenutni sadrzaj je nula. Provjerite:

FOR N=513 TO 515: PRINT PEEK(N): NEXT N

Sve dok je sadrzaj tih lokacija nula (a nula je tako dugo dok MI ne promijenimo sadrzaj), to je znak racunalu da mora koristiti karakter set iz ROM-a.

Kada zelimo upotrebljavati vlastiti karakter set, sadrzaj ovih lokacija trebamo promijeniti, ali naravno, ne napamet. Karakter set Orla podijeljen je u TRI GRUPE karaktera:

1. specijalni znakovi i brojevi
2. VELIKA slova
3. MALA slova

Svaka grupa sastoji se od 32 karaktera:
1. specijalni znakovi i brojevi:

razmak, !#\$%& '()*+,-./0123456789:@(=)?

2. VELIKA slova:

QABCDEFIGHIJKLMNOPQRSTUVWXYZCCDSZ

3. MALA slova:

Šabcdefgijklmnopqrstuvwxyzccdsz

Lokacija 513 predstavlja SPECIJALNE ZNAKOVE i BROJEVE. Kad je sadrzaj lokacije 513=0, na ekranu se ispisuju znakovi iz ROM-a.

Lokacija 514 predstavlja VELIKA, a lokacija 515 MALA slova. Dakle:

513 - SPECIJALNI ZNAKOVNI I BROJEVI

514 - VELIKA SLOVA

515 - MALA SLOVA

Kada promijenimo sadržaj bilo koje od ove tri lokacije, to je znak racunalu da zelimo koristiti vlastiti set znakova. Ako odlucimo promijeniti mala slova u razne likove, promijenit ćemo samo sadržaj lokacije 515. Tih likova može biti najviše 32, a može ih biti i manje.

Za svaki lik treba nam 8 bajtova. 32 lika će, prema tome, zauzeti $32 \times 8 = 256$ bajtova memorije.

Njegoznija stvar u svemu tome je ta da ne možemo smjestiti podatke za nove znakove bilo kuda u RAM, nego samo NA POCETEK MEMORIJSKE STRANICE. MEMORIJSKA STRANICA je grupa od 256 bajtova, lokacija memorije. NULTA stranica (engl. "zero page") je grupa od prvih 256 bajtova memorije, od adrese 0 do 255.

PRVA stranica zauzima lokacije od 256 do 511.

DRUGA stranica zauzima lokacije od 512 do 767, itd...

Na ove tri stranice NIJE DOZVOLJENO upisivati podatke za vlastiti karakter set.

Na nekoliko slijedećih stranica NIJE PREPORUCLJIVO upisivati vlastiti set znakova zato što se u tom dijelu memorije pamti BASIC program.

Najbolje će biti da podatke za ovaj set znakova upisemo negdje pri kraju (ne na samom kraju) korisnickog RAM-a.

Video RAM zapocinje na 24576/256 - na 96-toj stranici memorije. Najbolje će biti da koristimo, na primjer 90, 91, 92 ... 94, stranicu (samo ako imate Oroao s 32 kilobajta RAM-a).

Vratimo se opet naseom duhu kojeg smo malo prije definirali. Promijenit ćemo skup malih slova, tj. "izbaciti" sve mala slova i definirati samo duha, umjesto znaka (zato jer je taj znak prvi po redu među malim slovima). Podatke za duha smjestit ćemo na 94. stranicu memorije, a racunalu ćemo signalizirati da zelimo promijeniti set malih slova tako da u lokaciju 515 upisemo broj koji predstavlja stranicu memorije na kojoj se nalazi novi set malih slova.

Evo kako će izgledati taj program:

```
10 REM izmjena seta malih slova
15 REM -----
20 N=94*256: REM N=prva lokacija 94. stranice
30 POKE 515,94: REM novi set je na 94. stranici
40 FOR X=0 TO 7: REM citanje 8 podataka novog znaka
50 READ A: REM citaj podatak
60 POKE N+X,A: REM podaci se smjestaju na 94. stranicu
70 NEXT X
80 REM podaci za duha:
90 DATA 16,56,124,214,254,254,170,0
```

Izvršite ovaj program. Pritiskom na <PF1> predjite u mala slova. Pritisnite taster na kojem se nalaze majmunski znak i znak za potenciju, i pojavit će se duh. Ostala mala slova ne postoje jer smo definirali samo jedan znak.

Povratak seta malih slova iz ROM-a postigli bi pomocu naredbe POKE 515,0.

CHAR
=====

Ovu naredbu koristimo pri promjeni skupa znakova umjesto malo prije opisanih POKE instrukcija.
Opci oblik ove naredbe je:

CHAR X,M

X=0 - specijalni znakovi i brojevi
X=1 - velika slova

X=2 - mala slova

M - memorijска stranica na kojoj se nalazi novi set. Ako je M=0, tada se za izabrane znakove postavlja adresa skupa znakova u ROM-u.

Prethodni primjer, u kojem smo definirali duha, mogao bi izgledati i ovako:

```
10 REM izmjena seta malih slova
15 REM -----
20 N=94*256
30 CHAR 2,94
40 FOR X=0 TO 7
50 READ A
60 POKE N+X,A
70 NEXT X
80 REM
90 DATA 16,56,124,214,254,254,170,0
```

Ovako definirane znakove moguce je ispisivati i inverzno, ali se gubi mogucnost editiranja (ispravljanja) teksta pomocu tastera <PF4> - novi znakovi ne mogu se kopirati s <PF4>.

SMOVE
=====

Ovo je vrlo zanimljiva i vrlo "mocna" naredba. Da bi ju mogli razumjeti, potrebno je dobro vladati pojmovima kao sto su bit, bajt, binarni broj, a donekle treba poznavati i memorijsku mapu mikroracunala Orao.

Znamo vec dobro da je jedan karakter definiran pomocu mreze od 8*8 tocaka. Naredba SMOVE sluzi nam za iscrtavanje SLICICE dimenzije 16*16 tocaka.

Opci oblik naredbe SMOVE je:

SMOVE X,MM

X - funkcija izmedju tocaka slicice i tocaka podloge

- X = 0 - ekskluzivno ILI
- X = 1 - iscrtavanje slicice
- X = 2 - logicko "I"
- X = 3 - logicko "ILI"

MM - decimalno izrazena adresa pocetka slicice u memoriji.

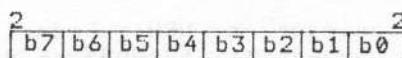
Slicica se pozicionira na ekranu pomocu naredbe MOVE koja određuje poziciju gornjeg lijevog kuta slicice.

Da bi slicicu nacrtali na ekranu, moramo ju prije toga "nacrtati" u memoriji racunala. Buduci da se slicica sastoji od 16*16 tocaka, bit ce nam potrebno 32 bajta memorije (jer je slicica velika kao cetiri normalna karaktera - siroka je kao dva znaka, a visoka isto toliko). Tek kad se podaci za slicicu nalaze u memoriji, mozemo ju pomocu naredbe SMOVE iscrtati na ekranu u zeljenom obliku.

32 bajta od kojih se sastoje slicica, organizirani su na malo neobican nacin. Evo kako to izgleda:

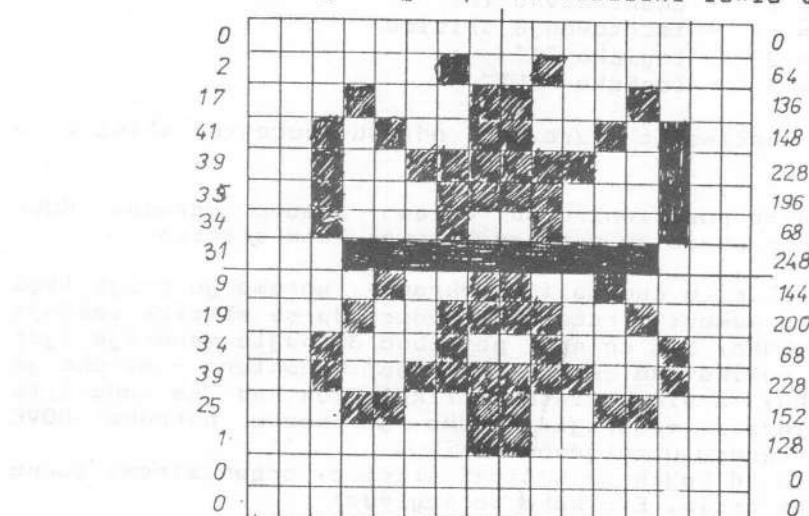
1. bajt	2. bajt
3. bajt	4. bajt
5. bajt	6. bajt
7. bajt	8. bajt
9. bajt	10. bajt
11. bajt	12. bajt
13. bajt	14. bajt
15. bajt	16. bajt
17. bajt	18. bajt
19. bajt	20. bajt
21. bajt	22. bajt
23. bajt	24. bajt
25. bajt	26. bajt
27. bajt	28. bajt
29. bajt	30. bajt
31. bajt	32. bajt

Svaki bajt sastoji se od 8 bita, a vrijednosti za svaki bajt izracunavaju se drugacije nego sto smo to radili kod promjene seta znakova. Tezine pojedinih bitova ovdje izgledaju ovako:



Primjer 1:

Nacrtat ćemo jednog leptira velicine 16*16 točaka:



Izracunate vrijednosti (podatke) potrebno je smjestiti u memoriju. Neka to budu 32 lokacije memorije s pocetkom od adresi 8000. Obratite pažnju na redoslijed spremanja izracunatih vrijednosti u memoriju.

```

5 REM LEPTIR
6 REM =====
10 FOR M=0 TO 31
20 READ A
30 POKE 8000+M,A
40 NEXT M
50 CLS
60 MOVE 100,100
70 SMOVE 1,8000
80 REM podaci za slicicu
90 DATA 0,0,2,64,17,136,41,148
100 DATA 39,228,35,196,34,68,31,248
110 DATA 9,144,19,200,34,68,39,228
120 DATA 25,152,1,128,0,0,0,0

```

ANIMACIJA pomocu naredbe SMOVE

=====

Obrisite prethodni primjer s NEW. Podaci za naseg leptira se i dalje nalaze u memoriji, od 8000 na dalje.

U sljedećem primjeru leptir će "ozivjeti" - kretat će se po ekranu. A kako?

Vrlo jednostavno. Nacrtat ćemo slicicu na jednom mjestu ekrana, nakon toga ćemo ju obrisati, nacrtati na drugom

mjestu ... U ovom primjer neće cak biti ni potrebno brisati slicicu. Leptir će se kretati prema gore. Svako iscrtavanje bit će pomaknuto za jednu točku prema gore. Zadnji donji red nase slicice je PRAZAN (0,0), tako da na ekranu neće ostajati nikakav trag od prethodne slicice!

Evo programa za animaciju leptira:

```
10 CLS  
20 FOR Y=20 TO 230  
30 MOVE 100,Y  
40 SMOVE 1,8000  
50 NEXT Y
```

Promijenite liniju 20 u:

```
20 FOR Y=20 TO 230 STEP 2
```

Leptir je sada dvostruko brzi. I dalje na ekranu ne ostaje nikakav trag od prethodno iscrtane slicice, zato što i predzadnji red nase slicice ima vrijednosti 0,0.

Probajte sad:

```
20 FOR Y=20 TO 230 STEP 3
```

Vjerojatno vam je jasno zbog cega je slika na ekranu ovakva.

Primjer 2:

Sada ćemo se pozabaviti još zgodnjom animacijom naseg leptira. On će letjeti po ekranu i mahati krilima.

Da bi postigli taj efekat, potrebno je napraviti slijedeće:

Nacrtat ćemo tri slicice leptira s razlicitim položajima krila. Na prvoj, krila su najrasirenija, na drugoj su vec malo manje rasirena, a na trecoj su krila skupljena. Tijelo leptira na sve tri slicice MORA ostati potpuno isto. Te tri slicice ćemo iscrtavati na ekranu na slijedeci nacin:

- na odredjenoj poziciji na ekranu iscrtat ćemo prvu slicicu. Na istom mjestu, odmah nakon toga, iscrtat ćemo drugu, pa onda i treću slicicu. Nakon toga ćemo pomaknuti pocetnu poziciju iscrtavanja za jednu točku prema gore (zadnji red svih slicica mora biti prazan). Na toj poziciji opet ćemo iscrtati sve tri slicice, i tako redom, točku po točku, sve do kraja ekrana.

Evo kako izgleda konacan program:

```
5 REM ANIMACIJA LEPTIRA
9 GOSUB 200
10 VDU
20 X=127: Y=20
30 FOR Y=20 TO 200
40 MOVE X,Y
50 SMOVE 1,8192
60 GOSUB 150
70 SMOVE 1,8448

80 GOSUB 150
90 SMOVE 1,8704
100 GOSUB 150
110 NEXT Y
120 END
130 FOR N=1 TO 25: NEXT N
140 RETURN
200 REM LEPTIR 1
210 FOR N=0 TO 31
220 READ A
230 POKE 8192+N,A: NEXT N
240 DATA 0,0,2,64,17,136,41,148,39,228,35,196,34,68,31
250 DATA 248,9,144,19,200,34,68,39,228,25,152,1,128,0,
0,0,0
300 REM LEPTIR 2
310 FOR N=0 TO 31
320 READ A
330 POKE 8448+N,A: NEXT N
340 DATA 0,0,2,64,17,136,25,152,23,232,19,200,18,72,15,
240,9,144
350 DATA 19,200,18,72,23,232,13,176,1,128,0,0,0
400 REM LEPTIR 3
410 FOR N=0 TO 31
420 READ A
430 POKE 8704+N,A: NEXT N
440 DATA 0,0,2,64,9,144,9,144,15,240,11,208,10,80,7,
224,5,160
450 DATA 11,208,10,80,11,208,13,176,1,128,0,0,0,0
500 RETURN
```

2904

Kad bi liniju 30 promijenili u:

```
30 FOR Y=20 TO 200 STEP 2
```

leptir bi se kretao jedanput brze.

Ako bi u liniji 150 smanjili N u FOR-NEXT petlji, leptir bi brze mahao krilima (to je ustvari prazna petlja za kasnjenje).

Primjer 3:

Evo još jednog vrlo zgodnog primjera animacije u kojem koristimo naredbu SMOVE. Radi se o jednom (svemirskom) brodu koji leti zvjezdanim nebom. Njime čak možemo i upravljati preko tastature pomoću sljedećih tastera:

A - lijevo
S - desno
Y - dolje
K - kocnica
L - gas

```
0 REM animacija svem. broda
1 FOR N=0 TO 31
2 READ A: POKE 4096+N,A: NEXT N
3 CLS: GOSUB 5000
4 A=100: N=2
5 FOR Y=10 TO 240 STEP N
10 MOVE A,Y
20 SMOVE 1,4096
25 GOSUB 4000
70 NEXT Y
99 GOTO 5
4000 INKEY A$
4002 IF A$="y" THEN Y=Y-3: IF Y<10 THEN Y=10
4005 IF A$="a" THEN A=A-2: IF A<10 THEN A=10
4010 IF A$="s" THEN A=A+2: IF A>240 THEN A=240
4015 IF A$="k" THEN Y=Y-1: IF Y<1 THEN Y=1
4017 IF A$="l" THEN Y=Y+2
4050 RETURN
5000 FOR N=1 TO 100
5010 Q=RND(7)*250
5020 W=RND(7)*250
5030 PLOT Q,W: NEXT X
5099 RETURN
6000 DATA 0,0,3,128,7,192,15,224,31,240,63,248,3,128,
1,0
6010 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
```

Naredbu SMOVE oblika SMOVE 0,MM / SMOVE 2,MM / SMOVE 3,MM upoznajte sami eksperimentiranjem. Radi se samo o drugacijem iscrtavanju slicice s obzirom na podlogu (0,1,2 i 3 označavaju funkciju između točaka slicice i točaka pozadine, odnosno mjesto na ekranu na kojem će se slicica iscrtati.

Crtanje slicica od 16*16 točaka na papiru je dosta mukotrpan posao, isto kao i kasnije izračunavanje vrijednosti za slicicu i njihovo unosjenje u memoriju. Ovaj posao olaksat će vam sljedeći program koji služi za kreiranje slicica 16*16

tocaka i njihovo smještanje u zeljeni dio memorije. Na ekranu se iscrta mreža od 16*16 polja u koje ucrtavate i brišete točke pritiskom na razmaknicu, sve dok slicica nije gotova. Iz polja u polje prelazite pomoću slijedeci tastera:

Q - gore O - lijevo
A - dolje P - desno
K - kraj
Razmaknica - popunjavanje/brisanje polja

Na pocetku programa odaberite adresu u memoriji gdje će se smjestiti slicica koju cete crtati. Kad se podaci za gotove slicice nalaze spremjeni u memoriji, mozete ih snimiti na kazetu (DMEM), ili "procitati" iz memorije pomoći naredbe PEEK. Evo tog programa koji se zove DIZAJNER SPRAJTOVA:

```
10 VDU: MODE 0:POKE 128,15
20 INPUT "ADRESA (4200-7000)":AD
22 IF AD<4200 OR AD>7000 THEN RUN
27 GOSUB 500
30 X=0: Y=0
40 MOVE X*8+64,Y*8+72
50 SMOVE 0,4096
60 INKEY A$:=SMOVE 0,4096
64 GOSUB 150: SMOVE 0,4096
66 IF LEN(A$)<1 THEN 60
70 IF A$="a" THEN Y=Y-1-(Y=0): GOTO 120
80 IF A$="q" THEN Y=Y+1+(Y=15): GOTO 120
90 IF A$="o" THEN X=X-1-(X=0): GOTO 120
95 IF A$="p" THEN X=X+1+(X=15): GOTO 120
98 IF A$="k" THEN 300
100 IF ASC(A$)=32 THEN SMOVE 0,4096: GOTO 115
110 GOTO 60
115 INKEY A$: IF LEN(A$)<>0 THEN 115
120 SMOVE 0,4096: GOTO 40
150 LNK 707A: MOVE 10,136: SMOVE 1,8064
160 MOVE X*8+64,Y*8+72: RETURN
300 CLS: FOR I=0 TO 31: L=PEEK(8064+I)
310 POKE AD+I,L: NEXT: RUN
500 FOR I=4096 TO 4127: POKE I,0: NEXT
510 FOR I=4098 TO 4110 STEP 2: POKE I,127: NEXT
520 FOR I=0 TO 16: FOR J=0 TO 16
530 PLOT 64+I*8,64+J*8: NEXT J,I
540 MOVE 62,62
550 DRAW 62,194,194,194,194,62,62,62
560 FOR I=7936 TO 7992: READ K
570 POKE I,K: NEXT: RETURN
580 DATA 169,8,133,224,169,104,133,225
590 DATA 162,0,160,0,177,224,24,240,1,56,189,128,31
600 DATA 42,157,128,31,200,192,8,208,238,232,177
620 DATA 224,24,240,1,56,189,128,31,42,157,128,31,200
630 DATA 192,16,208,238,230,225,232,224,32,208,210,96
```

120 *** DRAO PRIRUCNIK ***

POZIVANJE STROJNIH POTPROGRAMA IZ BASIC-a
/naredbe LNK i USR /

BASIC mikroracunala Orao nudi nam mogucnost da osim osnovnih naredbi, pod kontrolom BASIC-a koristimo i strojne potprograme koje smo sami napisali ili koji vec postoje napisani u ROM-u.

