STROJNI JEZIK ZA PROCESOR Z80

Jaro Lajovic

STROJNI JEZIK ZA PROCESOR Z80

s primeri za ZX Spectrum

Mladinska knjiga

Besedilo je avtor izpisal z urejevalnikom INES na računalniku Sinclair Spectrum.

KAZALO

Tabele

Mikroprocesorji so od začetka sedemdesetih let na neustavljivem pohodu. Postajajo vse zmogljivejši in vse bolj nepogrešljivi. Vse bolj nepogrešljiva postaja zato tudi pisana beseda o njih. Na našem knjižnem trgu imamo knjige, posvečene zgradbi in tehniki delovanja mikroprocesorjev. Nimamo pa del, ki bi seznanjala z mikroprocesorji iz zornega kota uporabnika - programerja.

Plima razmeroma cenenih hišnih mikroračunalnikov je zapljusnila tudi nas. Ti računalniki zmorejo precej več kot zgolj igrice. Vendar lahko iz njih "iztisnemo" kaj res uporabnega samo, če se podamo v strojni jezik. Ker je delo s takšnimi računalniki pravzaprav delo z njihovim mikroprocesorjem, naj citiram prof. Dušana Kodeka (Uvod v mikroprocesorske sisteme. Fakulteta za elektrotehniko, Ljubljana 1984): "...stanje se... seveda spreminja in obseg programiranja v višjih programskih jezikih narašča. Kljub temu lahko s precejšnjo gotovostjo trdimo, da bo programiranje v zbirnem, ali celo v strojnem jeziku, ki je pri večini (velikih) računalnikov danes že praktično izumrlo, še dolgo pomenilo enega od osnovnih načinov uporabe mikroprocesorjev." To je del odgovora na vprašanje čemu naj bi se uporabnik hišnega računalnika poglabljal v mikroprocesor oz. strojni jezik. Drugi, vsaj tako pomemben del odgovora pa je čar takšnega dela, ki je velik izziv znanju, iznajdljivosti in ustvarjalnosti.

Čas - v računalniškem svetu še posebej - neusmiljeno teče. Valu računalniškega navdušenja bodo le stežka sledili ustrezni plodovi, če ne bo literature - tudi domače. Zavedajoč se tega je bila pripravljena ta knjiga. Da bomo mogli z razvojem v korak, bo seveda potrebno storiti še marsikaj. Želel bi, da bi to delo ob tem ne ostalo osamljeno.

Za dragoceno pomoč, brez katere bi knjige ne bilo, se zahvaljujem Primožu Jakopinu. Za jezikovno skrb sem dolžan zahvalo Leonu Kreku, za razumevanje in pomoč pri izdaji knjige pa založbi Mladinska knjiga, tov. Cirilu Trčku in Jožetu Vilfanu.

Ljubljana, januar 1985

Jaro Lajovic

UVOD

Odločili ste se, da se boste natančneje seznanili s strojnim jezikom za mikroprocesor Z80. Pred vami je knjiga, ki vam bo pri tem pomagala. Služila naj bi tako začetniku kot že bolj izkušenemu uporabniku. Zato je zastavljena kot vodnik in priročnik obenem. Da bo sestava knjige jasnejša, orientacija v njej. pa lažja, dodajamo na začetku ta uvod (komur se neznansko mudi, naj preskoči na začetek naslednjega poglavja).

Programiranje, še posebej programiranje v strojnem jeziku, je nekoliko podobno vožnji z avtomobilom ali z motorjem. Kdor želi biti dober voznik, mora dobro poznati prometne predpise in mora (vsaj nekoliko) poznati zgradbo in delovanje vozila, koristno pa je tudi, če pozna ceste, po katerih vozi. Programerjevo "vozilo" je centralni procesor računalnika, "prometni predpisi" so ukazi strojnega jezika, "cesta" po kateri hiti, je njegov (mikro)računalnik s svojimi značilnostmi. Te trojne razdelitve se bomo oklenili v naši knjigi.

V prvih poglavjih se bomo seznanili s centralnim procesorjem in njegovimi (računskimi) sposobnostmi. Nato se bomo posvetili spoznavanju strojnega jezika za procesor Z80. Njegov nabor ukazov običajno razdelimo v nekaj večjih skupin, in sicer na ukaze za

- polnjenja,
- aritmetične operacije,
- logične operacije,
- skoke, klice podprogramov in povratke,
- skupinske operacije,
- delo z biti,
- pomike in
- vhodno/izhodne operacije.

Nekaterih manjših skupin ukazov tu nismo omenili, a v knjigi nanje seveda nismo pozabili. V smiselnem zaporedju, kakršnega smo uporabili, so nanizana tudi poglavja. Razdelitev je primerna tako zaradi vsebine kot zaradi zahtevnosti, saj vodi od enostavnega k bolj zapletenemu. Tretji del knjige smo namenili

"spoznavanju cest". Beseda bo podrobneje tekla o nekaterih pomembnejših uporabnih vprašanjih: pripravi in shranjevanju strojnih programov, izpisovanju na zaslon, rabi tipkovnice ter izvajanju zapletenejših izračunov. Končali bomo s posebej zanimivim poglavjem - programom za razgibano igro: "Ključar Martin in vražji metulj".

Zadnji del knjige je pripravljen za ZX Spectrum. To seveda ne pomeni, da bi morali uporabniki drugih hišnih računalnikov vreči puško v koruzo. Potrebno bo le malo več raziskovanja in tuhtanja, pa bodo manj zapletene tudi njihove "zanke in uganke".

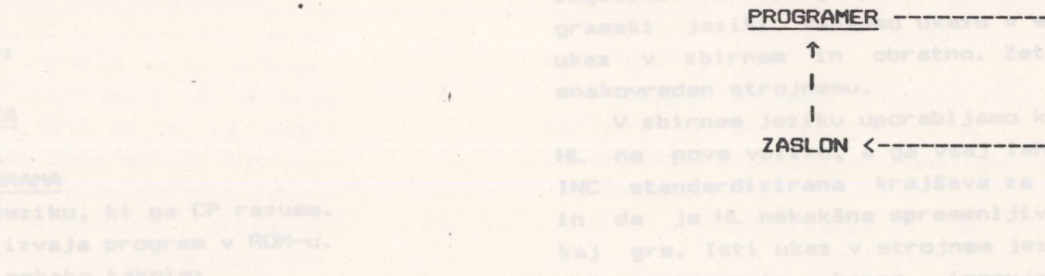
Star pregovor pravi: "Hrabrim sreča ne odreče.". Ker naš "vozni red" poznamo, se torej le hitro podajmo na pot.

ZAČETEK

Ta knjiga je uvod v programiranje v strojnem in zbirnem jeziku za računalnike z mikroprocesorjem Z80. Takšnih je kar nekaj: od Sinclairjevega Spectruma, ZX 81, Galaksije, Dialoga 20 Gorenje, Amstrada CPC 464 pa vse do večjih računalnikov, kakršen je Iskrin Partner. Knjiga je namenjena predvsem uporabnikom ZX Spectruma, vendar bo koristila tudi vsem drugim.

Kaj pravzaprav je strojni jezik ?

Najprej poglejmo preprosto shemo delovanja računalnika:



Shema kaže, da je med programerjem in centralnim procesorjem računalnika še nekaj. Programer pri običajnem programiranju ne more neposredno ukazati centralnemu procesorju (običajno ga označujemo kar CP), kaj naj stori.

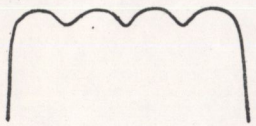
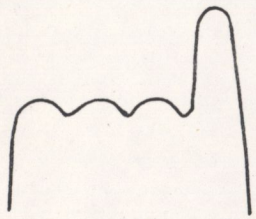
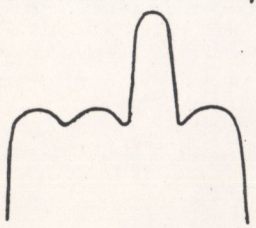
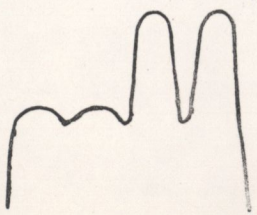
V mnogih računalnikih je centralni procesor vezje Z80. Prepričan sem, da ne boste prav nič presenečeni ob dejstvu, da vezje Z80 ne razume niti besede basica ! V resnici ni nobenega centralnega procesorja moč programirati z ukazi, ki bi bili neposredno razumljivi ljudem.

Če dobro premislite, boste ugotovili, da ne bi bilo v nobenem primeru mogoče dati CP-ju ukaza, ki bi vseboval za človeka kakšen smisel. Odprite pokrov vašega računalnika in poglejte vezje s 40 nožicami, označeno z "NEC" - to je centralni procesor Z80A. Takšno vezje lahko odgovarja le na električne impulze, ki do njega pritekajo z drugih delov. Ker Z80 sprejema signale z osmih nožic - tj. z osmih priključkov - ga prištevamo

KAKO PROCESOR ŠTEJE

V prejšnem poglavju smo omenili, da zna CP šteti z osmimi prsti do 255. Kako to zmore, ko pa mi z desetimi prsti lahko štejemo le do 10 ?

Vzrok je boljša razporeditev podatkov. Zakaj bi moral iztegnjeni prst kazalec predstavljati isto vrednost kot iztegnjeni mezinec ? Očitno lahko tako predstavimo dve različni števili, saj je tudi število 001 različno od števila 100. Ljudje pač nismo preveč učinkoviti pri štetju na prste. CP pa takšne stvari upošteva in tako že samo z dvema prstoma šteje od 0 do 3:

CP-jevi prsti	dvojiški zapis	opis
	00	vsi prsti skrčeni pomenijo število 0,
	01	iztegnjen prvi prst pomeni število 1,
	10	iztegnjen drugi prst pomeni 2,
	11	oba prsta pomenita 3 (2 + 1).

CP-jev prst je v resnici najmanjši del roke (se pravi registra ali pomnilniške celice) in ga imenujemo "bit". Vsak posamezni bit ima lahko samo dve vrednosti, 1 ali 0, zato imenujemo takšno zapisovanje števil dvojiško ali binarno.

Če bi dodali še tretji prst, bi lahko prikazali števila od 0 do 7. Samo s tremi prsti! Štirje prsti pa bi zmogli predstavljati vsa števila od 0 do 15. Da bi poenostavili pisanje in se izognili zmedi - npr. število 11 se mora razlikovati od dvojiškega zapisa 11 - je bil sprejet dogovor: števila od 10 do 15 označujemo s črkami od A do F.

desetiško 10 = A šestnajstiško (heksadecimalno)
11 = B
12 = C
13 = D
14 = E
15 = F

Preprosto, kajne ? To imenujemo šestnajstiški ali heksadecimalni zapis. Da bi preprečili vsako pomoto, pišejo nekateri za šestnajstiškimi števili črko H (ki v šestnajstiškem zapisu nima številčne vrednosti), npr. 15 = FH, 10 = AH.

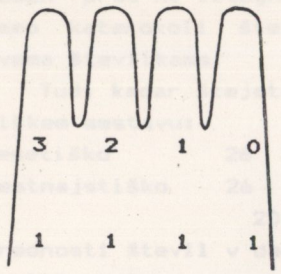
Pri delu s strojnim jezikom je raba šestnajstiškega zapisa zelo prikladna, ker
- tak zapis zlahka pretvorimo v dvojiškega in takoj vidimo, kateri bit (se pravi: kateri prst) ima vrednost 0 in kateri 1;
- zlahka vidimo ali gre za 8- ali 16-bitno število;
- omogoča standardiziran zapis vseh števil, ki jih obvlada CP, v obliki dvoštevilčnih vrednosti;
- je široko v rabi. Njegovo poznavanje vam bo olajšalo prebira- prebiranje drugih knjig in priročnikov.

Gre le za dogovor. Če bi želeli, bi lahko vse ukaze mirno zapisali v navadni desetiški obliki.

Preden nadaljujemo, se moramo seznaniti z novim pojmom: byte. To je angleška beseda, s katero označujemo skupino osmih bitov. Slovensko pravimo bytu zlog, nekateri pisci pa ga imenujejo tudi znak. V knjigi se bomo dosledno držali besede "zlog".

Šestnajstiški zapis omogoča, da s samo 4 biti predstavimo števila od 0 do 15. Katerokoli 8-bitno vrednost, se pravi vsak zlog (byte), lahko torej opišemo z dvema nizoma štirih bitov. To je zelo pomembno, ker ima večina mikroračunalnikov 8-bitne registre in 8-bitne pomnilniške celice - tako kot ima večina ljudi po pet prstov na rokah.

Poglejmo zdaj, kako bi šteli s štirimi prsti:



(prste smo tudi oštevilčili)

= 8 + 4 + 2 + 1 = 15 desetiško = FH

Ker so vsi prsti iztegnjeni, imajo vsi vrednost 1.

Če ste naklonjeni matematiki, ste najbrž opazili, da se vrednost, ki jo predstavlja posamezen prst, množi z 2, ko greste proti levi. Poglejte, kako smo oštevilčili prste: vrednost prsta je "2 na N-to potenco" (N je številka prsta). Roko s štirimi prsti kakršno smo uporabili, imenujemo "ročica".

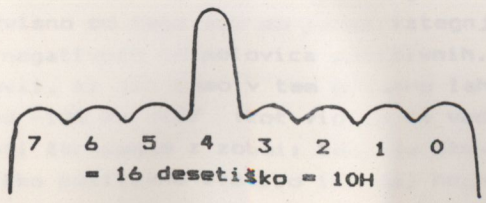
VAJA: Kakšne desetiške in šestnajstiške vrednosti predstavljajo tile nizi bitov ?

desetiško	šestnajstiško
0010	2
0110	6
1001	9
1010	10
1100	12

2	
6	
9	
A	
C	

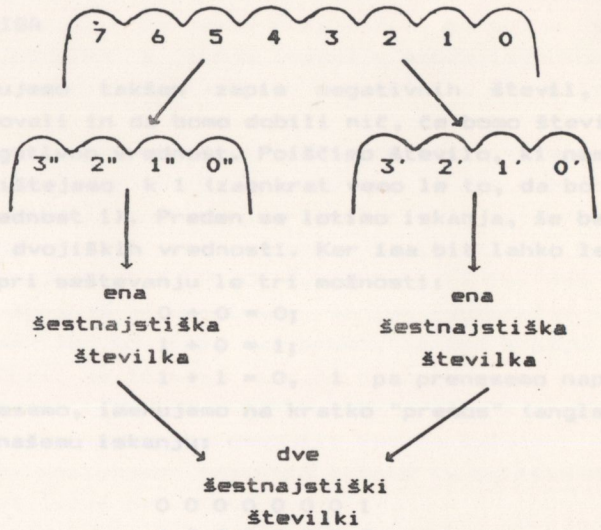
Pomembno je, da vam postane šestnajstiški zapis domač. Če imate težave z njim, preberite prejšnje strani še enkrat. Če ne, lahko nadaljujemo.