Pozivanje strojnog potprograma iz BASIC-a vrsti se pomocu naredbe:

LNK X

X - decimalna adresa pocetka strojnog potprograma

Ako se, na primjer, strojni program nalazi u memoriji racunala od adrese 1000 (heksadecimalno), iz BASIC-a bi ga pozvali naredbom:

30 ...
40 ...
50 LNK 4096
60 ...

4096 je decimalni ekvivalent heksadecimalnog broja 1000. Strojni potprogram mora zavrsavati naredbom RTS, kako bi se, nakon izvodjenja potprograma, kontrola ponovo vratila BASIC interpretelu.

Primjer 1:

Crtanje kruznice strojnim potprogramom.

U ROM-u mikroracunala Orao nalazi se strojni potprogram za crtanje kruznice. Pocetna adresa mu je FF06 (heksadecimalno) ili 65286 (decimalno). Za korektno izvodjenje, navedeni potprogram zahtijeva da u odredjene memoriske lokacije upisemo podatke o koordinatama sredista i polumjer kruznice. Dakle, prije nego sto pozovemo taj potprogram, potrebno je napraviti slijedece:

- u lokaciju 226 upisati X koordinatu sredista
- u lokaciju 227 upisati Y koordinatu sredista
- u lokaciju 248 upisati polumjer kruznice

```
10 REM
20 REM POZIVANJE STROJNIH POTPROGRAMA (LNK)
30 CLS
40 INPUT "X KOORDINATA SREDISTA";X
50 INPUT "Y KOORDINATA SREDISTA";Y
60 INPUT "POLUMJER KRUZNICE";R
70 POKE 226,X: POKE 227,Y: POKE 248,R
80 LNK 65286
90 END
```

Pozivanje strojnih potprograma moze se vrziti i pomocu funkcije:

X=USR(X)

Prije nego sto pozovemo potprogram, potrebno je pocetnu adresu staviti u lokacije 04 i 05.

Prethodni primjer u ovom slucaju izgledao bi ovako:
- startna adresa potprograma heksadecimalno je FF06:

FF 06
visi dio nizi dio

- visi dio (FF) decimalno = 255
- nizi dio (06) decimalno = 6

U lokaciju 04 upisat cemo nizi dio adrese (6), a u lokaciju 05 visi dio (255). Isti program bi, koristeci funkciju USR umjesto naredbe LNK, izgledao ovako:

```
10 REM
20 REM POZIVANJE STROJNIH POTPROGRAMA (USR)
30 CLS
40 INPUT "X KOORDINATA SREDISTA";X
50 INPUT "Y KOORDINATA SREDISTA";Y
60 INPUT "POLUMJER KRUZNICE";R
70 POKE 226,X: POKE 227,Y: POKE 248,R
80 REM INICIJALIZACIJA VEKTORA STARTNE ADRESE
90 POKE 4,6: POKE 5,255
95 X=USR(X)
99 END
```

SPREMANJE SADRZAJA DIJELA MEMORIJE (STROJNOG PROGRAMA)
NA KAZETU - naredba DMEM

=====

Za spremanje (snimanje) sadrzaja dijela memorije ili strojnog programa na kazetu koristi se naredba:

DMEM "IME PROGRAMA",S,D

S - pocetna (startna) adresa - DECIMALNO

D - duzina programa (broj bajtova) - DECIMALNO

Primjer:

Snimit cemo trenutni sadrzaj ekranu:

DMEM "EKRAN",24576,8192 <CR>

IME PROGRAMA ne smije biti duze od deset znakova.

UCITAVANJE STROJNOG PROGRAMA (DIJELA MEMORIJE)
S KAZETE - naredba LMEM

Malo prije snimljen sadržaj ekrana ucitat cemo u racunalo pomocu naredbe:

LMEM "EKRAN",24576 <CR>

ili jednostavno LMEM "" <CR>

Ako se iza imena datoteke (programa) NE NAVEDE adresa, tada se podaci ucitavaju na mjesto odakle su snimljeni.

Ako ih zelimo ucitati na drugo mjesto u memoriji, tada se iza imena navodi zarez i adresa nove pozicije u memoriji (DECIMALNO).

Na primjer:

LMEM "EKRAN" <CR> - ide na ekran

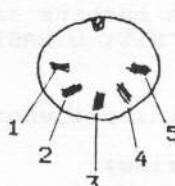
LMEM "EKRAN",8192 <CR> - ce se ucitati na novu poziciju u memoriji, od lokacije 8192 na dalje

Drugi parametar (duzina) se kod ucitavanja NE NAVODI, jer je duzina ucitanih podataka uvijek jednaka duzini snimljenih podataka.

2-20. POVEZIVANJE SA ŠTAMPACEM

Mikroracunalo Orao moze se povezati sa stampacem preko prikljucka za stampac koji se nalazi na straznjoj strani racunala.

Spajanje racunala i stampaca izvodi se na slijedeci nacin:

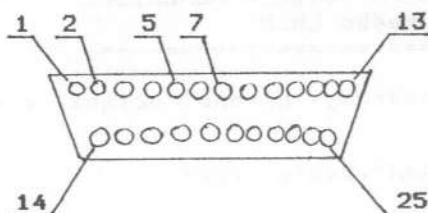


- ✓ 1 - uskladjivanje (handshake)
- ✓ 2 - ne spaja se
- ✓ 3 - masa
- ✓ 4 - ne spaja se
- ✓ 5 - vruci kraj printer-a (linija podataka)

Napomena:

Za korisnike koji imaju ugradjen 25-polni konektor, na slijedeći slici je prikazano kako se mora povezati stampac u tom slučaju:

- PROGRAM
RATE 2 RUN na ispis
rezultata



- AKO ŽELIM ISPIS PROGRAMA
NAPISATI
RATE 2 (CR)
LIST PF2 ((R))
- BIJEZNIKI NEVI PREDSTAVLJAJU
PRVO PRITISNEMO PF2

- 1 - masa
2 - izlaz na stampac
5 - uskladjivanje (handshake)
7 - uzemljenje (frame ground)

NACIN KORISTENJA STAMPACA

Mikroracunalo DRAO je konstruirano tako da moze raditi sa stampacem SERIJSKOG PRIJENOSA podataka (rijec je o serijskom standardu tzv. RS232 ili V24). Zbog toga je vazno poznavati na koju je brzinu prijenosa podesen prikljuceni stampac. Uobičajene brzine mogu biti:

300 <bauda>
600 <bauda>
1200 <bauda>
2400 <bauda>
4800 <bauda>
9600 <bauda> itd.

Mikroracunalo DRAO moze raditi s brzinama od 300 do 2400 baua.

Brzina prijenosa podataka odabire se na sljedeci nacin:
- kontrola racunala mora biti u BASIC-u
- naredba:

RATE 0 odgovara brzini prijenosa od 300 baua.

Slicno je i za ostale brzine:

RATE 0 - 300 baua
RATE 1 - 600 baua
RATE 2 - 1200 baua
RATE 3 - 2400 baua

Brzinu prijenosa potrebno je ponovo upisati nakon svakog ukljucivanja ili resetiranja racunala.

Nakon sto je podesena brzina prijenosa podataka, slanje podataka na stampac ukljuce se pritiskom na <PF2> ili naredbom:

PTR 1

Slanje podataka na stampac se isključuje ponovnim pritiskom na <PF2> ili naredbom:

PTR 0

Primjer:

Na kazeti nam se nalazi snimljen program kojeg zelimo "izlistati" na stampacu. Postupak je slijedeci:

- ukljucimo racunalo i stampac (moraju biti pravilno povezani)
- ucitamo program s kazete
- upisemo brzinu prijenosa, na primjer:
RATE 2 <CR>
- otkucamo:
LIST <PF2> <CR>

i stampac pocinje s ispisivanjem listinga programa. Kad je to gotovo, pritiskom na <PF2> iskljucimo slanje podataka na stampac (prekinemo vezu izmedju rzecunala i stampaca).

Ukoliko ste, slucajno, vlasnik PEL-ovog matricnog stampaca P-80, prepisite slijedeci program. Vidjet cete koliko razlicitih vrsti znakova mozete dobiti!

```
2 PTR1  
3 W=0  
4 IF W=1 THEN PRINT "ZGUSNUTI - ";  
5 IF W=1 THEN Q=3  
6 PRINT "NORMALNI ZNAKOVI"  
10 PRINT CHR$(14)  
19 IF W=1 THEN PRINT "ZGUSNUTI ";  
20 PRINT "ZNAKOVI DVOSTRUKE SIRINE"  
30 PRINT CHR$(20)  
40 PRINT CHR$(27);CHR$(91)  
49 IF W=1 THEN PRINT "ZGUSNUTI ";  
50 PRINT "ZNAKOVI DVOSTRUKE VISINE"  
60 PRINT CHR$(27);CHR$(51)  
  
70 PRINT CHR$(27);CHR$(92)  
79 IF W=1 THEN PRINT "ZGUSNUTI ";  
80 PRINT "ZNAKOVI DVOSTRUKE SIRINE I VISINE"  
90 PRINT CHR$(27);CHR$(51)  
100 PRINT CHR$(27);CHR$(69)  
109 IF W=1 THEN PRINT "ZGUSNUTO ";  
110 PRINT "PISANJE DVOSTRUJKIM INTENZITETOM"  
120 PRINT CHR$(27);CHR$(70)  
130 PRINT CHR$(27);CHR$(71)  
139 IF W=1 THEN PRINT "ZGUSNUTI ";  
140 PRINT "DVOSTRUKI - FAZNO POMAKNUTI ZNAKOVI"  
150 PRINT CHR$(27);CHR$(72)  
200 PRINT CHR$(15): PRINT "ZGUSNUTO PISANJE"  
300 IF Q=3 THEN PTR0: END  
310 W=1: GOTO 4
```

10 PTR 0 nije potrebno uloziti PTR 2
20 PRINT CHR\$(24);CHR\$(92) ispis smehova dvostrukih sirine + visine
30 PRINT "MERIDIO"
40 PRINT CHR\$(27);CHR\$(51) ponisti- nje pisanya naredenih znakova
50 PTR 0 RATE 2
RATE 2

Napisati ovaj program i naciscati aticnu 88-90

125 *** ORAO PRIRUCNIK ***

120 TELIM VOBIT PROGRAM NAPISANA LIST <PF2> CR
RATE 2 <CR> LIST PF2 <CR> Z PRIMER PRIMER
UVJETNO PROVREDITISNIH PF2 HKO SE IDE U RADNIK SAMO LIST I DRUGI

Iz ovog programa mozete vidjeti na koji nacin se uključuje, odnosno isključuje pojedini tip znakova.

Primjer:

```
Ako zelite ispis znakovima duple sirine, napisite:  
10 PRINT CHR$(14)  
20 PRINT "TEKST ..."  
30 PRINT CHR$(20)  
40 END
```

U liniji 10 ukljucili smo ispis slovima duple sirine, a u liniji 30 smo ga iskljucili.

2.21. RAD S DATOTEKAMA NA KAZETI

BASIC mikroracunala Draq omogucuje nam da vrlo jednostavno spremamo BASIC podatke na kazetu. Ti se podaci kasnije mogu ucitati i koristiti u daljem radu. Ova se mogucnost koristi za kreiranje vlastitih BAZA PODATAKA koje se pohranjuju na kazete. Tako se moze napraviti adresar, telefonski imenik, datoteka u kojoj cemo biljeziti ocjene ucenika ...

SPREMANJE BASIC PODATAKA NA KAZETU

=====

Kod spremanja BASIC podataka na kazetu, mora se postupiti na slijedeci nacin:

```
10 DIM A(100)      - kreiramo jednodimenzionalno  
20 FOR X=1 TO 100   polje podataka A(100) u koje  
30 A(X)=X         smo smjestili brojeve  
40 NEXT X          od 1 do 100  
50 OPENW "IME"     - priprema kazete za prihvatanje podataka  
60 FOR N=1 TO 100   - priprema petlje  
70 WRITE A(N)       - zapis podatka iz polja A na kazetu  
80 NEXT N          - nastavak  
90 CLOSEW          - odjava rada s kazetom (OBAVEZNO)
```

Uz WRITE naredbu (linija 70) moze biti naveden proizvoljan broj varijabli (razlicitih tipova) koje moraju biti odvojene zarezom.

IME DATOTEKE koju cemo kreirati na kazeti moze biti bilo koji BASIC string, na primjer:

```
5 INPUT "UNESI IME DATOTEKE";A$  
10 OPENW A$  
20 ...  
30 ...
```

Ovdje smo upoznali tri nove naredbe:

OPENW, WRITE i CLOSEW

OPENW, WRITE i CLOSEW

Uz OPENW obavezno moramo navesti IME DATOTEKE koju cemo kreirati na kazeti. OPENW sluzi za pripremu kazete za prihvat podataka.

Naredba WRITE zapisuje podatak na kazetu. Uz WRITE mozemo navesti i vise varijabli koje moraju biti odvojene zarezom.

Za odjavu rada s kazetom sluzi nam naredba CLOSEW. Ona mora obavezno stajati na kraju procesa zapisa podataka na kazetu.

CITANJE BASIC PODATAKA S KAZETE

=====

Procedura za citanje podataka s kazete je vrlo slicna proceduri za zapis podataka na kazetu. U prethodnom primjeru smo na kazetu zapisali sadrzaj polja A, odnosno brojeve od 1 do 100. S kazete cemo ih ucitati na slijedeci nacin:

10 DIM A(100) - otvaranje polja za upis
20 OPENG "IME" - priprema za citanje podataka
30 FOR N=1 TO 100 - petlja z citanje
40 INPUT A(N) - citanje podatka s kazete
50 NEXT N - i pridruzivanje varijablama
60 CLOSEG - zavrsetak rada

Ovdje susrecemo dvije nove naredbe: OPENG i CLOSEG. Uz OPENG obavezno stoji IME datoteke koju cemo ucitati s kazete. To ime takodjer moze biti bilo koji BASIC string.

Naredba CLOSEG upotrebljava se kao zavrsna naredba procesa citanja podataka s kazete. Uz nju se ne navode nikakvi parametri.

Izmedju OPENG i CLOSEG obavezna je naredba INPUT koju koristimo za citanje podatka s kazete i pridruzivanje istog odredjenoj varijabli.

Najvaznije je da kod ucitavanja podataka u INPUT naredbi stoji ISTI BROJ i ISTI TIPOVI varijabli koji su bili navedeni kod Upisivanja tih podataka na kazetu.

Ucitavanje se moze i prije kraja prekinuti s CLOSEG (ne moraju se ucitati svi podaci). Ali NE SMIJE se ucitavati vise podataka nego sto je zapisano na kazeti (nista strasno se nece dogoditi).

PRIMJER: TELEFONSKI IMENIK ZA 50 OSOBA

Slijedi vrlo zgodan primjer kreiranja (licnih) baza podataka na kazeti. U ovom slucaju to TELEFONSKI IMENIK ZA 50 OSOBA.

UPUTE ZA KORISTENJE PROGRAMA:

Pazljivo prepisite ovaj program i snimite ga na kazetu. Kad ga pokrenete (RUN), pred vama ce se pojaviti PRVI izbor:

1 - UCITAVANJE STAROG TELEFONSKOG IMENIKA S KAZETE

2 - KREIRANJE NOVOG TELEFONSKOG

IMENIKA BEZ UCITAVANJA STAROG S KAZETE

odaberite (1/2)

Za prvi puta pritisnite taster 2. Pred vama je DRUGI izbor:

- 1 - UPIS PODATAKA
- 2 - PREGLED IMENIKA
- 3 - SNIMANJE IMENIKA NA KAZETU
- 9 - KRAJ

odaberite (1/2/3/9)

Pritisnite taster 1. Pojavilo se pitanje:
OD REDNOG BROJA?

U ovom imeniku ima mesta za 50 osoba od kojih svaka ima svoj redni broj. Upis mozemo poceti od bilo kojeg rednog broja zelimo (1 do 50). Unesite "1". Sve je spremno za upis podataka za prvu osobu. Upisite zeljeno prezime i ime (upis NE SMIJE "izaci van" iz bijele maske). Upisite, dakle, prezime i ime i pritisnite <CR>. Sada upisite i broj telefona. Kad pritisnete <CR>, na dnu ekrana ce se pojaviti pitanje:

"nastavak (D/N)?"

Ako zelite dalje upisivati, pritisnite "D". Ako zelite zavrsiti s upisom, tada pritisnite "N".

"N" vas vraca u prethodni izbor - ponovo mozete izabrati izmedju upisa, pregleda, snimanja i kraja.

Pritiskom na "2" mozete pregledati sve ono sto ste upisali dosad. Probajte. Na kraju pregleda opet se vracamo u prethodni izbor. Ako ste slucajno pogrijesili kod upisa nekog podatka, gresku mozete vrlo lako ispraviti. Pritisnite "1" (upis) i redni broj osobe ciji podatak zelite promijeniti. Sada se unutar bijele maske nalazi podatak koji smo prethodno upisali. Ako ga zelite promijeniti, upisite novi podatak i stari ce biti izbrisani. Ako ga ne zelite promijeniti, pritisnite samo <CR> - stari podatak ce ostati sacuvan.

Kad ste unesli sve one podatke koje ste zeljeli, snimite sadrzaj telefonskog imenika na kazetu. Tako cete ga sutra moci ponovo ucitati, pregledavati, mijenjati... Ako se nalazis u DRUGOM izboru, odaberite opciju "3":

3 - SNIMANJE IMENIKA NA KAZETU

Sada upisite ime pod kojim ce se PODACI snimiti na kazetu (snimit ce se samo sadrzaj tel. imenika, a ne BASIC program). Pripremite praznu kazetu, ukljucite kazetofon na snimanje i pritisnite <CR>. Podaci su sada na sigurnom, snimljeni su na kazeti.