Kako prikažemo število, večje od 15 ? Na primer 16 ? Uporabiti moramo naslednji prst na levi:

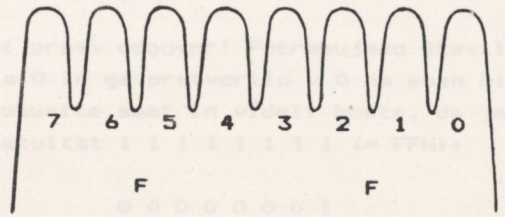


= 16 desetiško = 10H

Število pišemo 10H zaradi tega, ker razdelimo roko v dve 4-bitni ročici in označimo njuni vrednosti s šestnajstiškima številkama (0 - 9 in A - F). Omenili smo že, da lahko vsako 8-bitno roko opišemo z dvema 4-bitnima ročicama:



Ročica na levi označuje 16 krat večja števila kot ročica na desni. Podobno je v desetiškem zapisu, kjer je številka v stolpcu desetic vredna 10 krat več kot v stolpcu enic. Za pretvarjanje iz šestnajstiškega zapisa v desetiški pomnožimo številko na levi ročici s 16. Če uporabimo kar prejšnji primer: 10H = (1 * 16) + 0 = 16 desetiško. Tako lahko štejemo do 255 s samo osmimi prsti. Največje število dobimo, kadar so iztegnjeni vsi prsti:



FF = (F * 16) + F
= (15 * 16) + 15
= 255 desetiško

je, da predstavlja H višji del števila (angleško High = visok), L pa nižji del števila (Low = nizek). Enako velja za ostale pare. Tako ne pride do zmede. Če pogledate skico, boste videli, da vsebuje višji del števila v paru BC register B in v DE register D.

Nogi imata posebno ime: imenujemo ju indeksna registra, ker opravljata podobno delo kot indeks (kazalo) v knjigi.

ZASTAVICE

Para AF običajno ne obravnavamo kot registrski par. F namreč označuje register z zastavicami (Flag register; flag = zastavica). O tem bomo govorili kasneje.

ZAMENLJIVI REGISTRSKI NIZ

Zamislite si toge plastične rokavice, ki bi obdržale obliko še potem, ko bi jih slekli. Bile bi zelo uporabne. Če bi npr. iztegnili tri prste in slekli rokavice, bi staro število ostalo shranjeno. Seveda rokavice ne morejo računati, če v njih ni rok. Da bi mogli uporabiti shranjene podatke, bi morali spet zamenjati rokavice.

CP ima takšne rezervne delovne rokavice za vse svoje roke (za noge pa ne - kdo je še kdaj slišal o rokavicah za noge ?). Vsaka rokavica ustreza le eni roki, zato se jih ne da menjati - tudi vi ne oblečete leve rokavice na desno roko.

CP-jeve rezervne rokavice imenujemo zamenljivi registrski niz. Prikaz 18 od omenjenih 20 registrov je torej takšen:

```
[ A ][ F ] <---> [ A' ][ F' ]
[ B ][ C ] <---> [ B' ][ C' ]
[ D ][ E ] <---> [ D' ][ E' ]
[ H ][ L ] <---> [ H' ][ L' ]

        [ IX ]
        [ IY ]
```

Rokavice, ki jih ima CP oblečene, nosijo vedno isto ime kot roka, na kateri so. Rezervne pa vedno označujemo s črtico. Ukazi vplivajo le na oblečene rokavice. Čeprav torej označujemo zamenljive registre s črtico ob imenu, ukazi kot na primer LD A', 1 ne obstajajo. Edini ukazi, ki zadevajo zamenljivi registrski niz, so ukazi "zamenjaj rokavice!". Primer:


```
    ukazi                    opisi

LD A, (predal 1)        ;napolni A z vsebino predala 1
EX AF, A'F'             ;okrajšava za zamenjavo (EXCHANGE =
                         zamenjaj)
LD A, (predal 2)        ;napolni A z vsebino predala 2
EX AF, A'F'             ;še ena zamenjava
LD A, (predal 3)        ;napolni A z vsebino predala 3
```

Med izbranimi petimi ukazi ni nobenega, ki bi neposredno spreminjal vrednosti v zamenljivih registrih - ki pa smo jih nedvomno spremenili. Poskusite ugotoviti, kaj je v registru A po vsakem izvedenem ukazu! Vzemimo, da so vrednosti v predalih

```
(predal 1) = 1
(predal 2) = 2
(predal 3) = 3.
```

Ko se izvrši ukaz	je vrednost v A	in v A'
1	1	neznana
2	neznana	1
3	2	1
4	1	2
5	3	2

programa ne spremeni (tako kot v našem primeru), bo imela funkcija USR ob povratku vrednost začetnega naslova programa. Ker smo napisali (glej urejevalnik) PRINT USR 29000, smo dobili na ekranu izpisano 29000. Ta lastnost funkcije USR je zelo dragocena, ker z njo lahko nadziramo dogajanje v strojnem programu.

Poženite spet naš urejevalnik. Za začetni naslov spet izberite 29000 in vpišite koda 03 in C9. Rezultat izvajanja tega programa bo število, za 1 večje od začetnega naslova. Zakaj ? Kod 03 pomeni "povečaj BC". Ker smo BC povečali za ena, je bil temu ustrezen tudi rezultat!

Še nekaj podrobnosti o urejevalniku:

- prvi stolpec pod napisom VSEBINA kaže vsebino naslova, preden smo jo spreminjali, drugi pa potem;
- naenkrat lahko vnesete tudi več kodov (v naših primerih naenkrat 00C9 ali 03C9);
- vprašanje "SPREMEMBE ?" se pojavi vsakič, ko vnesete kod C9, a tudi na koncu strani (kot ena stran šteje 15 na ekranu izpisanih naslovov);
- če samo pritiskate ENTER, se vsebina zlogov ne spremeni.

POMEMBNO: če na vprašanje "NAPREJ ?" odgovorite z D, bo program zbrisal zaslon, kot prvi naslov nove strani pa se bo pokazal zadnji naslov prejšnje strani. Ne spreminjajte mu vsebine, ker tam že imate vrednost, ki ste jo želeli.

Če strojnega programa ne želite pognati, vpišite na vprašanje "START ZA USR ?" odgovor "STOP" (symbol shift in A).

RAZISKOVANJE: Poskusite z urejevalnikom preiskati razne naslove v pomnilniku in jim spreminjati vsebino. Kot rečeno: računalniku ne morete škodovati, v najslabšem primeru ga bo treba izključiti in ponovno vključiti. Izbrišite nato v vrstici 100 urejevalnika vse ukaze, razen prvega. Poženite program in odgovorite na vprašanje "START ?" z naslovom 0. Kaj se zgodi, če hočete spreminjati vsebino teh pomnilniških celic. Zakaj ?

POLNJENJE Z ENO ROKO

Centralni procesor Z80 je 8-bitni procesor. To pomeni, da najlaže obvlada 8-bitne podatke, v 8-bitnih registrih in 8-bitnih pomnilniških celicah. Z drugimi besedami: najlaže mu gre od rok delo, ki ga lahko opravi z osmimi prsti. Zato se bomo tudi mi najprej seznanili z "enorokimi" operacijami.

Podobno kot večina ljudi je tudi CP desničar. So opravila, ki jih zlahka zmore z desnico, medtem ko jim z ostalimi rokami ni kos. Njegova desnica je register A. To je edini register, kjer lahko opravlja zapletene operacije kot sta seštevanje in odštevanje. Vse, kar ima na desnici, pa lahko prenaša na ostale roke in obratno. Takšno prenašanje podatkov imenujejo računalniški vseznalci registrsko naslavljanje. Primer:

```
LD A, B
LD H, E
```

in podobno. Spomnite se, da "LD" pomeni "napolni", da "," pomeni "z" in da se ukazi berejo v enakem vrstnem redu kot vsak navaden stavek. Prvi primer bi torej prebrali "napolni A z B", drugega pa "napolni H z E".

Podatke lahko brez težav prenašate z roke na roko. Izjema je le register F. To je - kot smo že povedali - register z zastavicami, ki ne shranjuje števil v običajnem smislu. Z njim zamenjave niso mogoče. Sicer pa za prenašanje ni ovir. Celo na videz neumen ukaz "LD A, A" je dovoljen! Na kratko zapišemo ukaze za registrsko naslavljanje

```
LD r, r
```

pri čemer je r katerikoli 8-bitni register, z izjemo F.

Zdaj torej vemo, kako lahko premetavamo podatke iz roke v roko. Vendar nam to ne bi kaj prida pomagalo, če podatkov ne bi mogli od nekod prinesti. Pri tem nam pomaga naslednja skupina ukazov. Z njo lahko določimo, koliko naj CP pokaže na kateri roki. Na primer "pokaži 215 na roki D". Prepričan sem, da bi to že znali zapisati kot "LD D, D7" (D7H = 215). Ta način imenujemo sprotno naslavljanje, njegova splošna krajšava pa je

celico nn z A" - bo vpisal vrednost v celico na naslovu nn. Če bo nn naslov celice v ROM-u, bo ukaz seveda brez učinka.

Naslov nn, ki nastopa v teh ukazih, je neko določeno število in ne spremenljivka. Zato moramo že med pisanjem programa vedeti, kateri naslov bomo uporabili. Ukaza torej služita za prenašanje "spremenljivk" - podatkov iz pomnilnika v register A in iz registra A v naključni pomnilnik (RAM). Vzemimo za primer program - igro Pristajanje na Luni. V njem potrebujete podatke o hitrosti svoje vesoljske ladje, o njeni višini in količini goriva. Ob vnašanju programa boste morali določiti, katere pomnilniške celice bodo služile shranjevanju teh vrednosti. Ko bi pisali program, bi torej še lahko napisali: LD A, (gorivo). Pri vnašanju strojnega koda ali uporabi zbirnika pa bi morali nadomestiti "gorivo" z naslovom izbrane pomnilniške celice. Na primer:

30000 = hitrost
30001 = višina
30002 = gorivo

Šele tako bi lahko izvedli del programa, v katerem bi pogledali, koliko je še goriva, to količino zmanjšali in jo zopet spravili.

In kaj če ne vemo natančnega naslova celice, ki vsebuje podatek? Recimo, da lahko le izračunamo, kje bo podatek shranjen ? Ker vsak naslov zasede 16 bitov, moramo izračunano vrednost začasno shraniti ali v enega izmed registrskih parov (BC, DE, HL) ali v enega indeksnih registrov (IX, IY). Prvi način: naslov v registrskem paru, imenujemo registrsko posredno naslavljanje. Ukazi so:

ukaz		opis
LD r, (HL)	-	napolni register z vsebino celice na naslovu HL,
LD A, (BC)	-	napolni A z vsebino celice na naslovu BC ter
LD A, (DE)	-	napolni A z vsebino celice na naslovu DE.

Registrski par HL je prednostni par, podobno kot je register A prednostni enojni register. Kadar shranimo naslov v HL, lahko zato prenašamo podatke v katerikoli register, celo v H ali v L, čeprav izgleda čudno. Kadar pa uporabljamo BC ali DE, lahko podatke nalagamo le v register A. Tudi pri teh ukazih srečujemo somernost: na podoben način shranjujemo podatke v pomnilnik:


```
LD (HL), A
LD (BC), A
LD (DE), A.
```

Pri drugem načinu posrednega naslavljanja shranimo naslov v enega od indeksnih registrov. To imenujemo indeksno naslavljanje. Indeksna registra IX in IY lahko uporabimo kot kazalca za cele skupine podatkov. Na kratko zapišemo to takole

```
LD r, (IX +dis)
LD r, (IY +dis)
```

Znak r spet označuje katerikoli 8-bitni register (razen F), dis pa pomeni odmik od naslova, ki ga vsebujeta IX oziroma IY. Tako lahko shranimo npr. 1.podatek neke skupine (dis = 1), 10. podatek (dis = 10), 137. podatek (dis = 137) in tako naprej. Odmik dis je 8-bitno število, ki ne more biti spremenljivka, zato ga moramo določiti med programiranjem. To je pomanjkljivost teh ukazov in jih običajno rabimo za branje ali pisanje tabel ali seznamov podatkov. Na voljo sta tudi obratna niza ukazov

```
LD (IX +dis), r
LD (IY +dis), r
```

Indeksni način naslavljanja je nekoliko zapleten in zato manj pogosto rabljen.

Izmed vseh skupin so najhitrejši (4 - 7 T stanj) in najkrajši (1 zlog) ukazi registrskega ter registrskega posrednega naslavljanja. Ukazi ostalih dveh skupin so daljši (3 - 4 zloge) in precej počasnejši (16 - 20 T stanj).

Z80 omogoča sestavljanje nekaterih opisanih načinov naslavljanja, npr. sprotnega (tj., da določite število) in zunanjega (tj., da s pomočjo registrskega para določite naslov). Tako dobite - kdo bi si mislil? - sprotno zunanje naslavljanje. Naslov lahko žal določate le s parom HL. Kratki zapis ukaza je:

```
LD (HL), n.
```

Kljub omejitvi je ukaz zelo uporaben, ker je shranjevanje neposredno, brez uporabe 8-bitnega registra.