Sada slobodno mozete ugasiti racunalo i zaboraviti sve

neki novi telefonski broj, ili vam zatreba neki stari,
postupite na slijedeci nacin:

1. ucitajte BASIC program (tel. imenik)
2. pokrenite program (RUN)
3. pritisnite taster "1" - ucitajte s kazete samo
sadrzaj telefonskog imenika (podatke koje smo snimili)
4. kad su podaci ucitani, pred vama ce se pojaviti drugi
izbor - izaberite opciju "2" (pregled) i uvjerit cete se da
su podaci za sve osobe tu! Mijenjajte sadrzaj imenika po
zelji, ali na kraju nemojte zaboraviti ono najvaznije -
SNIMITI podatke na kazetu.

```
1 REM =====
2 REM tel. imenik za 50 osoba
3 REM =====
7 REM
50 DIM TI$(50)
55 DIM IP$(50): DIM TB$(50)

60 VDU
70 PRINT "1 - UCITAVANJE STAROG TELEFON-"
72 PRINT " SKOG IMENIKA S KAZETE"
75 PRINT
80 PRINT "2 - KREIRANJE NOVOG TELEFONSKOG"
82 PRINT " IMENIKA BEZ UCITAVANJA"
83 PRINT " STAROG S KAZETE"
84 PRINT: PRINT " odaberi (1/2)"
85 INKEY A$: IF A$="" THEN 85
87 SOUND 90,90
90 IF A$="1" THEN 100
95 IF A$="2" THEN 200
99 GOTO 85
100 REM stari podaci
110 CLS: PRINT "UCITAVANJE PODATAKA S KAZETE":PRINT:
PRINT
120 INPUT "IME TE. IMENIKA":A$
130 OPENG A$
140 FOR N=1 TO 50
150 INPUT TI$(N)
160 NEXT N
170 CLOSEG
180 GOTO 200
199 REM
200 REM *** GLAVNI IZBOR ***
210 VDU
220 PRINT "1 - UPIS PODATAKA": PRINT
240 PRINT "2 - PREGLED IMENIKA": PRINT
245 PRINT "3 - SNIMANJE IMENIKA NA KAZETU": PRINT
260 PRINT "9 - KRAJ": PRINT
270 PRINT " odaberi (1/2/3/9)"
280 INKEY A$: IF A$="" THEN 280
282 SOUND 90,90
285 IF A$="1" THEN 300
287 IF A$="2" THEN 500
289 IF A$="3" THEN 400
```

```

293 IF A$="9" THEN 900
299 GOTO 280
300 REM *** UPIS ***
310 GOSUB 1000
320 VDU
325 INPUT "OD REDNOG BROJA";RB
330 IF RB>50 OR RB<1 THEN 320
332 CLS: PRINT "REDNI BROJ: ";RB
333 CUR 13,5: INV1: PRINT "
REM 17 praznih mjesa
334 CUR 15,5: PRINT IP$(RB): INV0
335 CUR 0,5: PRINT "Prez. i ime"
336 CUR 13,5: INV1: INPUT IP$: INV0
337 IF IP$="" THEN IP$=IP$(RB)
340 IF LEN(IP$)>15 THEN 335
345 IF LEN(IP$)<15 THEN IP$=IP$+" "
:GOTO 345
350 CUR 13,7: INV1: PRINT "
REM 12 praznih mjesa
351 CUR 15,7: PRINT TB$(RB): INV0
352 CUR 0,7: PRINT "Telef. broj"
353 CUR 13,7: INV1: INPUT TB$: INV0
354 IF TB$="" THEN TB$=TB$(RB)
355 IF LEN(TB$)>10 THEN 350
360 IF LEN(TB$)<10 THEN TB$=" "+TB$: GOTO 360
370 TI$(RB)=IP$+TB$
375 GOSUB 800
380 IF A$="d" THEN 390
385 IF A$="n" THEN 200
387 GOTO 375
390 RB=RB+1
395 IF RB>50 THEN 200

399 GOTO 332
400 REM *** snimanje ***
410 VDU: PRINT "SNIMANJE IMENIKA NA KAZETU:":
PRINT: PRINT
420 INPUT "IME TEL. IMENIKA";B$
430 OPENW B$
440 FOR N=1 TO 50
450 WRITE TI$(N)
460 NEXT N
470 CLOSEW
499 GOTO 200
500 REM *** pregled ***
510 A$="": VDU
512 CUR 1,2: PRINT "RB"
513 CUR 3,2: PRINT " Prezime i ime"
514 CUR 21,2: PRINT "Tel. broj"
515 MOVE 0,230: DRAW 255,230
517 VDU 0,31,4,31: CLS
518 CUR 0,5
520 FOR N=1 TO 50
525 IF N<10 THEN PRINT " ";N;TI$(N): GOTO 535
530 PRINT N;TI$(N)
535 IF N=20 OR N=40 THEN GOSUB 800

```

130 *** DRAO PRIRUCNIK ***

```

540 IF (N=20 OR N=40) AND A$<>"n" THEN 550
543 IF (N=20 OR N=40) AND A$="n" THEN 200
544 MOVE 0,50: DRAW 255,50
545 NEXT N
547 GOSUB 850
549 GOTO 200
550 CLS: CUR 0,5: NEXT N
800 CUR 7,30: PRINT "nastavak (D/N) ?"
805 INKEY A$: IF A$="" THEN 805
810 RETURN
850 CUR 0,30: PRINT " pritisni bilo koju tipku"
860 INKEY A$: IF A$="" THEN 860
870 RETURN
900 REM *** kraj ***
999 VDU: PRINT "KRAJ": END
1000 FOR X=1 TO 50
1010 IP$(X)=LEFT$(TI$(X),15)
1020 TB$(X)=MID$(TI$(X),16,10)
1030 NEXT X
1099 RETURN

```

2.22. DODATNE NAPOMENE

Kod rada s kazetom razlikujemo tri tipa datoteka:

B - Basic PROGRAM	- SAVE
D - Basic PODACI	- WRITE
O - OBJECT (memorija)	- DMEM

Datoteka koja je kreirana sa SAVE moze se ucitati jedino LOAD naredbom.

Datoteka kreirana s DMEM moze se ucitati jedino s naredbom LMEM.

Procedura za ucitavanje Basic PODATAKA (tj. datoteke kreirane s WRITE naredbom), opisana je u prethodnom poglavlju (2.21.).

Iz prilozenog se vidi da isto ime moze postojati tri puta, a da ne dodje do zabune.

Kako ne postoji posebna naredba za listanje imena datoteka na kazeti, to se moze posticiti na sljedeci nacin:

LOAD "XYZQW" - ime koje sigurno NE POSTOJI na traci

Kod datoteka kreiranih s DMEM postoji i parametar adrese koji je dat heksadecimalno. On predstavlja adresu od koje je datoteka snimljena na kazetu.

2.22.1. PRIMJER KORISTENJA STROJNIH POTPROGRAMA IZ BASIC-a

Primjer: CRTANJE PROSTORNE FUNKCIJE
 $z=\exp(\sin x + \cos y)$
NA BIJELOJ POZADINI

Glavni program napisan je u BASIC-u. Iz Basic-a se poziva samo jedan potprogram koji je napisan u strojnem jeziku (nije potprogram iz ROM-a). Taj potprogram se koristi za inverziju ekrana (boji ga u bijelo), a nalazi se u DATA linijama BASIC programa. Da bi ga mogli pozvati, vrijednosti iz DATA linija potrebno je preseliti u memoriju, na točno određenu adresu koja predstavlja pocetnu adresu potprograma. U ovom slučaju to je adresa 8000 (1F40 heksadecimalno), tako da se potprogram poziva s LNK 8000.

```
10 REM ****
15 REM *
20 REM * GРЕБЕНИ *
25 REM *
30 REM ****
35 REM
40 REM
50 REM inverzija ekrana
55 CLS: GOSUB 60
57 GOTO 75
60 FOR X=8000 TO 8023
65 READ C: POKE X,C: NEXT X
70 RETURN
75 LNK 8000
80 REM glavno tijelo programa
85 CZ=60
90 PI=3.141593
95 K=2*PI/16
100 MODE 1
105 FOR X=-15 TO 15 STEP .5
110 FOR Y=-10 TO 10 STEP .3125
115 GOSUB 300
120 IF X1>255 THEN X1=255
125 MOVE Y1,X1
130 Y=Y+.5
135 GOSUB 300
140 IF X1>255 THEN X1=255
145 DRAW Y1,X1
150 NEXT:NEXT
155 END
300 REM recunanje točaka
305 L=SIN(K*X)
310 M=COS(K*Y)
315 U=EXP(L+M)
320 V=1+X/CZ
325 XP=(U*50+50)/V
```

```
330 X1=XP*0.5
335 YP=Y+15/V
340 Y1=YP*8
345 RETURN
400 REM
405 REM podaci za strojni
410 REM program za inverziju
415 REM ekrana
420 DATA 169,96,162,127,160,0
425 DATA 133,225,132,224,169,255
430 DATA 145,224,200,208,251,230
435 DATA 225,228,225,176,245,96
```

Kod pisanja ovakvih programa, posebno se napisce BASIC, a posebno strojni program (u miniassembleru). Nakon toga se strojni (pot)program mora "ugraditi" u BASIC program. To se radi tako da se strojni program procita iz memorije pomocu naredbe PEEK i jednostavno prepise u DATA linije.

U nasem primjeru (GREBENI) najprije smo napisali BASIC program, ali samo do linije 400. Nakon toga smo napisali strojni program za inverziju ekrana:

05/05/11
EXIT
*A1F40
1F40 LDA #60
1F42 LDX #7F
1F44 LDY #00
1F46 STA E1
1F48 STY E0
1F4A LDA #FF
1F4C STA (E0),Y
1F4E INY
1F4F BNE 1F4C
1F51 INC E1
1F53 CPX E1
1F55 BCS 1F4C
1F57 RTS
1F58 .
*BW

Sada smo taj program procitali u obliku sadrzaja memorije (znamo da se program nalazi na 1F40=8000 decimalno):

```
FOR N=8000 TO 8023:A=PEEK(N):PRINT A,:NEXT N <CR>
```

I na kraju smo te brojeve smjestili u DATA linije BASIC programa, cime smo strojni program za inverziju ekrana "ugradili" u glavni BASIC program.

2.22.2. OCITAVANJE PADDLE-a

=====

Ocitavanje PADDLE-a se vrši pomoću naredbe:

A=PDL

A poprima vrijednost 255 ako paddle nije priključen.
Ako je paddle priključen, varijabla A poprima odgovarajuću vrijednost iz intervala 0-255.

2.22.3. OGRANICAVANJE MEMORIJE

=====

Što znači pitanje

MEM SIZE ?

koje se pojavi svaki put kad "ulazimo" u BASIC pomoću monitorske naredbe "BC"?

Ovime nam se nudi mogućnost da, ukoliko imamo potrebu, ogranicimo kolicinu slobodne memorije koja će nam biti na raspolaganju za pisanje programa u BASIC-u.

Potreba za ogranicavanjem memorije javlja se ako, uz program u BASIC-u, pisemo i potprograme u strojnem jeziku. U slučaju da, zbog duzine BASIC programa, postoji opasnost da nam on "prekrije" strojne potprograme, ograniciti cemo kolicinu memorije koju BASIC interpreter smije uzeti za memoriranje BASIC programa.

BASIC program pamti se u dijelu memorije počevši od lokacije 0412 (heksadecimalno) ili 1042 (decimalno). Prema tome, ako na pitanje

MEM SIZE ?

unesemo

1043 <CR>

(manje nije dozvoljeno), računalo će nam javiti:

1 BYTES FREE (1 bajt memorije sloboden za BASIC)

Broj koji unosimo (DECIMALNO) predstavlja maksimalnu adresu koja određuje vrh korisničke memorije (RAMTOP) koju BASIC interpreter može uzeti za memoriranje BASIC programa.

Sadržaj RAMTOP-a se nalazi zapisan u lokacijama 127 i 128 (decimalno).

Primjer:

Ako na pitanje:

MEM SIZE ?

odgovorimo samo sa "MEM SIZE ?" odgovorom na ovaj pitanje.

8191 <CR>

racunalo ce nam javiti:

7149 BYTES FREE

sto znači 7149 slobodnih memorijskih lokacija za BASIC program. BASIC program se u memoriji pamti od lokacije 1042 na dalje, ali sada najdalje do lokacije 8191. Memorijski prostor od adrese 8192 do 24575 sada se iskoristiti u druge svrhe (za strojne potprograme, slike, tabele podataka ...).

Pomoći funkcije

PRINT FREE (X)

možemo ispitati trenutno raspolozivi iznos memorije, odnosno uvjeriti se da BASIC-u nije na raspolaganju cijelokupna RAM memorija, kao inace.

Ako na pitanje "MEM SIZE ?" odgovorimo samo sa "<CR>", BASIC-u će biti na raspolaganju cijela RAM memorija.

2.22.4. RAD S DISKETNOM JEDINICOM

=====

Ukoliko ste vlasnik disketne jedinice "PEL-EXT-02" za mikroracunalo Orao (proizvodnje PEL Varazdin), tada imate i prirucnik "ORAO - DOS" u kojem su objasnjene sve komande DOS operativnog sistema za Orao.

Medjutim, u ROM-u racunala se nalaze i dve naredbe za rad s DOS-om (DOS = Disk Operativni Sistem). To su naredbe DOSI i DOS.

Pomoći naredbe

DOSI

ucitava se DOS s diskete u racunalo, a kontrola racunala iz BASIC-a prelazi u DOS.

Pomoći naredbe

DOS

prelazimo iz BASIC-a u DOS (prethodno je DOS trebao biti ucitan s diskete).

Disketna jedinica PEL-EXT-02 ima ugradjenih sest praznih slotova za razne kartice za prosirenja. Jedna od njih mogla bi biti i 80-KOLONSKA KARTICA. Pomoći nje se na ekranu dobija prikaz teksta u 80 kolona i 25 redova. Po zelji se moze

koristiti i osnovni mod od 32 kolone i 32 reda. Prelazak iz osnovnog moda u mod 80*25 vrši se pomocu naredbe:

CRT 1

Povratak u osnovni mod 32*32 vrši se pomocu naredbe:

CRT 0

Ako na vase racunalo nije prikljucena 80-kolonska kartica, naredba

CRT 1

rezultirati će slijedećom porukom od strane racunala:

CRT NOT IMPLEMENTED

Ukoliko je u naredbi CRT NOT IMPLEMENTED navedeno neko drugi mod, tada će seCRT 1 naredba ignorirati, a rezultat će biti da vasa novogradna PC kartica neće biti primljena, te će se u novogradnoj PC kartici pojaviti naredba CRT NOT IMPLEMENTED.

REZULTAT CRT NOT IMPLEMENTED JE NEVJEZAN
Za rezultat naredbe CRT NOT IMPLEMENTED je potrebno da se u naredbi CRT NOT IMPLEMENTED navede neko drugi mod - npr. CRT 1. Ukoliko je u naredbi CRT NOT IMPLEMENTED naveden neko drugi mod, tada će se CRT NOT IMPLEMENTED ignorirati, a rezultat će biti da se u novogradnoj PC kartici pojaviti naredba CRT NOT IMPLEMENTED.

Ukoliko je u naredbi CRT NOT IMPLEMENTED naveden mod CRT 1, tada će se CRT 1 ignorirati, a rezultat će biti da se u novogradnoj PC kartici pojaviti naredba CRT NOT IMPLEMENTED.

Ukoliko je u naredbi CRT NOT IMPLEMENTED naveden mod CRT 0, tada će se CRT 0 ignorirati, a rezultat će biti da se u novogradnoj PC kartici pojaviti naredba CRT NOT IMPLEMENTED.

Ukoliko je u naredbi CRT NOT IMPLEMENTED naveden mod CRT 2, tada će se CRT 2 ignorirati, a rezultat će biti da se u novogradnoj PC kartici pojaviti naredba CRT NOT IMPLEMENTED.

Ukoliko je u naredbi CRT NOT IMPLEMENTED naveden mod CRT 3, tada će se CRT 3 ignorirati, a rezultat će biti da se u novogradnoj PC kartici pojaviti naredba CRT NOT IMPLEMENTED.

Ukoliko je u naredbi CRT NOT IMPLEMENTED naveden mod CRT 4, tada će se CRT 4 ignorirati, a rezultat će biti da se u novogradnoj PC kartici pojaviti naredba CRT NOT IMPLEMENTED.

Ukoliko je u naredbi CRT NOT IMPLEMENTED naveden mod CRT 5, tada će se CRT 5 ignorirati, a rezultat će biti da se u novogradnoj PC kartici pojaviti naredba CRT NOT IMPLEMENTED.

Ukoliko je u naredbi CRT NOT IMPLEMENTED naveden mod CRT 6, tada će se CRT 6 ignorirati, a rezultat će biti da se u novogradnoj PC kartici pojaviti naredba CRT NOT IMPLEMENTED.

Ukoliko je u naredbi CRT NOT IMPLEMENTED naveden mod CRT 7, tada će se CRT 7 ignorirati, a rezultat će biti da se u novogradnoj PC kartici pojaviti naredba CRT NOT IMPLEMENTED.

Ukoliko je u naredbi CRT NOT IMPLEMENTED naveden mod CRT 8, tada će se CRT 8 ignorirati, a rezultat će biti da se u novogradnoj PC kartici pojaviti naredba CRT NOT IMPLEMENTED.

Ukoliko je u naredbi CRT NOT IMPLEMENTED naveden mod CRT 9, tada će se CRT 9 ignorirati, a rezultat će biti da se u novogradnoj PC kartici pojaviti naredba CRT NOT IMPLEMENTED.

Ukoliko je u naredbi CRT NOT IMPLEMENTED naveden mod CRT A, tada će se CRT A ignorirati, a rezultat će biti da se u novogradnoj PC kartici pojaviti naredba CRT NOT IMPLEMENTED.

Ukoliko je u naredbi CRT NOT IMPLEMENTED naveden mod CRT B, tada će se CRT B ignorirati, a rezultat će biti da se u novogradnoj PC kartici pojaviti naredba CRT NOT IMPLEMENTED.

Ukoliko je u naredbi CRT NOT IMPLEMENTED naveden mod CRT C, tada će se CRT C ignorirati, a rezultat će biti da se u novogradnoj PC kartici pojaviti naredba CRT NOT IMPLEMENTED.

Ukoliko je u naredbi CRT NOT IMPLEMENTED naveden mod CRT D, tada će se CRT D ignorirati, a rezultat će biti da se u novogradnoj PC kartici pojaviti naredba CRT NOT IMPLEMENTED.

Ukoliko je u naredbi CRT NOT IMPLEMENTED naveden mod CRT E, tada će se CRT E ignorirati, a rezultat će biti da se u novogradnoj PC kartici pojaviti naredba CRT NOT IMPLEMENTED.

Ukoliko je u naredbi CRT NOT IMPLEMENTED naveden mod CRT F, tada će se CRT F ignorirati, a rezultat će biti da se u novogradnoj PC kartici pojaviti naredba CRT NOT IMPLEMENTED.

2.23. GREŠKE PRILIKOM PROGRAMIRANJA U BASIC-U

Prilikom pisanja programa u BASIC-u, kao i tokom njihovog izvodjenja, moguce je pojavljivanje raznih gresaka. BASIC kao interaktivni jezik, odmah ce registrirati pogresku i ispisati njezin KOD, te broj programske linije u kojoj se greska pojavila.

Prilikom rada na mikroracunalu DRAO, mogu se pojaviti greske sljedecih kodova:

KOD DEFINICIJA

✓ NF	- NEXT bez prethodnog FOR
✓ SN	- sintakticka greska
✓ RG	- RETURN bez prethodnog GOSUB
✓ OD	- nedovoljno podataka u DATA linijama / vise READ nego DATA /
FC	- greska kod pozivanja funkcije / vrijednost izvan opsega /
OV	- rezultat matematicke operacije je izvan opsega
OM	- memorija je puna
US	- nedefinirana naredba ili pokusaj skoka na nepostojecu liniju
BS	- pogresan poziv, pozvan nedefinirani element polja
DD	- dva puta dimenzionirana matrica
/0	- pokusaj dijeljenja s nulom
ID	- nedozvoljena naredba upotrijebljena u direktnom nacinu
✓ TM	- pogresan tip podataka
✓ LS	- duljina stringa je prevelika, max. 255 znakova
ST	- izraz previse slozen
CN	- nije definiran povratak CONT bez prethodnog STOP ili <CTL><C>
UF	- nije definirana funkcija

3. PROGRAMIRANJE U STROJNOM JEZIKU

Ovaj dio priručnika podijeljen je u tri cjeline:

- 1 - STROJNI JEZIK - UCITI ILI NE ?
 - kratak uvod u programiranje u strojnem jeziku
- 2 - PRIKAZ MNEMONICKIH INSTRUKCIJA MIKROPROCESORA 6502
 - programski model mikroprocesora 6502
 - kratak prikaz svih mnemonickih instrukcija mikroprocesora 6502 (instrukcija strojnog jezika)
- 3 - NACINI ADRESIRANJA MIKROPROCESORA 6502
 - prikaz i objasnjenje svih nacina adresiranja mikroprocesora 6502

Da jos jednom ponovimo - ovaj priručnik nije zamisljen ni kao udžbenik za BASIC, niti kao udžbenik za programiranje u strojnem jeziku. Buduci da će najveći dio vas koristiti mikroracunalo Orao prvenstveno za programiranje u BASIC-u, BASIC-u je zato posvecen najveći dio priručnika i BASIC se pomocu njega može solidno savladati. Međutim, programiranje u strojnem jeziku nije tako jednostavno. Sam uvod u strojni jezik zahtijevao bi toliko prostora koliko ga je potroseno dosad u ovom priručniku. Kod nas vec postoji dovoljno kvalitetne literature za programiranje na strojnem jeziku i to na hrvatsko-srpskom jeziku. Pomocu bilo koje knjige koja obradjuje mikroprocesor 6502 (ili bilo koji 65XX mikroprocesor) naučit cete programiranje na strojnem jeziku i za mikroracunalo Orao.