Podobno kombinacijo z indeksnima registroma imenujemo sprotno indeksno naslavljanje. To je manj pomembno, kratki zapis ukazov pa se glasi:

```
LD (IX +dis), n
LD (IY +dis), n.
```

UPORABA

Poskusimo zdaj v primerih uporabiti nekaj "LD" ukazov. Vemo, da je po povratku iz strojnega programa vrednost funkcije USR enaka vrednosti v registskem paru BC. Poskusite z naslednjim programom:

```
0E 00       LD C, 0         ;napolni C z 0
C9          RET
```

(Odslej bomo programe zapisovali na ta način, ki ga imenujemo "zbirna oblika" (angleško assembly format). Na levi je strojni kod v šestnajstiškem zapisu, v sredini so krajšave, na desni pa potrebne opombe. Z našim urejevalnikom boste tako programe vnašali brez težav.) Za začetni naslov izberite 30000. Program da registru C vrednost 0; kot veste, je začetna vrednost para BC enaka začetnemu naslovu programa. Kakšen rezultat torej pričakujete?

* 0
* 30000
* 29952

Poženite program. Ste izbrali pravi odgovor? Če si morda niste čisto na jasnem, zakaj je rezultat takšen, kakršen je, preberite še enkrat poglavje Kako računalniki štejejo.

Da bi vas natančneje seznanili z obravnavanimi ukazi in vam obenem pomagali premagati začetne težave, bomo to in vsa naslednja poglavja zaključili s kratkimi primeri. Toplo vam priporočamo, da jih preizkusite in jih čim pogosteje tudi sami dopolnjujete ali spreminjate. Le tako boste dobili uporabno znanje strojnega jezika.

Izbiro začetnega naslova v primerih prepuščamo vam. Začeli bomo s sprotnim naslavljanjem:


```
06 00       LD B, 0         ;napolni B z 0
0E XX       LD C, XX        ;XX nadomestite z raznimi vrednostmi
C9          RET             ;vrni se v basic
```

Spreminjati smete tudi vrednost v registru B. Ppskusite pred vsakim izvajanjem ugotoviti, kakšen bo rezultat.

Nadaljujmo s primerom zunanjega naslavljanja. Spreminjajte vrednost naslova (v primeru je enak 28672) - najbolj zanimive bodo vrednosti, manjše od 16384 (tj. naslovi v ROM-u). Vrednost v izbrani pomnilniški celici najprej naložimo v A (zunanje naslavljanje) in jo od tam prenesemo v C (registrsko naslavljanje):

```
06 00       LD B, 0
3A 00 70    LD A, (naslov) ;napolni A s (trenutno) vsebino
                           pomnilniške celice
4F          LD C, A
C9          RET
```

Končajmo s primerom za registrsko posredno naslavljanje. Spreminjajte tudi tu vrednost naslova.

```
06 00       LD B, 0
21 00 70    LD HL, naslov
4E          LD C, (HL)
C9          RET
```

Prepuščamo vam, da sami poiščete in raziščete primere, ki jih nismo napisali. Kar nekaj jih bo - začnete lahko že z LD (naslov), A, nadaljujete z LD (HL), C in tako naprej. V teh primerih si bo treba nekoliko več pomagati z basicom. Po izvajanju programa boste namreč šele z ukazom PEEK (npr. PEEK naslov) lahko ugotovili, če ste dosegli nameravano.

ŠTEVILA NA DVEH ROKAH

Videli smo, kako spreten je CP, kadar dela s števili na eni roki. Njegova sposobnost je tolikšna, da zmore opraviti zapletene račune na eni sami roki. Pridejo pa trenutki, ko samo 8-bitna števila ne zadostujejo. In če bi bili omejeni le na vrednosti med 0 in 255, bi bil naš računalnik zelo okorna priprava.

Najbolj potrebujemo 16-bitna števila za naslavljanje pomnilniških celic, se pravi prostorov v pomnilniku. Primer takšnega naslavljanja smo srečali že v prejšnjem poglavju, ko smo govorili o ukazih kot sta npr. LD A, (HL) ali LD A, (nn). Na zapleten in počasen način bi delo tovrstnih ukazov opravili tudi brez uporabe 16-bitnih števil. Na srečo pozna Z80 ukaze, ki (čeprav jih ni mnogo) omogočajo rabo 16-bitnih števil. V tem poglavju se bomo ukvarjali le z nalaganjem teh števil, kasneje pa bomo obravnavali 16-bitno aritmetiko.

DOLOČANJE NASLOVOV

Naslovi morajo biti vedno zapisani kot 16-bitna števila! Tudi kadar so med 0 in 255, jih ne morete določiti samo z osmimi biti. CP števil, ki nimajo dvakrat po osem bitov, ne upošteva kot naslov. Zato smo v kratkem zapisu npr. LD A,(nn) naslov označili "nn", in ne le "n".

SHRANJEVANJE 16-BITNIH ŠTEVIL V POMNILNIK

Ko smo se seznanjali s procesorjevimi registri, smo rekli, da je višji zlog 16-bitnega števila spravljen v registrskem paru prvi, nižji pa drugi (spomnite se: HL = High/Low = visoki/nizki). Pri nalaganju 16-bitnih števil v pomnilnik velja obraten dogovor kot pri uporabi registrskih parov: nizki zlog je v pomnilniku vedno spravljen prvi. Poglejmo na primeru, kako se v pomnilnik shrani vsebina para HL. Naj HL vsebuje število 258 (=0102H). Pomnilniški prostori so na začetku prazni:

POLNJENJE S 16-BITNIMI VREDNOSTMI

Krajšava	Zlogov	Čas stanj	Učinek na zastavice T	C	Z	PV	S	N	H
LD rr, nn	3	10		–	–	–	–	–	–
LD IX, nn	4	14		–	–	–	–	–	–
LD IY, nn	4	14		–	–	–	–	–	–
LD (nn), BC	4	20		–	–	–	–	–	–
LD (nn), DE	4	20		–	–	–	–	–	–
LD (nn), HL	3	16		–	–	–	–	–	–
LD (nn), IX	4	20		–	–	–	–	–	–
LD (nn), IY	4	20		–	–	–	–	–	–
LD BC, (nn)	4	20		–	–	–	–	–	–
LD DE, (nn)	4	20		–	–	–	–	–	–
LD HL, (nn)	3	16		–	–	–	–	–	–
LD IX, (nn)	4	20		–	–	–	–	–	–
LD IY, (nn)	4	20		–	–	–	–	–	–

Oznake:

rr = 16-bitni register
nn = 16-bitno število (naslov)
* = zastavica se ob operaciji spremeni
0 = zastavica se spusti (postane 0)
1 = zastavica se dvigne (postane 1)
– = zastavica ostane, kot je bila

Procesor Z80A: 7 T stanj = 2 mikrosekundi

H = 01, L = 02

naslov	vsebina		naslov	vsebina
30000	00		30000	02
30001	00	---LD (30000), HL---->	30001	01
30002	00		30002	00

V pomnilniku (in izpisih programov) je torej nizki zlog zapisan prvi. Za to spremembo ni prave razlage ampak se moramo z njo kar sprijazniti.

Prosim, prepričajte se, ali vam je zamenjava načinov popolnoma jasna. Zelo verjetno je, da bo to edini zelo pogosti vir napak v vaših programih. V registrskih parih je prvi spravljen visoki zlog, v pomnilniku in izpisih pa je prvi spravljen nizki zlog. Tega nikakor ne bi smeli lahkomiselno preskočiti. Vsakič, ko boste imeli v strojnem kodu opravka s 16-bitnimi števili, boste morali pazljivo razmisliti o zaporedju visokih in nizkih zlogov.

A ne dajte se prestrašiti - življenje z Z80 bi bilo brez 16-bitnih ukazov skoraj nemogoče. In to je cena, ki jo moramo plačati.