3.1. STROJNI JEZIK - UČITI ILI NE ?

Za prvi kontakt i upoznavanje s računalima, BASIC je vrlo pogodan programski jezik - vrlo brzo i lako se uči zbog svoje jednostavnosti i razumljivosti.

Pogreske koje su nastale u programu, BASIC interpreter uredno prijavljuje uz razne komentare i poruke o greskama.

Međutim, veliki broj programera dostici će nivo na kojem će im BASIC postati smetnja i prepreka za njihov daljnji napredak. Naime, BASIC uz sve nabrojane prednosti, ima i brojne mane. U momentu kad postanete svjesni tih mana, pravi je trenutak da pocnete razmisljati o strojnem jeziku. Strojni jezik programeru pruža mnogo veće mogućnosti nego BASIC, a uz to je program u strojnem jeziku nausporedivo mnogo brži od programa u BASIC-u.

Kao što svi znamo, mnogo vise vremena ce nam trebati da procitamo neku knjigu napisanu na stranom jeziku, nego knjigu

napisanu na nasej materinjem jeziku. Ili, mnogo lakse i prije cemo se sporazumjeti s osobom koja govori isti jezik kao i mi, nego s osobom koja nas jezik ne razumije, nego joj prevodilac prevodi svaku nasu rijec.

Za mikroracunalo Orao, BASIC je strani, a STROJNI JEZIK je materinji jezik.

Da bi mogli nauciti programirati u materinjem jeziku Orla, morat cete se najprije upoznati s unutrasnjom strukturom mikroracunala Orao i upoznati njegov mikroprocesor 6502.

Srce svakog mikroracunala Orao je mikroprocesor 6502. To je i mozak racunala jer je zaduzen za izvodjenje (skoro) svih operacija i radnji. Posao mikroprocesora je da uzme instrukciju iz memorije i izvrsi ono sto mu je pomocu te instrukcije naredjeno. To moze biti citanje nekog podatka iz memorije, zbrajanje ili oduzimanje dva broja, spremanje podatka u memoriju ili neki drugi jednostavan zadatak. Kad izvrsi jednu, mikroprocesor uzima drugu instrukciju, i tako redom, bez kraja, sve do trenutka dok ne iskljucimo racunalo.

Svaka instrukcija sadrzana je u jednom bajtu memorije u obliku broja. Uz svaku instrukciju nalazi se i jedan ili vise podataka (isto brojeva) - to mogu biti adrese memorijskih lokacija, operandi,... Dakle, u memoriji je zapisano mnoštvo brojeva, koji svi zajedno cine STROJNI PROGRAM mikroprocesora 6502.

Znaci, mogli bi programirati u strojnem jeziku tako da direktno u memorijske lokacije zapisujemo odredjene brojeve. Tocno, ali to je vrlo tezak posao i tako to vise nitko ne radi. Svaki mikroprocesor ima TOCNO ODREDJEN skup instrukcija koji su mu odredili proizvodjaci. U te instrukcije ne spada niti jedna od vama poznatih BASIC naredbi. Svaka instrukcija koju poznaje mikroprocesor ima svoj KOD (numericki). Strojno programiranje se sastoji bas u zapisivanju tih kodova mikroprocesorskih instrukcija u memoriju, zajedno sa svim potrebnim podacima. Slozili smo se vec da se to ne radi tako da se upisuje broj po broj u memoriju. Da bi si olaksali taj posao, koristit cemo jedan specijalni programski jezik koji se zove ASEMLBLER. Naredbe koje poznaje asembler su iste naredbe koje poznaje i mikroprocesor. Medjutim, mikroprocesor prepoznae naredbu samo ako mu je ona predstavljena brojem, a asembler prepoznae tu istu naredbu i ako ju napisemo slovima, kao (dogovoren) rjec, koju ce on prevesti u numericki KOD kako bi mikroprocesor tu naredbu mogao izvrsiti. Tako je naredbe asemblera mnogo lakse razumjeti nego numericke kodove mikroprocesora. Tako, na primjer, naredba mikroprocesoru za povratak iz potprograma ima kod 60 (heksadecimalno). Uz desetke ostalih naredbi (a svaka ima svoj kod), vrlo je vjerojatno da sve te brojeve tesko pamtili. Mnogo lakse cemo upamtiti naredbu "RTS" (ReTurn from Subroutine = povratak iz potprograma). I jos ako imamo program koji ce nam, kad napisemo "RTS", u memorijsku lokaciju zapisati 60 ... To je uloga asemblera. Naredbe koje pripadaju asembleru zovu se MNEMONICI ("RTS" je MNEMONIK), a one predstavljaju SIMBOLICKI JEZIK MIKROPROCESORA. Dakle, programiranje u strojnem jeziku isto je sto i programiranje u asembleru.

Strojno programiranje je mnogo drugacije od programiranja u BASIC-u. Citav program odvijat ce se uz pomoc nekoliko REGISTARA mikroprocesora. Ti registri su memorejske celije mikroprocesora, svaki sa specijalnom namjenom. U tih nekoliko registara moze se smjestiti tek nekoliko bajtova podataka, ali na srecu, sadržaji registara mogu se vrlo brzo pohranjivati u memoriju, a nakon toga, prema potrebi vratiti natrag u registre. Sto se tice aritmetickih operacija, moze se izvoditi samo zbrajanje i oduzimanje (mnogozenje se izvodi pomocu zbrajanja,...), plus logicke operacije AND, OR i EOR. Instrukcije za grananje s jednog dijela strojnog programa na drugi dio su takodjer moguce, uz jos mnogo drugih operacija s bitovima, bajtovima,... Sve zajedno omogucavaju spretnom programeru da iz racunala izvuče maksimum, da postigne sve ono sto se iz BASIC-a nije moglo.

Kao sto nas BASIC marljivo izvjestava o nasim greskama prilikom programiranja, isto tako nas asembler ne izvjestava o slicnim greskama (kontrolira se samo sintaksa). Dok BASIC program ne moze biti logicki nepotpun ili nedorecen, strojni program i te kako moze. Ako, na primjer, instrukciji za spremanje odredjenog podatka damo pogresnu adresu memorejske lokacije, moze se dogoditi da ce podatak biti spremjen bas u sredinu naseg programa, cime ce program biti izmijenjen. Racunalo pri tome ne razmislja da li je ta adresa tocna ili ne. Izvrsavanje takvog programa krenut ce u nezeljenom pravcu, a racunalo ce vjerojatno zaglaviti negdje u tom programu, tako da iz njega nece nikad izaci. Na srecu, za to postoji vrlo djelotvoran lijek - RESET taster koji se nalazi na straznjoj strani racunala. Pritiskom na RESET taster, kontrola racunala je ponovo u monitoru, a nas program je ostao sacuvan u memoriji, tako da ga na ovaj nacin mozemo testirati i ispravljati.

Mozda cete sada pomisliti da od programiranja u ovakvom jeziku nema narocite koristi. Ima, i to velike. Vec smo spomenuli veliku brzinu izvrsavanja strojnih programa. Mnogi korisni potprogrami iz ROM-a mogu se korisno upotrijebiti samo u strojnem programu. Otkrit cete ih jos vise kad sami savladate tajne mikroprocesora 6502.

Da zakljucimo: mikroprocesor 6502 izvrsava programe koji su napisani u strojnem jeziku (strojnem kodu, asembleru). Takvi se programi izvrsavaju mnogo brze od programa pisanih u BASIC-u, ali su dosta komplicirani i teze se pisu, a od programera zahtijevaju mnoga dodatna znanja. Medjutim, pisanje programa u strojnem jeziku je jedini nacin da se iskoriste sve mogucnosti mikroracunala ORAO, a i jedini nacin da se detaljnije upoznate s racunalom.

Strojni jezik - uciti ili ne ? Odlucite sami.

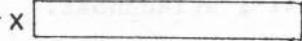
3.2. PRIKAZ MNEMONICKIH INSTRUKCIJA MIKROPROCESORA 6502

3.2.1. PROGRAMSKI MODEL MIKROPROCESORA 6502

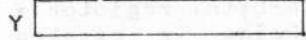
Mikroprocesor 6502 sastoji se od sljedećih registara:



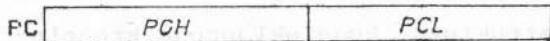
- AKUMULATOR



- INDEKSNI REGISTAR X



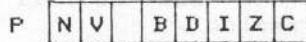
- INDEKSNI REGISTAR Y



- PROGRAMSKO BROJILO



- POKAZIVAC SLOZAJA



- REGISTAR STANJA PROCESORA

BIT C

BIT Z

BIT I

BIT D

BIT B

BIT V

BIT N

AKUMULATOR (A) - je 8-bitni registar koji, osim sto se upotrebljava za privremeno pohranjivanje jednog operanda, sudjeluje pri izvodjenju svih aritmetickih i logickih operacija na podacima. S programskog gledista, akumulator je pravi radni prostor - svi rezultati aritmetickih i logickih operacija pohranjuju se u njemu.

INDEKSNI REGISTRI X i Y - su 8-bitni registri koji se upotrebljavaju pri indeksnom nacinu adresiranja (pri kojem se adresa u instrukcijskoj rjeci i sadržaj indeksnog registra pribrajaju u cilju određivanja stvarne adrese operanda). Uz pomoć indeksnih registara X i Y, mikroprocesor 6502 može tvoriti nekoliko nacija adresiranja koji osnovnom skupu instrukcija mnogostruko povećavaju mogućnosti i efikasnost.

PROGRAMSKO BROJILO (PC-Program Counter) - je 16-bitni registar koji služi za preusmjeravanje toka programa od jedne instrukcije prema drugoj. Budući da je to 16-bitni registar, omogućeno je izravno pribavljanje instrukcije iz cijelog memorijskog prostora od 64 kilobajta.

POKAZIVAC SLOZAJA (S)

Slozaj je memorijска struktura koja uključuje kronološku komponentu, a služi za privremeno pohranjivanje podataka kao što su:

- tekuće stanje mikroprocesora
- povratne adrese
- podaci

U mikroracunalu Orao, slozaj je dio memorijskog prostora koji zauzima 256 bajtova, od adrese 256 do 511 (decimalno), ili od 0100 do 01FF (heksadecimalno).

Slozajem se upravlja samo s dvije instrukcije:

- PH (PUSH) - stavljanje na slozaj
PL (PULL) - vadjenje sa slozaja

Pokazivac slozaja je 8-bitni registar koji sadrži adresu slozaja. Svaki memorijski pristup slozaju (spremanje ili vadjenje podatka) smanjuje ili povećava sadržaj pokazivaca slozaja.

Pokazivac slozaja ima 8 bita koji, sa svoje lijeve strane, uključuje i 9. bit koji je uvijek 1 (0100-01FF). Slozaj uvijek pocinje na adresi 511, a svakim stavljanjem podatka na slozaj, pokazivac slozaja se smanjuje i obrnuto. Podatak koji se najranije stavi na slozaj je uvijek na dnu ("LIFO" - Last In First Out).

REGISTAR STANJA PROCESORA (P) - Mikroprocesor 6502 za vrijeme izvodjenja instrukcija, a u zavisnosti od rezultata aritmeticko-logickih operacija, automatski postavlja ili briše odgovarajuće bitove registra stanja procesora. Testiranjem ovih bitova moguce je utjecati na dalji tok odvijanja programa.

N	V		B	D	I	Z	C
---	---	--	---	---	---	---	---

BIT C (Carry flag) je dojavni bit PRIJENOSA, a ima dvije uloge:

- pohranjuje bit kod aritmetickog prijenosa (prijenos s najznačajnijeg mesta)
- upotrebljava se kao bit pri operacijama posmaka sadržaja registara ili memorijskih lokacija

BIT Z (Zero flag) je dojavni bit NULE i postavlja se ($Z=1$) kada je rezultat aritmetičke operacije nula. Koristi se i za logicke operacije, na primjer usporedjivanje, gdje se, ako su dva operanda jednaka, bit Z postavlja u jedinicu (ako nisu jednaki, Z će biti nula).

BIT I (IRQ disable) je bit PREKIDA. Obично se NE upotrebljava u aritmetičkim operacijama. Bit I se postavlja ($I=1$) u slučaju prekida. Naziva se i PREKIDNA MASKA, jer u slučaju da je $I=1$ i da se pojavi zahtjev za prekid nizg prioriteta, prekid se neće dogoditi.

BIT D (Decimal mode) - aktiviranjem ovog bita, nacin racunanja procesora 6502 prelazi iz potpuno binarnog u decimalno binarni.

BIT B (Break) - pokazuje da li je zahtjev za prekidom (interrupt request) bio prouzrokovani mnemonickom instrukcijom prekida (BRK) ili je prekidni zahtjev bio generiran nekom perifernom jedinicom.

BIT V (overflow-pretek) - pretek se pojavljuje samo kod označene aritmetike i to u slučaju kada je rezultat zbrajanja ili oduzimanja veći od +127 (dec.) ili manji od -128 (dec.).

BIT N (Negative flag) - bit predznaka. Indicira da li je rezultat aritmetičke operacije negativan.

3.2.2. PRIKAZ MNEMONIKA ZA 6502

U ovom dijelu priručnika opisane su sve asembleriske instrukcije mikroprocesora 6502 s pripadajućim nacinima adresiranja. Osim toga, uz svaki mnemonik nalazi se i potrebnii opis registra stanja. Znak "+" predstavlja promjenu u nekom bitu registra stanja, a znak "-" nam govori da u doticnom bitu nije doslo do nikakve promjene.

ADC / PRI BROJI SADRZAJ MEMORIJE AKUMULATORU NZCIDV
 A+M+C--A,C +----+

ADRESIRANJE ASEMLB. FORMAT BR. BAJTOVA BR. CIKLUSA

Izravno	ADC#	operand	2	2
Nulta str.	ADC	operand	2	3
Nulta str.,X	ADC	operand,X	2	4
Apsolutno	ADC	operand	3	4
Apsolutno,X	ADC	operand,X	3	4*
Apsolutno,Y	ADC	operand,Y	3	4*
Preindeksno	ADC	(operand,X)	2	6
Postindeksno	ADC	(operand),Y	2	5*

*dodaje se 1 ciklus ako se prelazi na sljedecu stranicu

AND / LOGICKA OPERACIJA AND MEMORIJE S AKUMULATOROM
 A=A AND M

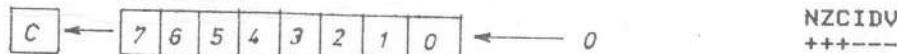
NZCIDV
 +----+

ADRESIRANJE ASEMLB. FORMAT BR. BAJTOVA BR. CIKLUSA

Izravno	AND#	operand	2	2
Nulta str.	AND	operand	2	3
Nulta str.,X	AND	operand,X	2	4
Apsolutno	AND	operand	3	4
Apsolutno,X	AND	operand,X	3	4*
Apsolutno,Y	AND	operand,Y	3	4*
Preindeksno	AND	(operand,X)	2	6
Postindeksno	AND	(operand),Y	2	

*dodaje se 1 ciklus ako se prelazi na sljedecu stranicu

ASL / POMICANJE U LIJEVO MEMORIJE ILI AKUMULATORA
 ZA 1 BIT



NZCIDV
 +----+

ADRESIRANJE ASEMLB. FORMAT BR. BAJTOVA BR. CIKLUSA

Izravno	ASL A		1	2
Nulta str.	ASL	operand	2	5
Nulta str.,X	ASL	operand,X	2	6
Apsolutno	ASL	operand	3	6
Apsolutno,X	ASL	operand,X	3	7

BCC / GRANAJ AKO JE BIT PRIJENOSA (C) BRISAN
- GRANAJ AKO JE C=0

NZCIDV

ADRESIRANJE ASEML. FORMAT BR. BAJTOVA BR. CIKLUSA

Relativno BCC operand 2 2*

*dodaje se jedan ciklus ako se grananje pojavi na istoj stranici.

Dodaju se 2 ciklusa ako se grananje pojavljuje na razlicitim stranicama.

BCS / GRANAJ AKO JE BIT PRIJENOSA POSTAVLJEN
- GRANAJ AKO JE C=1

NZCIDV

ADRESIRANJE ASEML. FORMAT BR. BAJTOVA BR. CIKLUSA

Relativno BCS operand 2 2*

*dodaje se jedan ciklus ako se grananje pojavi na istoj stranici.

Dodaju se 2 ciklusa ako se grananje pojavljuje na razlicitim stranicama.

BEQ / GRANAJ AKO JE REZULTAT NULA
- GRANAJ AKO JE Z=0

NZCIDV

ADRESIRANJE ASEML. FORMAT BR. BAJTOVA BR. CIKLUSA

Relativno BEQ operand 2 2*

*dodaje se jedan ciklus ako se grananje pojavi na istoj stranici.

Dodaju se 2 ciklusa ako se grananje pojavljuje na razlicitim stranicama.

BIT / TESTIRANJE BITOVA MEMORIJE S AKUMULATOROM
A AND M, N=M7, V=M6

NZCIDV
a---b

a=M7 b=M6

Bitovi 6 i 7 se prenose u registar stanja.

ADRESIRANJE ASEML. FORMAT BR. BAJTOVA BR. CIKLUSA

Nulta str.	BIT	operand	2	3
Apsolutno	BIT	operand	3	4

BMI / GRANAJ KOD NEGATIVNOG REZULTATA
- GRANAJ AKO JE N=1

NZCIDV

ADRESIRANJE ASEML. FORMAT BR. BAJTOVA BR. CIKLUSA

Relativno	BMI	operand	2	2*
-----------	-----	---------	---	----

*dodaje se jedan ciklus ako se grananje pojavi na istoj stranici.

Dodaju se 2 ciklusa ako se grananje pojavljuje na razlicitim stranicama.

BNE / GRANAJ AKO REZULTAT NIJE NULA
- GRANAJ AKO JE Z=0

NZCIDV

ADRESIRANJE ASEML. FORMAT BR. BAJTOVA BR. CIKLUSA

Relativno	BNE	operand	2	2*
-----------	-----	---------	---	----

*dodaje se jedan ciklus ako se grananje pojavi na istoj stranici.

Dodaju se 2 ciklusa ako se grananje pojavljuje na razlicitim stranicama.

BPL / GRANAJ AKO JE REZULTAT POZITIVAN
- GRANAJ AKO JE N=0

NZCIDV

ADRESIRANJE ASEML. FORMAT BR. BAJTOVA BR. CIKLUSA

Relativno BPL operand 2 2*

*dodaje se jedan ciklus ako se grananje pojavi na istoj stranici.

Dodaju se dva ciklusa ako se grananje pojavi na razlicitim stranicama.

BRK / FORSIRANJE PREKIDA

NZCIDV

---1---

ADRESIRANJE ASEML. FORMAT BR. BAJTOVA BR. CIKLUSA

Ukljucujuce BRK 1 1

BVC / GRANAJ KOD BRISANOG PRETEKA

- GRANAJ AKO JE V=0

NZCIDV

ADRESIRANJE ASEML. FORMAT BR. BAJTOVA BR. CIKLUSA

Relativno BVC operand 2 2*

*dodaje se jedan ciklus ako se grananje pojavi na istoj stranici.

Dodaju se dva ciklusa ako se grananje pojavi na razlicitim stranicama.

BVS / GRANAJ AKO JE PRETEK POSTAVLJEN

- GRANAJ AKO JE V=1

NZCIDV

ADRESIRANJE ASEML. FORMAT BR. BAJTOVA BR. CIKLUSA

Relativno BVS operand 2 2*

*dodaje se jedan ciklus ako se grananje pojavi na istoj stranici.

Dodaju se dva ciklusa ako se grananje pojavi na razlicitim stranicama.

CLC / BRISANJE BITA PRIJENOSA
- C=0

NZCIDV
---0---

ADRESIRANJE ASEML. FORMAT BR. BAJTOVA BR. CIKLUSA

Ukljucujuće CLC 1 2

CLD / BRISANJE DECIMALNOG MODA
- D=0

NZCIDV
----0-

ADRESIRANJE ASEML. FORMAT BR. BAJTOVA BR. CIKLUSA

Ukljucujuće CLD 1 2

CLI / BRISANJE BITA ZA ONEMOGUCAVANJE PREKIDA
- I=0

NZCIDV
---0---

ADRESIRANJE ASEML. FORMAT BR. BAJTOVA BR. CIKLUSA

Ukljucujuće CLI 1 2

CLV / BRISANJE BITA PRETEKA
- V=0

NZCIDV
----0

ADRESIRANJE ASEML. FORMAT BR. BAJTOVA BR. CIKLUSA

Ukljucujuće CLV 1 2

CMP / USPOREDJIVANJE MEMORIJE S AKUMULATOROM
A-M

NZCIDV
+---

ADRESIRANJE ASEML. FORMAT BR. BAJTOVA BR. CIKLUSA

Izravno CMP# operand 2 2
Nulta str. CMP operand 2 3
Nulta str.,X CMP operand,X 2 4
Apsolutno CMP operand 3 4
Apsolutno,X CMP operand,X 3 4*
Apsolutno,Y CMP operand,Y 3 4*
Preindeksno CMP (operand,X) 2 6
Postindeksno CMP (operand),Y 2 5*

*dodaje se 1 ciklus ako se prelazi na slijedecu stranicu

CPX / USPOREDJIVANJE MEMORIJE S INDEKSnim REGISTROM X
X-M

ADRESIRANJE	ASEMBL. FORMAT	BR. BAJTOVA	BR. CIKLUSA	NZCIDV +----
Izravno	CPX# operand	2	2	
Nulta str.	CPX operand	2	3	
Apsolutno	CPX operand	3	4	

CPY / USPOREDJIVANJE MEMORIJE S INDEKSnim REGISTROM Y
Y-M

ADRESIRANJE	ASEMBL. FORMAT	BR. BAJTOVA	BR. CIKLUSA	NZCIDV +----
Izravno	CPY# operand	2	2	
Nulta str.	CPY operand	2	3	
Apsolutno	CPY operand	3	4	

DEC / UMANJI SADRZAJ MEMORIJE ZA 1
M=M-1

ADRESIRANJE	ASEMBL. FORMAT	BR. BAJTOVA	BR. CIKLUSA	NZCIDV +----
Nulta str.	DEC operand	2	2	
Nulta str.,X	DEC operand,X	2	6	
Apsolutno	DEC operand	3	6	
Apsolutno,X	DEC operand,X	3	7	

DEX / UMANJI INDEKSNI REGISTAR X ZA 1
X=X-1

ADRESIRANJE	ASEMBL. FORMAT	BR. BAJTOVA	BR. CIKLUSA	NZCIDV +----
Ukljucujuće	DEX	1	2	

DEY / UMANJI INDEKSNI REGISTAR Y ZA 1
Y=Y-1

ADRESIRANJE	ASEMBL. FORMAT	BR. BAJTOVA	BR. CIKLUSA	NZCIDV +----
Ukljucujuće	DEY	1	2	

EOR / EKSKLUZIVNO "ILI" MEMORIJE S AKUMULATOROM
A=A EXOR M

ADRESIRANJE	ASEMBL.	FORMAT	BR. BAJTOVA	BR. CIKLUSA	NZCIDV
					++---
Izravno	EOR#	operand	2	2	
Nulta str.	EOR	operand	2	3	
Nulta str.,X	EOR	operand,X	2	4	
Apsolutno	EOR	operand	3	4	
Apsolutno,X	EOR	operand,X	3	4*	
Apsolutno,Y	EOR	operand,Y	3	4*	
Preindeksno	EOR	(operand,X)	2	6	
Postindeksno	EOR	(operand),Y	2	5*	

*dodaje se 1 ciklus ako se prelazi na slijedecu stranicu

INC / POVECAJ SADRZAJ MEMORIJE ZA 1
M=M+1

ADRESIRANJE	ASEMBL.	FORMAT	BR. BAJTOVA	BR. CIKLUSA	NZCIDV
					++---
Nulta str.	INC	operand	2	5	
Nulta str.,X	INC	operand,X	2	6	
Apsolutno	INC	operand	3	6	
Apsolutno,X	INC	operand,X	3	7	

INX / POVECAJ SADRZAJ INDEKSNOG REGISTRA X ZA 1
X=X+1

ADRESIRANJE	ASEMBL.	FORMAT	BR. BAJTOVA	BR. CIKLUSA	NZCIDV
					++---
Ukljucujuce	INX		1	2	

INY / POVECAJ SADRZAJ INDEKSNOG REGISTRA Y ZA 1
Y=Y+1

ADRESIRANJE	ASEMBL.	FORMAT	BR. BAJTOVA	BR. CIKLUSA	NZCIDV
					++---
Ukljucujuce	INY		1	2	

JMP / SKOK NA NOVU LOKACIJU
PCL=(PC+1)
PCH=(PC+2)

NZCIDV
+----

ADRESIRANJE ASEML. FORMAT BR. BAJTOVA BR. CIKLUSA

Apsolutno	JMP operand	3	3
Indirektno	JMP (operand)	3	5

JSR / SKOK NA FOTPROGRAM
PCL=(PC+1)
PCH=(PC+2)

NZCIDV
+----

ADRESIRANJE ASEML. FORMAT BR. BAJTOVA BR. CIKLUSA

Apsolutno	JSR operand	3	6
-----------	-------------	---	---

LDA / PUNI AKUMULATOR SADRZAJEM MEMORIJE
A=M

NZCIDV
+----

ADRESIRANJE ASEML. FORMAT BR. BAJTOVA BR. CIKLUSA

Izravno	LDA# operand	2	2
Nulta str.	LDA operand	2	3
Nulta str.,X	LDA operand,X	2	4
Apsolutno	LDA operand	3	4
Apsolutno,X	LDA operand,X	3	4*
Apsolutno,Y	LDA operand,Y	3	4*
Preindeksno	LDA (operand,X)	2	6
Postindeksno	LDA (operand),Y	2	5*

*dodaje se 1 ciklus ako se prelazi na novu stranicu

LDX / PUNI INDEKSNI REGISTAR X SADRZAJEM MEMORIJE
X=M

NZCIDV
+----

ADRESIRANJE ASEML. FORMAT BR. BAJTOVA BR. CIKLUSA

Izravno	LDX# operand	2	2
Nulta str.	LDX operand	2	3
Nulta str.,Y	LDX operand,Y	2	4
Apsolutno	LDX operand	3	4
Apsolutno,Y	LDX operand,Y	3	4*

*dodaje se 1 ciklus ako se prelazi na slijedecu stranicu

LDY / NAPUNI INDEKSNI REGISTAR Y S MEMORIJOM

Y=M

NZCIDV

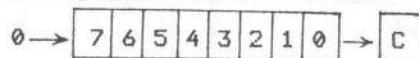
++---

ADRESIRANJE ASEMLB. FORMAT BR. BAJTOVA BR. CIKLUSA

Izravno	LDY# operand	2	2
Nulta str.	LDY operand	2	3
Nulta str.,X	LDY operand,X	2	4
Apsolutno	LDY operand	3	4
Apsolutno,X	LDY operand,X	3	4*

*dodaje se 1 ciklus ako se prelazi na slijedecu stranicu

LSR / LOGICKO POMICANJE U DESNO MEMORIJE ILI AKUMULATORA ZA 1 BIT



NZCIDV

++---

ADRESIRANJE ASEMLB. FORMAT BR. BAJTOVA BR. CIKLUSA

Akumulatorsko	LSR A	1	2
Nulta str.	LSR operand	2	5
Nulta str.,X	LSR operand,X	2	6
Apsolutno	LSR operand	3	4
Apsolutno,X	LSR operand,X	3	7

NOP / NE IZVRSAVA NISTA

NZCIDV

ADRESIRANJE ASEMLB. FORMAT BR. BAJTOVA BR. CIKLUSA

Ukljucujuce	NOP	1	2
-------------	-----	---	---

ORA / LOGICKA OPERACIJA "ILI" S. MEMORIJOM ILI AKUMULAT.

A=A ORA M

NZCIDV

++---

ADRESIRANJE ASEMLB. FORMAT BR. BAJTOVA BR. CIKLUSA

Izravno	ORA# operand	2	2
Nulta str.	ORA operand	2	3
Nulta str.,X	ORA operand,X	2	4
Apsolutno	ORA operand	3	4
Apsolutno,X	ORA operand,X	3	4*
Apsolutno,Y	ORA operand,Y	3	4*
Preindeksno	ORA (operand,X)	2	6
Postindesno	ORA (operand),Y	2	5*

*dodaje se 1 ciklus ako se prelazi na slijedecu stranicu

PHA / POSPREMI SADRZAJ AKUMULATORA U SLOZAJ

NZCIDV

ADRESIRANJE ASEMLB. FORMAT BR. BAJTOVA BR. CIKLUSA

Ukljucujuće PHA 1 3

PHP / POSPREMI SADRZAJ REGISTRA STANJA U SLOZAJ

NZCIDV

ADRESIRANJE ASEMLB. FORMAT BR. BAJTOVA BR. CIKLUSA

Ukljucujuće PHP 1 3

PLA / UZMI SADRZAJ AKUMULATORA IZ SLOZAJA

NZCIDV

+++++

ADRESIRANJE ASEMLB. FORMAT BR. BAJTOVA BR. CIKLUSA

Ukljucujuće PLA 1 4

PLP / UZMI SADRZAJ REGISTRA STANJA IZ SLOZAJA

NZCIDV

+++++

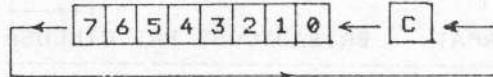
ADRESIRANJE ASEMLB. FORMAT BR. BAJTOVA BR. CIKLUSA

Ukljucujuće PLP 1 4

ROL / ROTIRANJE U LIJEVO MEMORIJE ILI AKUMULATORA ZA
JEDAN BIT

NZCIDV

+++++



ADRESIRANJE ASEMLB. FORMAT BR. BAJTOVA BR. CIKLUSA

Akumulatorsko ROL A 1 2

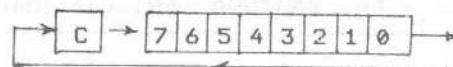
Nulta str. ROL operand 2 5

Nulta str.,X ROL operand,X 2 6

Apsolutno ROL operand 3 6

Apsolutno,X ROL operand,X 3 7

ROR / ROTIRANJE U DESNO MEMORIJE ILI AKUMULATORA ZA JEDAN BIT



NZCIDV
+++++

ADRESIRANJE	ASEMBL. FORMAT	BR. BAJTOVA	BR. CIKLUSA
<hr/>			
Akumulatorsko	ROR A	1	2
Nulta str.	ROR operand	2	5
Nulta str.,X	ROR operand,X	2	6
Apsolutno	ROR operand	3	6
Apsolutno,X	ROR operand,X	3	7

RTI / POV'RATAK IZ PREKIDA

NZCIDV
iz slozaja

ADRESIRANJE	ASEMBL. FORMAT	BR. BAJTOVA	BR. CIKLUSA
<hr/>			
Ukljucujuce	RTI	1	6

RTS / POV'RATAK IZ POTPROGRAMA

NZCIDV

ADRESIRANJE	ASEMBL. FORMAT	BR. BAJTOVA	BR. CIKLUSA
<hr/>			
Ukljucujuce	RTS	1	6

SBC / ODUZIMANJE MEMORIJE OD AKUMULATORA S PRIJENOSOM
 $A, C = A - M - (C-1)$

NZCIDV
+++++

ADRESIRANJE	ASEMBL. FORMAT	BR. BAJTOVA	BR. CIKLUSA
<hr/>			
Izravno	SBC# operand	2	2
Nulta str.	SBC operand	2	3
Nulta str.,X	SBC operand,X	2	4
Apsolutno	SBC operand	3	4
Apsolutno,X	SBC operand,X	3	4*
Apsolutno,Y	SBC operand,Y	3	4*
Preindeksno	SBC (operand,X)	2	6
Postindeksno	SBC (operand),Y	2	5*

*dodaje se 1 ciklus ako se prelazi na slijedecu stranicu

SEC / POSTAVI BIT PRIJENOSA
 $C=1$

NZCIDV
---1---

ADRESIRANJE	ASEMBL. FORMAT	BR. BAJTOVA	BR. CIKLUSA
<hr/>			
Ukljucujuce	SEC	1	2

SED / POSTAVI DECIMALNI BIT

D=1

NZCIDV

---1---

ADRESIRANJE ASEML. FORMAT BR.BAJTOVA BR. CIKLUSA

Ukljucujuce SED 1 2

SEI / POSTAVI BIT PREKIDA

I=1

NZCIDV

---1---

ADRESIRANJE ASEML. FORMAT BR.BAJTOVA BR. CIKLUSA

Ukljucujuce SEI 1 2

STA / POSPREMI SADRZAJ AKUMULATORA U MEMORIJU

M=A

NZCIDV

ADRESIRANJE ASEML. FORMAT BR.BAJTOVA BR. CIKLUSA

Nulta str.	STA	operand	2	3
Nulta str.,X	STA	operand,X	2	4
Apsolutno	STA	operand	3	4
Apsolutno,X	STA	operand,X	3	5
Apsolutno,Y	STA	operand,Y	3	5
Freindeksno	STA	(operand,X)	2	6
Postindeksno	STA	(operand),Y	2	6

STX / POSPREMI SADRZAJ INDEKSNOG REGISTRA X U MEMORIJU

M=X

NZCIDV

ADRESIRANJE ASEML. FORMAT BR.BAJTOVA BR. CIKLUSA

Nulta str.	STX	operand	2	3
Nulta str.,Y	STX	operand,Y	2	4
Apsolutno	STX	operand	3	4

STY / POSPREMI SADRZAJ INDEKSNOG REGISTRA Y U MEMORIJU

M=Y

NZCIDV

ADRESIRANJE ASEML. FORMAT BR.BAJTOVA BR. CIKLUSA

Nulta str.	STY	operand	2	3
Nulta str.,X	STY	operand,X	2	4
Apsolutno	STY	operand	3	4

TAX / PREMJESTI SADRZAJ AKUMULATORA U REGISTAR X
X=A

NZCIDV

++--

ADRESIRANJE ASEML. FORMAT BR. BAJTOVA BR. CIKLUSA

Uk1 jucujuce TAX 1 2

TAY / PREMJESTI SADRZAJ AKUMULATORA U REGISTAR Y
Y=A

NZCIDV

++--

ADRESIRANJE ASEML. FORMAT BR. BAJTOVA BR. CIKLUSA

Uk1 jucujuce TAY 1 2

TSX / PRENESI SADRZAJ KAZALJKE SLOZAJA U REGISTAR X
X=S

NZCIDV

++--

ADRESIRANJE ASEML. FORMAT BR. BAJTOVA BR. CIKLUSA

Uk1 jucujuce TSX 1 2

TXA / PRENESI SADRZAJ INDEKSNOG REGISTRA X U AKUMULATOR
X=A

NZCIDV

++--

ADRESIRANJE ASEML. FORMAT BR. BAJTOVA BR. CIKLUSA

Uk1 jucujuce TXA 1 2

TXS / PRENESI SADRZAJ REGISTRA X U KAZALO SLOZAJA
X=S

NZCIDV

++--

ADRESIRANJE ASEML. FORMAT BR. BAJTOVA BR. CIKLUSA

Uk1 jucujuce TXS 1 2

TYA / PRENESI SADRZAJ INDEKSNOG REGISTRA Y U AKUMULATOR
Y=A

NZCIDV

++--

ADRESIRANJE ASEML. FORMAT BR. BAJTOVA BR. CIKLUSA

Uk1 jucujuce TYA 1 2

3.3. NACINI ADRESIRANJA MIKROPROCESORA 6502

Jedna od najvaznijih prednosti mikroprocesora 6502 lezi u njegovim brojnim nacinima adresiranja. Ima ih 13 i oni mu omogucaavaju rjesavanje zadane problematike lako i brzo. Usprendjujuci ostale, cesto koristene 8-bitne mikroprocesore, vidimo da 6502 ima sest adresnih nacina vise nego Motorola 6800, osam vise od Intela 8080 ili 8085, tri vise nego Zilog Z80 i pet vise od Texas Instruments-a 9900.

Zbog lakseg razumijevanja i nejfikasnijeg koristenja svakog nacina adresiranja koji su moguci na 6502, u ovom dijelu cemo kratko opisati svaki adresni nacin uz prikaz pravilne sintakse.

Napomena: kad mikroracunalo Oroo radi u MONITOR-u, prihvaca iskljucivo HEKSADECIMALNE brojeve. Brojevi se unose bez ikakvih prefiksa i svi se prihvacaju kao heksadecimalni brojevi.

3.3.1. Izravno adresiranje (Immediate addressing)

Ovaj nacin adresiranja omogucava izravni rad s 8-bitnim brojevima. "Izravni" operand mora biti proslijedjen znakom # (hash).

Takav nacin omogucava rad s konstantama, kao kad, na primjer, zelimo u akumulator zapisati vrijednost A1:

LDA #A1

Izravni nacin adresiranja zauzima dva bajta memorije.