NALAGANJE 16-BITNIH ŠTEVIL

Najpreprostejši ukaz v tej skupini je polnjenje registrskega para s 16-bitnim številom; splošna krajšava zanj je

```
LD rr, nn.
```

Spet uporabljamo zapis z dvema črkama za označevanje 16-bitnih vrednosti. Znak rr pomeni katerikoli registrski par, nn pa katerokoli število.

Če nimate zbirnika - in boste prevajali krajšave v strojni kod s pomočjo tabel na koncu knjige - potem postaja pravilo o zaporedju zlogov za vas zelo pomembno. Celo če imáte zbirnik, bi se morali dobro zavedati spremembe zapisa. Le tako boste brez težav prebirali zbirnikov izpis ali "brali" strojni kod s pomočjo ukaza PEEK v basicu.

Poglejmo primer "napolni HL z 258". Krajšava za to je LD HL, 0102H. Kod ukaza LD HL, nn je 21 XX XX. Namesto XX XX moramo vstaviti število - v našem primeru 0102H. Vendar tega ne

zapišemo 21 01 02, ampak je pravilna oblika 21 02 01. V zbirni obliki bo napisano:

```
21 02 01    LD HL, 0102H.
```

Z vnašanjem programov iz te knjige torej ne boste imeli težav. Vseeno vam mora postati to domače, da boste lastne programe pisali brez težav.

Prav tako kot v registrske pare naložimo 16-bitna števila v indeksna registra (ki sta oba nogi s po 16 prsti kot se spominjate):

```
LD IX, nn
LD IY, nn
```

Podatke lahko prenašamo tudi iz registrskih parov v dve zaporedni pomnilniški celici in obratno. Splošne krajšave so

```
LD (nn), rr          LD rr, (nn)
LD (nn), IX          LD IX, (nn)
LD (nn), IY          LD IY, (nn)
```

Kot vedno pomenijo oklepaji "vsebino". Zadnji ukaz bi npr. prebrali "napolni pomnilniško celico nn z vrednostjo registra IY".

Pravkar omenjeni ukazi vedno vplivajo na dve 8-bitni pomnilniški celici (delamo s 16 prsti!). Vzemimo za primer ukaz LD BC, (nn). Predstavljamo si lahko, da je sestavljen iz ukazov

```
LD C, (nn) ter
LD B, (nn+1)
```

(v resnici takšna ukaza ne obstajata). Enako velja za somerne ukaze LD (nn), rr. Vsi delujejo na dve celici. Vendar zadostuje, da navedemo samo naslov "nn", ker "nn+1" CP izračuna sam. Paziti pa je treba, da ne bi zamenjali ukazov za delo z 8- in 16-bitnimi števili.

POZOR! Pogosto ti ukazi programerja zapeljejo, da skuša v registrski par naložiti vsebino ene same pomnilniške celice! To je lahko zahrbten vir napak v programih.

UPORABA: Udomačite si te ukaze z naslednjimi programi. Prvi bo prebiral vrednosti parov pomnilniških zlogov:

```
ED 4B XX XX   LD BC, (nn)  ;namesto XX XX uporabite različne
                             vrednosti
C9            RET
```

Z drugim programom boste pisali v pomnilnik. Tudi tu preizkušajte različne vrednosti:

```
01 XX XX      LD BC, nn
ED 43 XX XX   LD (nn), BC  ;v tej vrstici priporočamo števila
                             med 4000H in 5B00H (za začetek pa
                             med 5800H in 5AA0)
C9            RET
```

Razlika med 8- in 16- bitnimi ukazi vam bo bolj jasna, če uporabite še tale programček:

```
3E XX         LD A, n
32 XX XX      LD (nn), A   ;svetujemo isti obseg naslovov kot v
                            prejšnjem programu
C9            RET
```


ZASTAVICE IN NJIHOVA RABA

Zastavice so tiste lepe stvari, ki zaplapolajo ob državnih praznikih ... Napačno!

V strojnem jeziku je "zastavica" v resnici "kazalec" - je nekaj, kar označuje določeno stanje. Podobno kot na ladjah: tam izobesite zastavico, da bi označili nesrečo, državno pripadnost, gusarstvo ali še kaj drugega.

Zastavice dajejo programerju informacijo o stanju na CP-jevi desnici (= v registru A) ali o ravnokar opravljeni računski operaciji. Ukazi polnjenja, o katerih smo že govorili, so eni redkih, ki ne vplivajo na nobeno zastavico. Ukazi, s katerimi se bomo seznanili v naslednjih poglavjih, pa nanje vplivajo. Zato se moramo pred tem nekoliko pomuditi pri zastavicah in njihovi rabi.

VRSTE ZASTAVIC

Zastavice centralnega procesorja so spravljene v registru F. To je navaden 8-biten register. Ker ima vsaka zastavica lahko le dve vrednosti, 1 ali 0, lahko je le "dvignjena" ali "spuščena", je za vsako zastavico potreben samo en bit. Register F bi torej mogel vsebovati osem zastavic, a jih ima le sedem:

Z	ničelna zastavica	(Zero flag)
S	zastavica znaka	(Sign flag)
C	zastavica prenosa	(Carry flag)
P/V	zastavica parnosti/prepolnjenja	(Parity/Overflow flag)
N	zastavica odštevanja	(Subtract flag)
H	zastavica polprenosa	(Half carry flag)

Ker eden od bitov rabi za dve zastavici (za zastavico parnosti in prepolnjenja) je programerju dosegljivih le šest bitov. Štirje od teh (ničelna zastavica, zastavica znaka, prenosa ter parnosti/prepolnjenja) so tako imenovane "pomembnejše" zastavice. Preostali dve sta manj pomembni in le redko uporabljeni.

Oglejmo si zdaj vsako zastavico posebej.

NIČELNA ZASTAVICA

Njeno delovanje je najlaže razumljivo. Dvigne se, kadar je rezultat (računske) operacije enak nič. Mnogo odločitev je odvisnih od stanja ničelne zastavice. Vendar je treba biti previden. Vseeno se namreč lahko zgodi, da je v registru vrednost nič, ničelna zastavica pa je spuščena. Primer je ukaz LD A, 0. Nobeden od polnilnih ukazov namreč nima učinka na zastavice. Sicer pa na ničelno zastavico delujejo vsi ukazi prištevanja, odštevanja in povečevanja 8-bitnih registrov ter seštevanja in odštevanja (s prenosom) 16-bitnih registrov. Prav tako delujejo nanjo logični ukazi, krožni in drugi pomiki ter ukazi testiranja bitov. Pri slednjih je ničelna zastavica sploh edini rezultat. Dvigne se, če je vrednost testiranega bita nič.

ZASTAVICA ZNAKA

Je zelo podobna ničelni zastavici. Nanjo vplivajo v glavnem isti ukazi kot na ničelno zastavico (glavna razlika je skupina testiranja bitov - vrednost bita pač ne more biti negativna). Zastavica znaka se dvigne, če je rezultat operacije negativen.

ZASTAVICA PRENOSA

Gre za zelo pomembno zastavico. Brez nje bi bili rezultati računanja v strojnem jeziku povsem brez pomena.