3.3.2. Akumulatorsko adresiranje (Accumulator addressing)

Mikroprocesor 6502 ima cetiri instrukcije koje omogucavaju pomicanje ili rotiranje sadrzaja akumulatora za jedan bit u lijevo ili u desno. Upucivanje na ovaj adresni nacin postizemo dodavanjem slova "A" uz mnemonik.

Za primjer mozemo uzeti rotiranje akumulatora u desno:

ROR A

3.3.3. Relativno adresiranje (Relative addressing)

Efektivna adresa kod relativnog nacina adresiranja dobija se pribajanjem trenutnog stanja programskog brojila s adresnim dodatkom (offset-om). Adresni dodatak moze biti pozitivan ili negativan, sto omogucuje 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 uzimimo:

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

Priloženi primjer opisuje slučaj grananja (pozitivni "offset"), ako je sadržaj akumulatora razlicit od #4F. Kod sadržaja akumulatora od #4F neće doći do grananja, odnosno instrukcija BNE (Branch on result Not Equal to zero) se jednostavno preskace.

Sve "branch" instrukcije (BNE, BEQ, BCC, BCS, BMI,...) zauzimaju po dva bajta memorije, s time da drugi bajt određuje velicinu pozitivnog ili negativnog dodatka programskom brojilu.

3.3.4. Apsolutno adresiranje (Absolute addressing)

Apsolutno adresiranje omogucava direktno adresiranje bilo koje memorijske lokacije mikroracunala Orao, a to znači unutar granica od 0 do 65535.

Sve instrukcije kod apsolutnog nacina adresiranja zahtijevaju tri bajta. Prvi bajt predstavlja kod mnemonicke oznake instrukcije, dok su drugi i treci bajt predviđeni za upis adrese. Napunimo akumulator sadržajem lokacije na adresi ABCD:

LDA ABCD

Ovdje treba napomenuti da se, sa stanovista memorijske mape racunala, prvo upisuje kod mnemonicke instrukcije (u nasem slučaju to je AD=kod instrukcije LDA), zatim manje značajan bajt adrese (CD), a na kraju više značajan bajt (AB) apsolutne adrese.

3.3.5. Apsolutno indeksno adresiranje (Absolute indexed addressing)

Ovaj nacin adresiranja razlikuje se od prethodno opisanog po tome sto se efektivna adresa tvori kao zbroj absolutne adrese i sadrzaja indeksnog registra (X ili Y).

Efektivna adresa = absolutna adresa + X
Efektivna adresa = absolutna adresa + Y

Operand kod apsolutno indeksnog adresiranja dobije se dodavanjem vrijednosti indeksnog registra (X ili Y) apsolutnoj adresi.

LDA ABC0,X
LDA ABC0,Y

Ako je sadrzaj X registra 05, a sadrzaj Y registra 09, efektivne adrese za navedene primjere biti ce:

ABC0+05=ABC5
ABC0+09=ABC9

Dodavanjem indeksa apsolutnom adresiranju dobili smo vvrlo vaznu kvalitetu prilikom oblikovanja strojnih programa - utoliko vise sto raspolazemo s dva indeksna registra, a to nam omogucuje pristup do svake lokacije u memoriji mikroracunala ORAO.

3.3.6. Adresiranje nulte stranice (Zero page addressing)

Adresiranje nulte stranice slicno je apsolutnom adresiranju, samo sto se kod ovog nacina adresiranja efektivna adresa nalazi unutar jednog bajta.

LDA AA

Ovaj primjer posprema sadrzaj memorijске lokacije AA u akumulator. Ovu istu operaciju mozemo napraviti koristeci apsolutno adresiranje:

LDA 00AA

Razlika je u tome sto kod adresiranja nulte stranice trosimo dva bajta memorijje, a kod apsolutnog adresiranja tri bajta. Naravno, povecani broj bajtova dodatno opterecuje memoriju, a gubi se i na brzini. Prilikom programiranja u strojnom jeziku mora se posebno voditi briga oko koristenja i trosenja nulte stranice, jer je mikroprocesor 6502 vezan bas uz nultu stranicu memorije.

3.3.7. Uključujuće adresiranje (Implied addressing)

Ovo adresiranje, za razliku od ostalih adresnih nacina, nema operanda. On je UKLJUCEN u mnemoniku, a odnosi se na povecanje ili smanjenje indekdnog registra (X ili Y), te postavljanje ili brisanje bitova u registru stanja procesora. Navedimo nekoliko primjera:

DEY - smanjenje Y registra za 1
DEX - smanjenje X registra za 1
SEC - postavljanje bita prijenosa
Sve instrukcije vezane uz uključujuće adresiranje zauzimaju po jedan bajt memorije.

3.3.8. Indirektno apsolutno adresiranje (Indirect absolute addressing)

Indirektno apsolutno adresiranje koristi se kod mikroprocesora 6502 samo instrukciju indirektnog skoka. Odredisnu adresu dobijamo preko sadržaja dviju apsolutnih adresa, dakle indirektno.

Za lakše razumijevanje navedimo primjer indirektnog apsolutnog adresiranja iz mikroracunala Orao. Opisat ćemo primjer potprograma za ispis znaka za ekran (OUTCH). Adresa ovog potprograma u ROM-u je E762, međutim u sistemskom softveru primjetit ćemo da se taj potprogram nalazi za adresi FFF1, dakle za ispis znaka koristimo JSR FFF1. Fogledajmo disasemblierski listing:

FFF1 6C 18 02 JMF (0218)

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

Stvarna adresa ovog potprograma sadrzana je u memorijskim lokacijama 0218 i 0219. Možda će se neki korisnici iznenaditi zbog cega se koristi taj nacin pozivanja OUTCH potprograma, kad bi jednostavnije bilo uvijek pisati JSR E762. Međutim, ovaj nacin omogucava tzv. preusmjerenje vektora na željenu memorijsku lokaciju, što znaci dozvoliti korisniku kreiranje vlastitog OUTCH potprograma.

3.3.9. Indeksno adresiranje nulte stranice (Zero page indexed addressing)

Slicno kao što smo imali kod usporedjivanja apsolutnog i apsolutno indeksnog adresiranja, možemo i sada usporedjivati indeksno adresiranje nulte stranice s "običnim" adresiranjem nulte stranice. Dakle, razlika postoji samo u velicini

adresnog operanda, od trobojtnog smo presli na dvobajtni operand.

LDA AB,X
LDA 45,Y

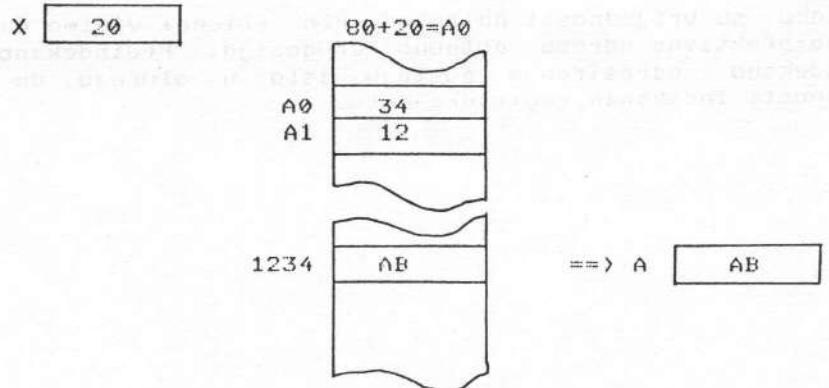
Ulogu indeksa moze na sebe preuzeti X ili Y registar. Odredisna adresa tvori se kao zbroj sadrzaja indeksnog registra (indeksa) i adrese.

3.3.10. Preindeksno adresiranje (Indexed indirect addr.)

Ovaj nacin adresiranja predstavlja kombinaciju indirektnog i indeksnog nacina adresiranja. Znamo da kod indeksnog adresiranja efektivnu adresu dobivamo kao zbroj neke osnovne adrese i sadrzaja indeksnog registra, a kod indirektnog nacina adresiranja operand u instrukciji vec predstavlja adresu, ciji sadrzaj daje konacnu efektivnu adresu. Kod vih indirekcija koriste se dva bajta u memoriji zbog mogucnosti 16 bitnog adresiranja, dakle, pokrivanja svih 65536 adresnih lokacija.

Mozda ce slijedeci primjer olaksati razumijevanje preindeksnog adresiranja. Uzmimo da je sadrzaj indeksnog registra 20, a adresa 80:

LDA (80,X)



Efektivnu adresu nulte stranice dobijemo zbrojem adrese i indeksa.

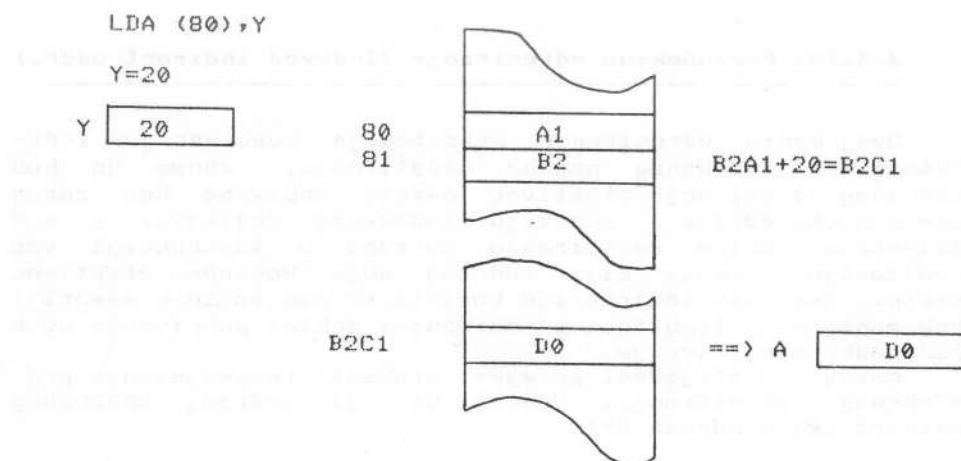
$$(80+20)=A0$$

Manje znacajan bajt efektivne adrese dobijemo kao sadrzaj lokacije A0, a znacajniji bajt kao sadrzaj slijedeće lokacije - A1. Dakle, konacna efektivna adresa je 1234. Sadrzajem lokacije 1234 konacno punimo akumulator i nastavljamo program.

3.3.11. Postindeksno adresiranje (Indirect indexed add.)

Poстоји izvjesna sličnost između preindeksnog i postindeksnog adresiranja. I u jednom i u drugom slučaju kombiniramo indeksno i indirektno adresiranje, a razlika je samo u redoslijedu. Kod prethodnog nacina prvo zbrojimo adresu i vrijednost indeksa, pa onda izvršimo indirekciju, a kod postindeksnog prvo nacinimo indirekciju, a tek onda pribrajamo indeks da bi dobili efektivnu adresu.

Cijeli problem pokusajmo laksati malim primjerom:



Iako su vrijednosti naizgled vrlo slike, vidimo da je konacna efektivna adresa potpuno drugacija. Preindeksno i postindeksno adresiranje postaju isto u slučaju da su vrijednosti indeksnih registara nula.

4. OPIS MONITORA

Rad u MONITOR-u predviđen je uglavnom za korisnike koji će programirati u strojnem jeziku, ovisno ili neovisno o BASIC-u. Sa zadovoljstvom će ga koristiti svi - i oni koji tek uče strojno programiranje, ali i oni koji pisu ozbiljne asemblerске programe.

Korisniku, naime, treba omogućiti pregled, izmjene i kopiranje sadržaja memorije, disasembliranje pojedinih dijelova memorije i sличno. MONITOR-ski program, dakle, nije namijenjen isključivo programeru koji se interesira za pisanje programa u asembleru - koristit ćemo ga kada krah programa ugrozi sate i sate naseg rada, kada naidjemo na neki dobro zasticeni program ...

Rad u MONITOR-u sličan je donekle direktnom nacinu rada u BASIC-u. Od jedanaest monitorskih naredbi mikroracunala Orao, ne može se sastaviti nikakav program. Naredbe se uvijek zadaju direktno i odmah se izvršavaju. Nije moguće navesti ni dvije naredbe odvojene dvotockom - vrijeme je da "zaboravite" navike iz BASIC-a.

Uskoro ćete se upoznati sa svim monitorskim naredbama mikroracunala Orao. Ima ih jedanaest i svaka se sastoji od samo jednog slova (?!). Tek kad ostavimo BASIC vidimo koliko nam je znacio. Razlog zbog kojeg su ove naredbe tako "kratke" nije taj što nije bilo moguće smisliti i napisati duže naredbe. Pri radu s MONITOR-om se vrlo mnogo puta ponavljaju isti postupci, vrlo ćeste izmjene pregledavanja sadržaja memorije, disasembliranja, seljenja bloka podataka, pa opet izmjene... U toku analize nekog programa stotinjak puta ćete koristiti naredbu za disasembliranje - njezinim "skracivanjem" na jedno slovo postedjeni ste mukotrpnog tipkanja po tastaturi, a uestidi se i u vremenu. Monitor nema mnogo naredbi i njih ćete vrlo lako zapamtiti.

Na kraju ovog uvodnog dijela još ćemo jedanput naglasiti:

- svi brojevi koje unosimo u monitoru (bilo da su to adrese ili podaci) moraju biti u HEKSADECIMALNOJ notaciji.

U mikroracunalu Orao postaje slijedeće monitorske naredbe:

- - POZIVANJE BASIC-a
- <A> - MINIASEMBLER
- <X> - DISASEMBLER
- <E> - CITANJE MEMORIJSKOG BLOKA (DUMP)
- <H> - CITANJE MEMORIJSKOG BLOKA U ASCII OBLIKU
- <M> - CITANJE I MIJENJANJE SADRŽAJA MEMORIJSKE LOKACIJE
- <C> - PROVJERA SUME
- <F> - PUNJENJE MEMORIJSKOG BLOKA
- <U> - IZVRSENJE STROJNOG PROGRAMA
- <Q> - KOPIRANJE DIJELA MEMORIJE
- <#> - PRETVARANJE BROJEVA IZ HEKSADECIMALNOG U DECIMALNI

4.1. - POZIVANJE BASIC-a IZ MONITOR-a

Radi se o, nama vec dobro poznatim naredbama

BC i BW

Nakon ukljucivanja racunala nalazimo se u Monitoru. Ulazak u Basic prvi puta moguc je jedino pomocu monitorske naredbe:

*BC <CR>

Ovime se brise sadrzaj RAM-a i inicijalizira se BASIC. Ulazak u Basic na ovaj nacin zove se HLADNI START Basica.

Kad je Basic jedanput hladno startan, moguce je po zelji prelaziti u Monitor i vracati se natrag u Basic i to pomocu monitorske naredbe:

*BW <CR>

Pozivanje Basica na ovaj nacin zove se VRUCI START Basica. Vrucim startom Basica ne dira se sadrzaj memorije, on ostaje sacuvan. Vruci start moguc je jedino ako je Basic vec jedanput bio hladno startan i ako nisu promijenjene vrijednosti u nultoj stranici memorije.

4.2. <A> - MINIASSEMBLER

Pisanje programa u strojnog kodu moguce je pomocu ugradjenog miniasemblera koji izravno pretvara raspoznatljivi mnemonik u odgovarajuci strojni kod, tako da nam miniasembler cijelo vrijeme ispisuje absolutne adrese na koje se smjesti generirani strojni kod. Miniasembler ne prepoznaje labele i sve adrese (relativne i absolutne) i svi skokovi se moraju navoditi absolutno.

Napisat cemo jedan kratak strojni program i pomocu njega cemo postepeno upoznavati nove monitorske naredbe mikroracuna Orao. Neka to bude program za zbrajanje dva broja:

- zadatok je da zbrojimo dva heksadecimalna broja (8 i 7). Program neka pocne od memorijске lokacije 1000 (heksadecimalno, naravno !), a rezultat zbrajanja neka se smjesti u memorijsku lokaciju 0501:

Miniasembler pozivamo naredbom:

Annnn <CR>

gdje je "nnnn" adresa od koje cemo poceti pisati program i od koje ce se smjestati generirani strojni kod.

Naredbe se unose u standardnom formatu. "Udjite" u miniasembler pomocu naredbe "A1000" i prepisite program:

```
*A1000 <CR>
1000 LDA #08    <CR>          nnah X
1002 STA 0500   <CR>
1005 LDA #07    <CR>
1007 CLC        <CR>
1008 ADC 0500   <CR>
100B STA 0501   <CR>
100E RTS       <CR>
100F Q         <CR>
?
*—
```

Pogledajmo kako program radi. U prvom koraku (LDA #08) smo izravno upisali broj 8 u akumulator i pospremili ga radi kasnijeg zbrajanja u lokaciju 0500 (STA 0500). Zatim smo ponovo napunili akumulator drugim operandom, brojem 7 (LDA#07) i prijenos stavili na nulu (CLC). U sljedecem koraku zbrojili smo trenutni sadrzaj akumulatora (7) sa sadrzajem lokacije 0500 (8). Dobivena suma trenutno se nalazi u akumulatoru, koju u sljedecem koraku spremamo u lokaciju 0501.

Kao sto ste mogli promjetiti, pomocu "Q" smo izasli iz miniasemblera. Mogli smo izaci tako da bi upisali bilo koji znak koji ne predstavlja niti jedan 6502 mnemonik.

Na taj smo nacin zbrojili dva heksadecimalna broja, ne razmisljajući o mogucnosti da njihov zbroj moze biti i veci od 255, sto prelazi mogucnost upisa u samo jednu lokaciju. Da je, u kojem slucaju, zbroj veci od 255, pojavio bi se bit prijenosa (C=1). Nakon zbrajanja mozemo ispitati bit prijenosa u registru stanja procesora i tako znati da li je zbroj veci ili manji od 255. Evo primjer kad bi bio veci:

$$\begin{array}{r} 80 \\ + 8F \\ \hline 10F \end{array}$$

Ovdje se primjecuje potreba za jos jednim, devetim bitom, koji ne postoji niti u jednoj memorijskoj lokaciji. Da bi se moglo raditi s visebajtnim podacima, problem je rjesen pomocu C bita (bita prijenosa) u registru stanja. U ovom slucaju u akumulatoru ostaje vrijednost 0F, a i prelazi u C bit (zato je obavezna CLC instrukcija prije zbrajanja).

4.3. <X> - DISASSEMBLER

Disassembler se poziva naredbom

X nnnn

gdje "nnnn" predstavlja heksadecimalnu adresu pocetka dijela memorije koji zelimo disasembliрати.

Disassembler ima obrnutu ulogu od asemblera. Uzima strojni kod (broj) iz memorije i pretvara ga u mnemonicku oznaku instrukcije. Njegova upotrebljivost je vrlo velika.

Disasemblirajte program koji ste malo prije upisali:

```
*X1000 <CR>
 1000 A9 0B      LDA #0B
 1002 8D 00 05    STA 0500
 1005 A9 07      LDA #07
 1007 18          CLC
 1008 6D 00 05    ADC 0500
 100B 8D 01 05    STA 0501
 100E 60          RTS
 100F 0F          @@@
 1010 1C          @@@
 1011 28          PLP
 1012 AD 0F 50    LDA 500F
 1015 94 AB      STY AB,X
```

*

Pojavilo se, dakle, 12 linija disasemblierskog listinga. Nas program se nalazi u memoriji od lokacije 1000 do 100E. Sto se nalazi iza toga nas ne zanima, ovdje je to slucajan sadrzaj memorije, ili dio nekog starog programa.

Ako zelite disasembliратi točno određen dio memorije, uz naredbu X i pocetnu adresu, napisite i zavrsnu adresu disasembliranja. Probajte:

*X1000 100E <CR>

i na ekranu se pojавio listing samo naseg programa.

Ako sada zelite disasembliратi dalje, dovoljno je da napisete samo:

*X <CR>

i disasembliranje ce se nastaviti od mjesta gdje smo prije toga stali.

Pomocu naredbe <X> mozete disasembliратi sadrzaj ROM-a. Basic ROM se nalazi od C000 do DFFF, a Monitor od E000 do FFFF.

4.4. <U> - IZVRSENJE STROJNOG POTPROGRAMA

Poslije unosa strojnog programa, potrebno je taj program i izvrsiti. Naredbom:

U nnnn

postavlja se programsko brojilo na zeljenu lokaciju "nnnn" (na pocetnu adresu naseg programa) i izvodjenje pocinje. Nije vazno da li se izvrsavaju programi iz RAM-a, ili potprogrami iz ROM-a.

Nas program za zbrajanje brojeva izvrsit cemo pomocu naredbe:

*U1000 <CR>
*-

Nije se dogodilo nista vidljivog na ekranu, ali mi znamo da je sad rezultat zbrajanja u lokaciji 0501. To cemo (kasnije) i provjeriti.

Pokusajte izvrsiti i neki potprogram iz ROM-a. Ako ne znate niti jednu adresu, probajte resetirati racunalo bez da pritisnete RESET taster. Napisite:

*UFF89 <CR>

i racunalo je resetirano.

4.5 <M>CITANJE I MIJENJANJE SADRZAJA MEMORIJSKE LOKACIJE

Pomocu monitorske naredbe:

M nnnn

mozemo u bilo kojem trenutku citati i mijenjati sadrzaj bilo koje memorijске lokacije (iz RAM-a).

Nakon maloprijasnjeg resetiranja, program za zbrajanje je i dalje u memoriji. Izlistajte ga pomocu naredbe:

*X1000 <CR>

Sada cemo provjeriti da li se rezultat zbrajanja nalazi u lokaciji 0501:

*M0501 <CR>
0501 0F_-

0F je trenutni sadrzaj lokacije 0501, a racunalo je spremno da prihvati i novi sadrzaj za tu lokaciju koji mozemo unijeti preko tastature. Sadrazaj lokacije 0501 necemo

mijenjati. U njoj se nalazi točan zbroj ($08+07=0F$). Vratite se u Monitor tako da pritisnete bilo koji znak koji ne pripada heksadecimalnoj notaciji. Pritisnite, na primjer:

*M0501

0501 0F Q

*

i ponovo ste u Monitoru. Sada ćemo umjesto $08+07$ zbrojiti $09+07$. Ovu sitnu izmjenu mogli bi napraviti na dva nacina. Prvi:

- iz miniasemblera ispravite liniju 1000:

*A1000

1000 LDA #09

1002 Q

*

Drugi nacin je taj da izmjenimo samo sadržaj lokacije u kojoj se nalazi broj 08, tako da u nju zapisemo 09:

*M1001

1001 08 09 <CR>

1002 8D Q <CR>

*

Disasemblirajte program i vidjet ćete da je program promijenjen. Mozete ga i izvršiti (U), pa provjerite i sadržaj lokacije 0501, gdje će biti novi rezultat.

4.6. <E> - CITANJE MEMORIJSKOG BLOKA

Ponekad zelimo brzi ispis i pregled većeg bloka memorije. To nam omogucava monitorska naredba:

E nnnn mmmm

gdje je "nnnn" početna, a "mmmm" završna adresa bloka.

Primjer:

```
*E E000 E030 <CR>
E000 00 00 00 00 00 00 00 00
E008 08 08 08 08 08 00 08 00
E010 14 14 00 00 00 00 00 00
E018 28 28 7C 28 7C 28 28 00
E020 10 78 14 38 50 3C 10 00
E028 46 26 10 08 64 62 00 00
E030 08
```

*

4.7. <H> - CITANJE MEMORIJSKOG BLOKA U ASCII OBLIKU

Monitorska naredba:

H nnnn mmmm

Vrlo je slična monitorskoj naredbi E. U ovom slučaju će ispisivati pet bajtova u jednom redu, s tim da će bajtovi najprije biti ispisani heksadecimalno, a iza toga (u istom redu) će biti ispisani i odgovarajući ASCII znakovi tih bajtova. "nnnn" je početna, a "mmmm" završna adresa bloka memorije kojeg ćemo na ovaj način pregledavati.

Primjer:

*H C100 C120 <CR>
C100 54 41 42 A8 54 TAB.T

Racunalo je tu stalo s ispisom. Čeka da pritisnemo bilo koji taster. Pritisnite razmaknicu i ispis će se nastaviti:

C105 CF 46 CE 53 50	.F.SP
C10A 43 A8 54 48 45	C.THE
C10F CE 4E 4F D4 53	.NO.S
C114 54 45 D0 AB AD	TE...
C119 AA AF E0 41 4E	...AN
C11E C4 4F D2	.O.

*—

Ovaj blok slučajno pripada Basic ROM-u, a u ovom dijelu se nalazi tabela Basic naredbi.

4.8. <F> - PUNJENJE MEMORIJSKOG BLOKA (Fill)

Prilikom strojnog programiranja, ponekad je potrebno brzo napuniti dio memorije s nekim brojem (konstantom). Pomocu naredbe:

F nnnn mmmm bb

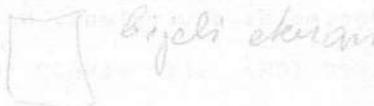
ćemo u svaku lokaciju dijela memorije s početnom adresom "nnnn" i završnom adresom "mmmm" staviti broj "bb".

Ovu naredbu ćete najlakše shvatiti na sljedećem primjeru:

- znamo da se VIDEO RAM nalazi od 6000 do 7FFF. Cijeli ovaj blok memorije napunit ćemo s FF, odnosno invertirati ćemo sadržaj ekrana. Napisite:

*F 6000 7FFF FF <CR>

*—



4.9. <C> - PROVJERA SUME

Pomocu naredbe:

C nnnn mmmm

mozemo izracunati kolika je zajednicka suma svih bajtova unutar memorijskog bloka s pocetnom adresom "nnnn" i završnom adresom "mmmm".

Za primjer cemo izracunati ukupnu sumu Basic ROM-a:

```
*C C000 DFFF <CR>
= 0F E7 09
*_
```

Ovu naredbu mozemo korisno upotrijebiti kod provjere sume bloka memorije ucitanog s kazete, da budemo sigurni da li je ucitavanje podataka proslo bez pogreske.

4.10. <Q> - KOPIRANJE DIJELA MEMORIJE

Dio memorije mozemo preseliti na drugo mjesto u memoriji uz pomoc naredbe:

Q nnnn oooo dddd

Ovom naredbom ce se na lokaciju "nnnn" iskopirati (preseliti) blok memorije s pocetnom adresom "oooo" i završnom adresom "dddd". Dakle:

nnnn - NA adresu
oooo - OD adrese
dddd - DO adrese

Primjer: u Video RAM cemo iskopirati sadrzaj jednog djela ROM-a (nema nikakvog smisla):

```
*Q 6000 E000 F000 <CR>
```

4.11. <#> - PRETVARANJE BROJEVA IZ HEKSADEC. U DECIMALNI

Pomocu naredbe:

#nnnn

cemo heksadecimalni broj "nnnn" na brzinu pretvoriti u njegov decimalni ekvivalent. Na primjer:

```
*#1000 <CR> ili *#0A32 <CR> ili *#00F7 i slicno.
```

5. ROM I UPOTREBA RUTINA IZ ROM-a

Cjelokupan rad mikroracunala Orao se zasniva na izvršavanju 16 krajnjih strojnog programa koji se nalazi u ROM-u. Strojni program u ROM-u podijeljen je u dvije cjeline. Jedna je BASIC interpreter, a druga MONITOR program. Obje cjeline odlikuju se velikom kompleksnoscu, a njihovo razumijevanje predstavlja put prema razumijevanju rada mikroracunala Orao i drugih racunala.

Postepenim upoznavanjem strojnog programiranja ćemo u poziciju da pisemo vlastite programe u strojnem jeziku. Međutim, postoji vjerojatnost da u mikroracunalu Orao već postoje potprogrami koji će biti potrebni u sklopu rješavanja naše problematike. Jasno je da bi pisanje postojecih programa bilo sasvim suvisno, pa će zbog toga u ovom dijelu priručnika biti navedeni neki najčešći koristeni potprogrami koji se uglavnom odnose na komuniciranje korisnika s racunalom.

5.1. RUTINE ISPISIVANJA

5.1.1. Rutina ispisivanja karaktera

5.1.2. Rutina ispisivanja stringa

5.1.3. Rutina ispisivanja numeričkih vrijednosti

5.1.1. RUTINA ISPISIVANJA KARAKTERA

Ovo je glavna rutina za ispisivanje u mikroracunalu Orao. Zadatak ovog potprograma je ispisivanje sadržaja akumulatora na ekran. Njegova primjenjivost je gotovo svugdje gdje se zeli ispisati neki alfanumerički znak na ekran.

Adresa ovog potprograma je FFF1. Prije nego što ga pozovemo, u akumulator trebamo staviti KOD karaktera kojeg zelimo ispisati.

Primjer:

Ispisat ćemo slovo "A" na ekran:

*A1000

1000 LDA #41

1002 JSR FFF1

1005 RTS

1006 Q

AU1000

*

Kad pokrenemo ovaj program (U 1000), na ekranu će se ispisati slovo "A".

5.1.2. RUTINA ISPISIVANJA STRINGA

Cesto se javlja potreba za ispisivanjem tekstualnog bloka na ekran. U mikroracunalu Orao postoji potprogram koji nam taj posao mnogo olaksava.

Adresa ove rutine je E63B. Prije nego sto ju pozovemo, potrebno je kreirati tabelu u kojoj će se nalaziti tekst koji cemo ispisivati. Prije poziva, NIZI dio adrese tabele teksta treba spremiti u Y registar, a VISI dio adrese tabele teksta u akumulator. Tabelu teksta organizirat cemo pomocu monitorske naredbe "M" uz napomenu da se na kraju tabele MORA nalaziti znak "04".

Primjer:

Ispisat cemo sadržaj tabele teksta koja se nalazi na adresi 1100. Program neka pocinje na adresi 1000.

Najprije cemo kreirati tabelu s tekstrom:

*M1100

```
1100 AA 4D  
1101 AA 49  
1102 AA 4B  
1103 AA 52  
1104 AA 4F  
1105 AA 52  
1106 AA 41  
1107 AA 5B  
1108 AA 55  
1109 AA 4E  
110A AA 41  
110B AA 4C  
110C AA 4F  
110D AA 20  
110E AA 4F  
110F AA 52  
1110 AA 41  
1111 AA 4F  
1112 AA 04  
1113 AA Q
```

*_

Sada cemo, koristeci rutinu za ispis stringa iz ROM-a, ispisati ovaj tekst na ekran. Program ce izgledati ovako:

*A1000

```
1000 LDY $00      - NIZI dio adrese tabele teksta  
1002 LDA $11      - VISI dio adrese tabele teksta  
1004 JSR E63B    - poziv rutine za ispis (iz ROM-a)  
1007 RTS
```

*_

*U1000

MIKRORACUNALO ORAO

?_

Rutina za pozicioniranje kursora

Kad bi htjeli ispisati neki tekst na točno određenoj poziciji ekrana (u Basicu to radimo pomocu naredbe za pozicioniranje kursora - CUR), morat ćemo se poslužiti još jednim gotovim potprogramom iz ROM-a. Pomocu nje ćemo, dakle, pozicionirati cursor na ekranu.

Adresa ove rutine je E39D. Prije nego što je pozovemo, u lokaciju E8 trebamo zapisati RED, a u lokaciju E9 KOLONU ekrana gdje zelimo ispisati neki tekst.

Slijedecim primjerom ćemo ispisati tekst iz prethodnog primjera u petom redu ekrana, počevši od nulte kolone:

```
*A2000  
2000 LDA #05  
2002 STA E8  
2004 LDA #00  
2006 STA E9  
2008 JSR E39D  
200B LDY #00  
200D LDA #11  
200F JSR E63B  
2012 RTS  
*—
```

5.1.3. RUTINA ISPISIVANJA NUMERICKIH VRIJEDNOSTI

Adresa ove rutine je E803, a ona ispisuje sadržaj akumulatora u heksadecimalnom obliku.

Primjer:

```
*A1000  
1000 LDA #41  
1002 JSR E803  
1005 RTS  
*U1000
```

Ako zelimo ispisati sadržaj akumulatora DECIMALNO, poslužit ćemo se rutinom na kojoj se bazira monitorska naredba "#". To je rutina na adresi EF57. Dakle, ako zelimo ispisati neki broj u decimalnom obliku, taj ćemo broj staviti u lokacije F6 (nizi dio) i F7 (visi dio) i pozvat ćemo rutine E882 i EFB6. Za primjer ćemo ispisati heksadecimalni broj FF u decimalnom obliku:

```
*A1000  
1000 LDA #00  
1002 STA F7  
1004 LDA #FF  
1006 STA F6  
1008 JSR E882  
100B JSR EFB6  
100E RTS
```

Kad ga izvedemo:

*U1000
255 — dečimalno
*— 126157071 SA X1000

napisat M1000

255

5.2. RUTINE ZA ISPITIVANJE TASTATURE

-
- 5.2.1. Potprogram za unos znaka s tastature
 - 5.2.2. Unos znaka s prikazom na ekranu

5.2.1. POTPROGRAM ZA UNOS ZNAKA S TASTATURE

Kada zelimo unijeti neki znak s tastature bez prikaza na ekranu, koristit ćemo potprogram na adresi E500. Njegova upotrebljivost je vrlo velika kada je izvodjenje programa vezano s dinamickim mijenjanjem parametara koje korisnik zeli unijeti izravno (npr. kod pisanja igara).

Opisat ćemo slijedeci primjer koji će ispitivati da li je pritisnut taster "A". Kad pritisnemo bilo koji znak osim "A", na ekranu će se ispisati znak "@" . Kad pritisnemo "A" program se prekida i vracamo se u Monitor:

```
*A1000
 1000 JSR E500
 1003 LDA FC
 1005 CMP #41
 1007 BEQ 1011
 1009 LDA #40
 100B JSR FFF1
 100E JMP 1000
 1011 CLC
 1012 RTS
 1013 Q
```

5.2.2. UNOS ZNAKA S PRIKAZOM NA EKRANU

Ovaj potprogram je gotovo isti kao i prethodno opisani. Razlika je samo u tome sto se u ovom slučaju znak koji se unese s tastature prikazuje na ekranu. Adresa ovog potprograma je E71C.

Primjer:

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

5.3. POTPROGRAMI ZA CRTANJE

-
- 5.3.1. Crtanje točke
 - 5.3.2. Crtanje linije
 - 5.3.3. Crtanje kružnice

5.3.1. POTPROGRAM ZA CRTANJE TOCKE (PLOT)

Adresa ovog potprograma je FE69. Prije nego sto ga pozovemo, potrebno je u lokacije E2 i E3 zapisati koordinate tocke:

E2 - koordinata X
E3 - koordinata Y

Primjer: nacrtat cemo tocku na sredini ekrana

*A1000

```
1000 LDA #80
1002 STA E2
1004 STA E4
1006 JSR FE69
1009 RTS
100A Q
```

Primjer: nacrtat cemo, tocku po tocku, dijagonalu ekrana

*A1000

```
1000 LDA #FF
1002 STA E2
1004 STA E3
1006 JSR FE69
1009 DEC E2
100B DEC E3
100D BNE 1006
100F RTS
1010 Q
```

5.3.2. POTPROGRAM ZA IZVLACENJE LINIJE (DRAW)

Ovaj potprogram pocinje na adresi FE8B. Za izvlacenje linije potrebno je odrediti koordinate pocetka i koordinate kraja linije, dakle, potrebne su nam jos cetiri pomocne memorijске lokacije:

E2 - X koordinata pocetne tocke linije
E3 - Y koordinata pocetne tocke linije
E4 - X koordinata zavrsne tocke linije
E5 - Y koordinata zavrsne tocke linije

Primjer: nacrtat cemo drugu dijagonalu ekrana

*A1000

```
1000 LDA #00
1002 STA E3
1004 STA E4
1006 LDA #FF
1008 STA E2
100A STA E5
100C JSR FE8B
100F RTS
```

5.3.3. POTPROGRAM ZA CRTANJE KRUZNICE (CIR)

Adresa ovog potprograma je FF06. Prije crtanja kruznice potrebno je odrediti koordinate sredista kruznice i njezin polumjer, a za to nam trebaju tri pomocne lokacije:

- E2 - X koordinata sredista kruznice
- E3 - Y koordinata sredista kruznice
- F8 - polumjer kruznice

Primjer: nacrtat cemo kruznici u sredistu ekrana

```
*A1000  
1000 LDA #80  
1002 STA E2  
1004 STA E3  
1006 LDA #50  
1008 STA F8  
100A JSR FF06  
100D RTS  
100E Q
```

X u 1000 (02) rečinjano i
dobjeće se koordinata
X RESEŠENJE SE PROGRAM
X MUF 89
uvrženo i u svih tipov priborom
ako naprimer *M1000 (02)

5.4. RUTINE SNIMANJA I UCITAVANJA

Ove rutine su također dostupne prilikom programiranja u strojnog jeziku i lako se koriste u vlastitim programima.

5.4.1. RUTINE SNIMANJA

5.4.1.1. SNIMANJE ZAGLAVLJA

5.4.1.2. SNIMANJE BLOKA MEMORIJE BEZ ZAGLAVLJA

5.4.1.1. Snimanje zaglavlja

ZAGLAVLJE ili HEADER je početni dio svakog programa. Sastoji se od samo nekoliko bajtova, a sadrži sve najvažnije podatke o programu, kao što su IME programa, POČETNA ADRESA programa, DUZINA programa i VRSTA programa (Basic, Object...).

Adresa ove rutine je F3E4. Njenim startanjem snima se dio memorije od 0280 do 037F, zato što se od adrese 0280 nalazi zaglavlj. Duzina pilot signala (TGAP u Basicu) nalazi se u lokaciji 024F.

Da bismo mogli snimiti zaglavlj, trebali bi znati kako ono izgleda. Evo kako izgledaju zaglavlj jednog Basic i jednog strojnog programa:

BASIC program

STROJNI program

SAVE "ime"	DMEM "ime",start,duzina
0280 1B	1B
0281 1B	kontrolni
0282 1B	bitovi
0283 1B	
0284 42 = B (Basic)	4F = O (Object)
0285 .	.
0286 .	.
0287 .	10
0288 .	.
0289 .	lokacija
028A .	.
028B .	za
028C .	.
028D .	IME PROGRAMA
028E .	.
028F 00	00
0290 xx	POCETNA ADRESA
0291 xx	programa
0292 yy	DUZINA
0293 yy	programa

Ako cete snimati programe u dijelovima (posebno zaglavje, posebno blok podataka) koristeci ove rutine, pazite kod snimanja zaglavja - pocetak INPUT BUFFERA MÖNITORA se nalazi takodjer na adresi 0280, a to bi vam moglo smetati.