Naj vas spomnim, da se ukazi v strojnem jeziku vedno nanašajo ali na 8- ali na 16-bitna števila. Opravka imamo torej s števili med 0 in 255 ali med 0 in 65535. Zaradi omejenega obsega se vrednosti v registrih ali pomnilniku pri računskih operacijah obnašajo "krožno". V 8-bitnem registru bomo ob odštevanju

```
   200
 - 201
```

dobili rezultat 255 !

Enako velja za 16-bitne vrednosti. V takšnih primerih se dvigne zastavica prenosa.

Seveda vpliva na zastavico prenosa tudi prištevanje. Zato jo včasih obravnavamo kot deveti bit registra A:

	prenos	število v dvojiški obliki
132	0	1 0 0 0 0 1 0 0
+ 135	0	1 0 0 0 0 1 1 1
= 267	1	0 0 0 0 1 0 1 1

Ker ima register A le osem bitov, bo po tem seštevanju v njem število 11, zastavica prenosa pa bo dvignjena (tj. = 1). "Izposojanje" iz 9. bita bi tudi pri odštevanju pustilo v njem vrednost 1, se pravi dvignjeno zastavico.

Vrednost vseh ostalih zastavic se spreminja le posredno, kot posledica drugih operacij. Edino na zastavico prenosa lahko vplivamo tudi neposredno, s posebnima ukazoma. To sta
SCF - dvigni zastavico prenosa (SET CARRY FLAG) in
CCF - preklopi zastavico prenosa (COMPLEMENT CARRY FLAG).
Ukaz SCF dvigne zastavico prenosa, CCF pa ji spremeni vrednost: če je 1, postane 0 in obratno.

ZASTAVICA PARNOSTI/PREPOLNJENJA

Kot zastavico parnosti jo uporabljamo pri logičnih, kot zastavico prepolnjenja pa pri aritmetičnih operacijah. Možnost pomote je zelo majhna, saj se obe vrsti ukazov izjemno redko pojavljata skupaj.

Zastavica parnosti se pri logičnih operacijah dvigne, kadar je v rezultatu parno število bitov.

Zastavica prepolnjenja je svarilni pripomoček, ki opozori, da se je zaradi opravljene operacije spremenil osmi bit registra. To pomeni, da rezultat računanja morda (ni pa nujno) ne bo več ustrezal v osem bitov. V primeru našega seštevanja (132 + 135), je bil osmi bit pred seštevanjem 1, po njem pa 0, zato bi se zastavica prepolnjenja dvignila. Vendar pa bi se dvignila tudi v primeru seštevanja

```
   64      0 1 0 0  0 0 0 0
 + 65      0 1 0 0  0 0 0 1
 = 129     1 0 0 0  0 0 0 1
```

"IF ... THEN ... " V STROJNEM JEZIKU

V basicu smo navajeni takšnih ukazov

IF A = 0 THEN GO TO ...

V strojnem jeziku namesto "IF A = 0" pogledamo ničelno zastavico. Če je dvignjena, vemo, da je A = 0.

Zastavice, o katerih smo ravnokar govorili, so edine, ki omogočajo vejanje programa oz. izbiro ukaza, ki naj se izvrši naslednji. Oblika pogojnega ukaza je "JP pogoj, naslov". JP je krajšava za "skoči" (angleško JUMP). Ukaz torej preberemo "skoči - če je izpolnjen pogoj - na naslov". Pogoj pa je lahko:

Z	(-> Zero)	dvignjena ničelna zastavica
NZ	(-> Not Zero)	spuščena ničelna zastavica
C	(-> Carry)	dvignjena zastavica prenosa
NC	(-> Not Carry)	spuščena zastavica prenosa
M	(-> Minus)	dvignjena zastavica znaka
P	(-> Positive)	spuščena zastavica znaka
PE	(-> Parity Even)	dvignjena zastavica parnosti
PO	(-> Parity Odd)	spuščena zastavica parnosti

Preostali dve zastavici sta le redko v rabi. To sta zastavica odštevanja (ki se dvigne , če je bila zadnja operacija odštevanje) in zastavica polprenosa (ki se dviga podobno kot zastavica prenosa, a le v primeru sprememb v petem bitu registra A).

POVEČEVANJE IN ZMANJŠEVANJE

Spoznali smo že, kako CP nalaga števila na svoje roke in noge. Raziščimo zdaj še preprosto možnost spreminjanja števil, ki so na prstih. Ta števila lahko povečujemo (tj. jim prištevamo 1) ali zmanjšujemo (tj. jim odštevamo 1). To je sicer zelo preprosto računstvo, je pa le več kot zgolj nalaganje. Povečevanje lahko že s pridom uporabljamo npr. pri štetju prometa ali popisu prebivalstva.

POVEČEVANJE

Povečujemo lahko števila na katerikoli roki - se pravi: v kateremkoli 8-bitnem registru. Splošna krajšava za to je

```
INC r.
```

Ukaz preberemo "povečaj vrednost registra" (angl. INCREASE = povečaj). Prav tako lahko povečujemo števila v vseh registrskih parih in 16-bitnih registrih:

```
INC rr
INC IX
INC IY
INC SP
```

Znak rr označuje registrski par (BC, DE ali HL). Spomnite se, kako preprosto ločimo operacije z 8- in 16-bitnimi števili: 8-bitna števila označujemo z eno črko, 16-bitna označujemo z dvema črkama.

Ukazi povečevanja so še močnejši, kot se zdi na prvi pogled. Povečujejo lahko tudi števila v pomnilniku, le s pomočjo indeksnih registrov ali para HL moramo določiti naslov pomnilniške celice:

```
INC (IX +dis)
INC (IY +dis)
INC (HL)
```

UKAZI POVEČEVANJA IN ZMANJŠEVANJA

Krajšava	Zlogov	Čas stanj	Učinek na zastavice T	C	Z	PV	S	N	H
INC r	1	4	-	*	*	*	0	*	
INC rr	1	6	-	-	-	-	-	-	
INC IX	2	10	-	-	-	-	-	-	
INC IY	2	10	-	-	-	-	-	-	
INC (HL)	1	11	-	*	*	*	0	*	
INC (IX +dis)	3	23	-	*	*	*	0	*	
INC (IY +dis)	3	23	-	*	*	*	0	*	
DEC r	1	4	-	*	*	*	1	*	
DEC rr	1	6	-	-	-	-	-	-	
DEC IX	2	10	-	-	-	-	-	-	
DEC IY	2	10	-	-	-	-	-	-	
DEC (HL)	1	11	-	*	*	*	1	*	
DEC (IX +dis)	3	23	-	*	*	*	1	*	
DEC (IY +dis)	3	23	-	*	*	*	1	*	

Oznake:

r = 8-bitni register
rr = 16-bitni register
* = zastavica se ob operaciji spremeni
0 = zastavica se spusti (postane 0)
1 = zastavica se dvigne (postane 1)
\- = zastavica ostane, kot je bila

Procesor Z80A: 7 T stanj = 2 mikrosekundi

Pomembno opozorilo: Prikličite si v spomin naš dogovor o rabi oklepajev: oklepaji ---> vsebina !
To je zelo pomembno, kajti npr. ukaza

INC HL in
INC (HL)

sta si zelo podobna. Vendar je njun učinek popolnoma različen. Prvi pomeni "povečaj HL", drugi pa "povečaj vsebino pomnilniške celice, ki ima naslov HL" (slednje navadno preberemo kar "povečaj vsebino HL"). Dokler boste imeli to pred očmi, boste varni pred pomotami. Poglejmo še, kako ukaza delujeta. Naj ima HL vrednost 29000. Primer:

INC HL — Poglej HL. Povečaj število na njegovih prstih za 1. Rezultat: 29001.
INC (HL) — Poglej HL. Poišči pomnilniško celico z naslovom 29000. Povečaj število v njej za 1. Rezultat: (29000) + 1.