5.4.1.2. Snimanje bloka memorije bez zaglavja

Pocetna adresa ovog potprogramma u ROM-u je ED80.
Medjutim, zgodnije ju je koristiti ako ne skacemo na sam pocetak te rutine, nego najprije namjestimo potrebne parametre (pocetnu adresu i duzinu bloka) i ubacimo se u tu rutinu od adrese ED99.

Prije poziva te rutine treba biti u:

F4, F5 - pocetna adresa bloka
FE, FF - duzina bloka (broj bajtova bloka)

Primjer: snimit cemo dio memorije od lokacije 1000 do lokacije 1500 (heksadecimalno)

```
*A1000
 1000 LDA #00
 1002 STA F4
 1004 STA FE      F4=00
 1006 LDA #10      F5=10 => POČETAK (1000)
 1008 STA F5      FE=00
 100A LDA #05      FF=05 => DUZINA (0500)
 100C STA FF
 100E INC FE      - OBAVEZNO
 1010 INC FF      - OBAVEZNO
 1012 LDA #20      - DUZINA PILOT SIGNALA
 1014 JMP ED99      - NA SNIMANJE
 1017 Q
```

5.4.2. RUTINE UCITAVANJA

5.4.2.1. UCITAVANJE ZAGLAVLJA

5.4.2.2. UCITAVANJE BLOKA MEMORIJE BEZ ZAGLAVLJA

5.4.2.1. Ucitavanje zaglavlja

Pocetna adresa ovog potprograma je F5F4.

Njenim izvrsenjem ucita se blok memorije od 0200-037F.

5.4.2.2. Ucitavanje bloka memorije bez zaglavlja

Pocetna adresa ovog potprograma u ROM-u je EE65.

Njenim pozivanjem bez namjestanja dodatnih parametara (samo pocetne adrese ucitavanja), podaci bi se poceli ucitavati na adresu 0400. Posto to nema nece uvijek odgovarati, namjestit cemo adresu NA KOJU cemo ucitati blok memorije i uskociti u ovaj potprogram na adresu EE69.

Y registar - NIZI dio adrese na koju cemo ucitavati
Akumulator - VISI dio adrese na koju cemo ucitavati

Primjer: blok memorije (1000-1500) koji smo malo prije snimili, ucitat cemo na adresu 2000

```
*A1000
 1000 LDY #00
 1002 LDA #20
 1004 JSR EE69
 1007 RTS
 1008 Q
```

5.5. ADRESE OSTALIH VAZNIJIH POTPROGRAMA IZ ROM-a

E000 - adresa grafickog karakter generatora
E4F4 - "BEEP" zvucni signal
E500 - ispitivanje KOJI je taster pritisnut
E5B0 - ispitivanje DA LI je taster pritisnut
E79F - PADDLE rutina
E7B7 - prazna petlja za kasnjenje
E7DC - ispis jednog praznog mesta na ekran
E7E4 - ispis cetiri prazna mesta (TAB 4)
E7F6 - CR+LF na ekran
E800 - ispis praznog mesta+sadrzaj akumulatora (heksa.)
E817 - ispis sadrzaja programskog brojila
FF89 - RESET sekvenca
E620 - adresa
E906 - adresa <U>
E90E - adresa <E>
EAD0 - adresa <C>
EB09 - adresa <Q>
EB48 - adresa <F>
EB6A - adresa <M>
EBAB - adresa <A>
EC80 - adresa <X>
EED0 - adresa <H>
EF57 - adresa <#>

5.6. GENERIRANJE ZVUKA

Princip generiranja zvuka iz miniasemblera je slijedeci:

```
*A1000
1000 LDX #10
1002 LDY #FF
1004 DEY
1005 BNE 1004
1007 BIT 8800 - nije vazan mnemonik, vazno je da se
100A DEX      adresira zvucnik (adresa zvucnika=8800)
100B BNE 1002
100D RTS
100E Q
```

U ovom primjeru vrlo lako mozemo kontrolirati i duzinu tona i visinu tona. Duzina tona odredjena je sadrzajem registra X. Cim je sadrzaj registra X manji, ton traje krace, i obrnuto. VISINA tona odredjena je sadrzajem Y registra. Cim je broj u Y registru veci, ton je dublji. Manji broj generira visi ton.

Evo nekoliko primjera razlicitih zvucnih efekata:

```
*A1000
1000 LDX #FF
1002 LDY #FF
1004 TYA
1005 STY 8800
1008 DEY
1009 BNE 1000
100B TAY
100C DEY
100D BNE 1004
100F DEX
1010 TXA
1011 TAY
1012 DEY
1013 BNE 1004
1015 RTS
```

Probajte mijenjati vrijednosti X i Y registra.

```
*A1000
1000 LDA #FF
1002 STA 20
1004 LDX #FE
1006 LDY 20
1008 DEY
1009 BNE 1008
100B BIT 8800
100E DEC 20
1010 DEX
1011 BNE 1006
1013 RTS
```

U sljedećem primjeru zvuk se prekida pritiskom na "A":

```
*A1000
1000 LDX #FF
1002 LDY #FF
1004 TYA
1005 STY 8800
1008 DEY
1009 BNE 1008
100B TAY
100C DEX
100D DEY
100E BNE 1004
1010 JSR E5B0 - da li je pritisnut koji taster ?
1013 BCC 1000 - ako nije idi na 1000
1015 JSR E500 - taster je pritisnut, ali koji ?
1018 CMP #41 - da li je pritisnuto "A" ?
101A BNE 1000 - ako nije idi na pocetak
101C CLC - pritisnuto je "A" i ide se na
101D RTS - kraj programa
```

Primjer: "sviranje" po tastaturi

*A1000

```

1000 JSR E5B0
1003 BCC 1000
1005 LDX FA
1007 DEX
1008 BNE 1007
100A BIT 8800
100D JMP 1000
1010 Q

```

X1000 za sviranje

Ako ste pomislili da Orao može svirati samo jednoglasno, prevarili ste se. Slijedeci primjer je princip generiranja DVOKANALNOG ZVUKA:

*A2000

```

2000 LDA $FF
2002 STA 20
2004 LDA #FE
2006 STA 21
2008 DEX
2009 BNE 2010
200B BIT 8800
200E LDX 20
2010 DEY
2011 BNE 2008
2013 BIT 8800
2016 LDY 21
2018 JMP 2008
201B Q

```

X1000 za sviranje

Ustvari, ne postoji nikakva prepreka da se generira i viseglasni svuk. Evo primjera TROKANALNOG zvuka, a vrlo slicno bi bilo i za vise kanala (malo teze bi bilo od toga sloziti akord, ali mogu se komponirati zaista efektne melodije s ritmovima, basom, pratnjom,...)

*A1000

```

1000 LDA #FF
1002 STA E2
1004 LDA #FD
1006 STA E3
1008 LDA #FB
100A STA E0
100C DEX
100D BNE 1014
100F BIT 8800
1012 LDX E2
1014 DEY
1015 BNE 101C
1017 BIT 8800
101A LDY E3
101C DEC E0
101E BNE 100C
1020 BIT 8800
1023 STA E0
1025 JMP 1000
1028 Q

```

5.7. ISPITIVANJE TASTATURE

Poznata vam je INKEY naredba Basic-a. Poznati su vam i nacini ispitivanja tastature i pritisnutosti nekog tastera pomocu potprograma iz ROM-a.

Kako god da ste ispitivali tastaturu dosad, ispitivali ste pritisnutost SAMO JEDNOG TASTERA u odredjenom trenutku.

Zamislite si da pisete akcione igru u kojoj je potrebno upravljati svemirskim brodom, pucati na napadace, ... Bilo bi divno da mozete upravljati brodom tako da u istom trenutku dajete gas i bacate bombe. Ili, da skrecete gore i u istom momentu pucate rafal na neprijatelja. U svakom od ova dva slucaja racunalo bi trebalo u istom trenutku prihvati oba dvije komande, a to znaci da bi u istom trenutku trebali pritisnuti DVA tastera, a racunalo bi to trebalo prihvati i izvrsiti obje komande. Pomocu standardnih naredbi i rutina to nije moguce izvesti. Ali, postoji rjesenje i za to.

Dobro proucite slijedecu "tabelu":

LDA 87FC	LDA 87FA	LDA 87F6	LDA 87EE
lijevo - Ex	PF1 - Ex	R - Ex	7 - 7x
gore - Dx	FF2 - Dx	Z - Dx	U - Bx
dole - Bx	FF3 - Bx	T - Bx	I - Dx
desno - 7x	PF4 - 7x	6 - 7x	O - Ex

LDA 87BE	LDA 877E	LDA 86FE	LDA 85FE
J - Dx	A - Dx	F - Ex	C - Ex
K - Bx	S - Bx	G - Bx	C - Dx
L - Ex	D - Ex	H - Dx	Z - Bx
M - 7x	Y - 7x	N - 7x	* - 7x

LDA 87DE	LDA 83FE	LDA 87FD	LDA 87FB
1 - 7x	P - Ex	CTL - Dx	SPC - Ex
Q - Dx	S - Bx	CR - Ex	SHIFT-Dx
W - Bx	D - Dx		
E - Ex	+ - 7x		

$$\begin{aligned} E &= 14 = 1110xxxx \\ \text{AND } 00010000 &= 10 \\ \hline &= 00000000 \end{aligned}$$

$$\begin{aligned} B &= 11 = 1011xxxx \\ \text{AND } 01000000 &= 40 \\ \hline &= 00000000 \end{aligned}$$

$$\begin{aligned} D &= 13 = 1101xxxx \\ \text{AND } 00100000 &= 20 \\ \hline &= 00000000 \end{aligned}$$

$$\begin{aligned} 7 &= 7 = 0111xxxx \\ \text{AND } 10000000 &= 80 \\ \hline &= 00000000 \end{aligned}$$

Ako bi htjeli ispitati da li je pritisnuto "A", pisali bi:

LDA 877E
AND #20

i ako je "A" bilo pritisnuto, u akumulatoru je nula (zbog AND #20). Nakon toga provjerom stanja akumulatora doznaćemo da li je "A" bilo pritisnuto ili ne.

U slijedećem primjeru ispitujemo tastere "Q" i "S". Ako nije pritisnut niti jedan taster, nista se ne dogadja, kao i onda kad pritisnemo bilo koji od njih. Tek kad pritisnemo DVA ZAJEDNO (i "Q" i "S"), na ekranu se ispisuje slovo "A" !!

```
*A2000
2000 LDA 87DE
2003 AND #20 - da li je pritisnuto "Q" ?
2005 BNE 2000 - ako nije, idi na pocetak
2007 LDA 877E - "Q" je, dakle, pritisnuto, a
2009 AND #40 - da li je pritisnuto i "S" ?
200C BNE 2000 - ako nije, idi na pocetak programa
200E LDA #41 - kod slova "A" u akumulator
2010 JSR FFF1 - ispisi "A" na ekran i
2013 JMP 2000 - idi na pocetak programa
2016 Q
```

Na ovaj nacin može se postaviti i uvjet da se neka akcija sprovodi samo u slučaju da su pritisnuta tri, cetiri ili više tastera zajedno, što se vrlo korisno može upotrijebiti u programu.

PRILOG A

Tabela YU (ASCII) kodova

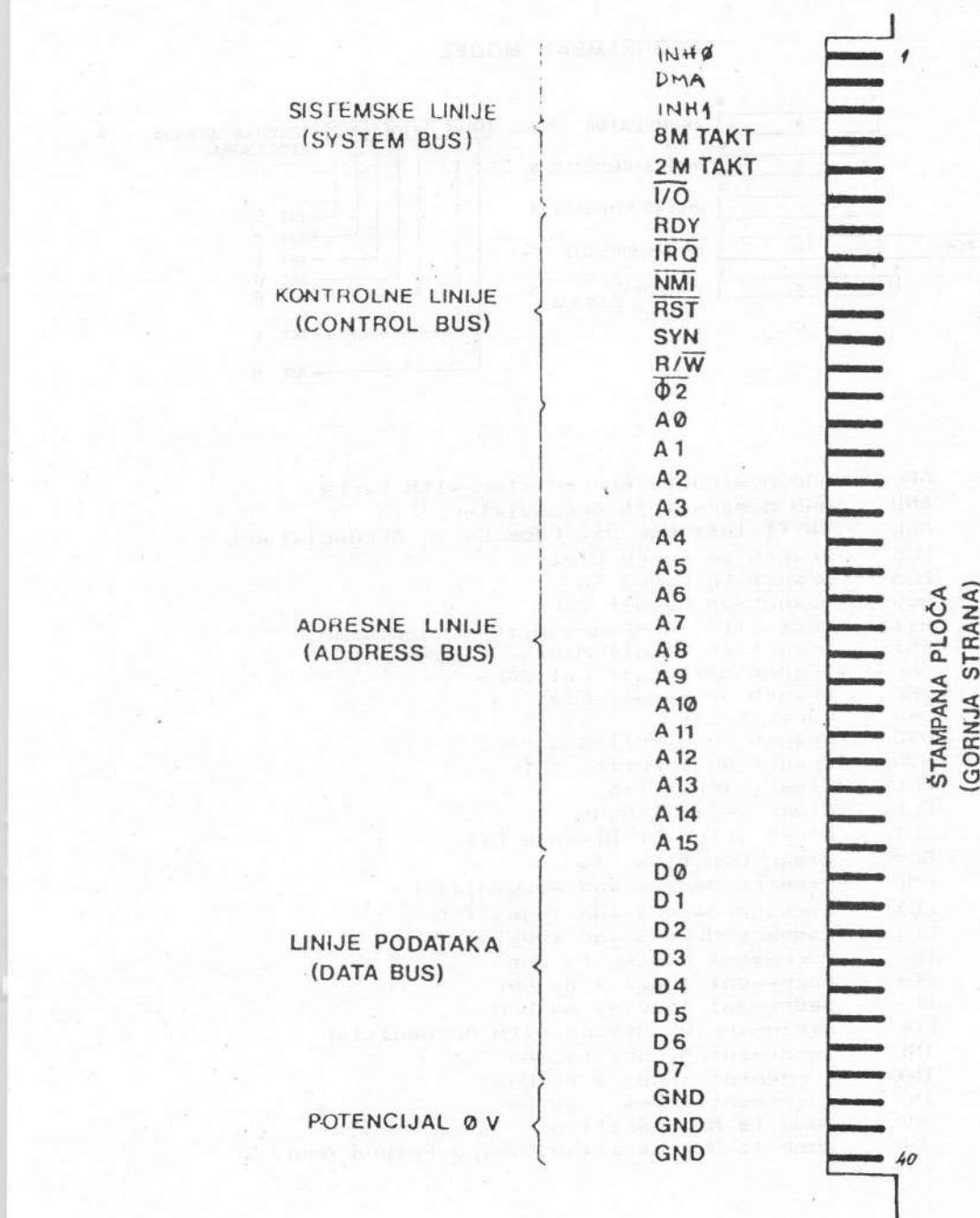
standardni set znakova

	0	16	32	48	64	80	96	112
MSB	0000	0001	0010	0011	0100	0101	0110	0111
LSB \	0	1	2	3	4	5	6	7
0000 0				Ø	@	P	^	P
0001 1			!	1	A	Q	a	q
0010 2		"	2	B	R	b	r	
0011 3	#	3	C	S	c	s		
0100 4	\$	4	D	T	d	t		
0101 5	%	5	E	U	e	u		
0110 6	&	6	F	V	f	v		
0111 7	'	7	G	W	g	w		
1000 8	(8	H	X	h	x		
1001 9)	9	I	Y	i	y		
1010 A	*	:	J	Z	j	z		
1011 B	+	,	K	C	k	c		
1100 C	.	<	L	C	l	c		
1101 D	-	=	M	D	m	d		
1110 E	.	>	N	S	n	s		
1111 F	/	?	O	Z	o	z		

KONTROLNI KODOVI

PRILOG B

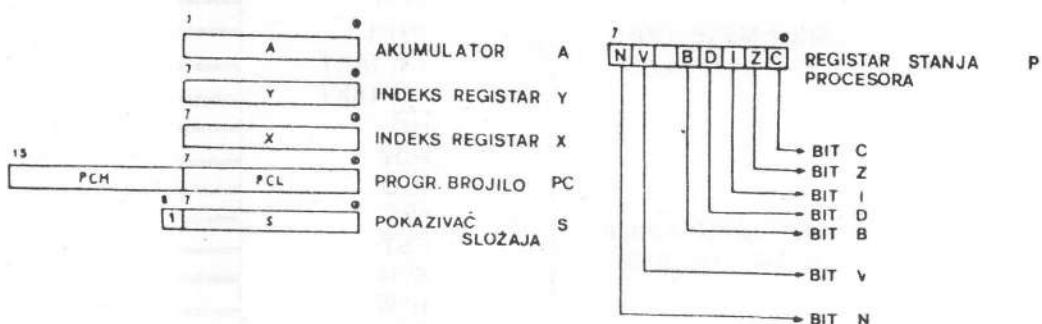
Konektor za proširenje



PRILOG C

Abecedni popis mnemonika za mikroprocesor 6502

PROGRAMSKI MODEL



AIC	Add Memory to Accumulator with Carry
AND	AND Memory with Accumulator
ASL	Shift left One Bit (Memory or Accumulator)
BCC	Branch on Carry Clear
BCS	Branch on Carry Set
BEQ	Branch on Result Zero
BIT	Test Bits in Memory with Accumulator
BMI	Branch on Result Minus
BNE	Branch on Result not Zero
BPL	Branch on Result Plus
BRK	Force Break
BVC	Branch on Overflow Clear
BVS	Branch on Overflow Set
CLC	Clear Carry Flag
CLD	Clear Decimal Mode
CLI	Clear Interrupt Disable Bit
CLV	Clear Overflow Flag
CMF	Compare Memory and Accumulator
CPX	Compare Memory and Index X
CPY	Compare Memory and Index Y
DEC	Decrement Memory by One
DEX	Decrement Index X by One
DEY	Decrement Index Y by One
EOR	Exclusive OR Memory with Accumulator
INC	Increment Memory by One
INX	Increment Index X by One
INY	Increment Index Y by One
JMP	Jump to New Location
JSR	Jump to New Location Saving Return Address

LIA	Load Accumulator with Memory
LIX	Load Index X with Memory
LDY	Load Index Y with Memory
LSR	Shift One Bit Right (Memory or Accumulator)
NOP	No Operation
ORA	OR Memory with Accumulator
PHA	Push Accumulator on Stack
PHP	Push Processor Status on Stack
PLA	Pull Accumulator from Stack
PLP	Pull Processor Status from Stack
ROL	Rotate One Bit Left (Memory or Accumulator)
ROR	Rotate One Bit Right (Memory or Accumulator)
RTI	Return from Interrupt
RTS	Return From Subroutine
SBC	Subtract Memory from Accumulator with Borrow
SEC	Set Carry Flag
SEI	Set Decimal Mode
SEI	Set Interrupt Disable Status
STA	Store Accumulator in Memory
STX	Store Index X in Memory
STY	Store Index Y in Memory
TAX	Transfer Accumulator to Index X
TAY	Transfer Accumulator to Index Y
TSX	Transfer Stack Pointer to Index X
TXA	Transfer Index X to Accumulator
TXS	Transfer Index X to Stack Pointer
TYA	Transfer Index Y to Accumulator