ZMANJŠEVANJE

Somerni ukazi uporabljajo iste operande kot ukazi povečevanja:

DEC r	DEC (HL)
DEC rr	DEC (IX +dis)
DEC IX	DEC (IY +dis)
DEC IY	DEC SP

DEC pomeni "zmanjšaj" (angleško DECREASE). Tudi pri teh ukazih je treba biti pozoren na rabo oklepajev.

Ukazi povečevanja in zmanjševanja 8-bitnih števil vplivajo na vse zastavice, razen na zastavico prenosa. Osvežimo zato na hitro delovanje zastavic:

- ničelna zastavica (Z): bo dvignjena (= 1), če bo rezultat 0;
- zastavica prepolnjenja (PV): bo dvignjena, če se bo najvišji bit 8-bitnega števila spremenil;
- zastavica znaka (S): bo dvignjena, če bo najvišji bit 8-bitnega rezultata enak 1;

- zastavica polprenosa (H): bo dvignjena, če se bo zaradi operacije spremenil peti bit (tj. bit 4) števila;
- zastavica odštevanja(N): bo spuščena (= 0) po ukazih INC in dvignjena (= 1) po ukazih DEC;
- zastavica prenosa (C): nanjo povečevanje in zmanjševanje ne vpliva. Sicer pa se dvigne, kadar pride do prenosa iz najvišjega bita registra.

Ukazi povečevanja in zmanjševanja 16-bitnih števil ne vplivajo na zastavice.

UPORABA: Poskusite sami napisati program, s katerim boste v par BC naložili število med 0 in 65535 (tj. neko 16-bitno število) in nato uporabite ukaza za povečevanje in zmanjševanje!
Primerjajte zdaj vaš program z našim:

```
01 XX XX    LD BC, XX XX    ;ne pozabite na pravo zaporedje !
03          INC BC
C9          RET
```

Za zmanjševanje nadomestite drugo vrstico z "0B DEC BC".

RAČUNANJE NA ENI ROKI

Vse operacije, o katerih bomo zdajle spregovorili, uporabljajo zgolj 8-bitna števila in jih lahko opravljamo le na CP-jevi desnici, v registru A.

Kaže, da le desnica ve, kako seštevati in odštevati! S tem so prežete celo krajšave, tako da v njih izpuščamo oznako registra A. Če bi hoteli prišteti npr. B k A, bi pričakovali ukaz

ADD A, B

(ADD = prištej). V resnici pa je krajšava le

ADD B.

(Povedati moramo, da nekateri zbirniki zahtevajo zapis ukaza v razširjeni obliki, se pravi na primer ADD A, B).

Kljub tej omejitvi je strojni jezik še vedno zelo vsestranski, saj lahko k številu na desnici prištejemo marsikaj:

ADD r	prištej k A katerikoli 8-bitni register
ADD n	prištej k A 8-bitno število
ADD (HL)	prištej k A vsebino pomnilniške celice na naslovu HL
ADD (IX + dis)	prištej k A vsebino pomnilniške celice na naslovu IX + dis
ADD (IY + dis)	prištej k A vsebino pomnilniške celice na naslovu IY + dis.

To pomeni že pravo bogastvo možnosti. Številu v registru A lahko prištejemo katerokoli 8-bitno število, katerikoli 8-bitni register (razen F) in vsebino katerekoli pomnilniške celice (seveda moramo v zadnjem primeru prej določiti naslov). Vrednost v operandu (tj. v r, v (HL) itn.) se ob operaciji ne spremeni. Najbrž pogrešate ukaz

ADD (nn).

Takšnega ukaza ni, zato ga moramo sestaviti

LD HL, nn
ADD (HL).

Ker lahko na ta način naslov spreminjamo med izvajanjem programa, je to po svoje še ugodneje. Register HL je tu spet prednosten. S paroma BC in DE naslova ne moremo določiti.

UKAZI 8-BITNE ARITMETIKE

Krajšava	Zlogov	Čas stanj	T	C	Učinek na zastavice Z	PV	S	N	H
ADD r	1	4	*	*	*	*	*	0	*
ADD n	2	7	*	*	*	*	*	0	*
ADD (HL)	1	7	*	*	*	*	*	0	*
ADD (IX +dis)	3	19	*	*	*	*	*	0	*
ADD (IY +dis)	3	19	*	*	*	*	*	0	*
ADC r	1	4	*	*	*	*	*	0	*
ADC n	2	7	*	*	*	*	*	0	*
ADC (HL)	1	7	*	*	*	*	*	0	*
ADC (IX +dis)	3	19	*	*	*	*	*	0	*
ADC (IY +dis)	3	19	*	*	*	*	*	0	*
SUB r	1	4	*	*	*	*	*	1	*
SUB n	2	7	*	*	*	*	*	1	*
SUB (HL)	1	7	*	*	*	*	*	1	*
SUB (IX +dis)	3	19	*	*	*	*	*	1	*
SUB (IY +dis)	3	19	*	*	*	*	*	1	*
SBC r	1	4	*	*	*	*	*	1	*
SBC n	2	7	*	*	*	*	*	1	*
SBC (HL)	1	7	*	*	*	*	*	1	*
SBC (IX +dis)	3	19	*	*	*	*	*	1	*
SBC (IY +dis)	3	19	*	*	*	*	*	1	*
CP r	1	4	*	*	*	*	*	1	*
CP n	2	7	*	*	*	*	*	1	*
CP (HL)	1	7	*	*	*	*	*	1	*
CP (IX +dis)	3	19	*	*	*	*	*	1	*
CP (IY +dis)	3	19	*	*	*	*	*	1	*

Oznake:
r = 8-bitni register
n = 8-bitno število
* = zastavica se ob operaciji spremeni
0 = zastavica se spusti (postane 0)
1 = zastavica se dvigne (postane 1)
- = zastavica ostane, kot je bila
Procesor Z80A: 7 T stanj = 2 mikrosekundi

Razen prve omejitve - uporabljamo lahko le register A imamo še drugo: s temi ukazi so na voljo le 8-bitna števila. Ukaza:

```
LD A, 80H
ADD 81H
```

bosta dala skupaj rezultat 1 v registru A. Seveda bo dvignjena (= 1) zastavica prenosa kazala, da je bil rezultat prevelik. Zaradi tega je zelo koristen ukaz ADC (ADD WITH CARRY = prištej s prenosom). Je povsem enak ukazu ADD, le da prišteje še zastavico prenosa. To nam s pomočjo verižanja operacij omogoča seštevanje števil, večjih od 255. Seštejmo na ta način npr. 1000 (= 03E8H) in 2000 (= 07D0H) in spravimo rezultat v BC:

```
3E E8       LD A, E8H          ;nižji del prvega števila
C6 D0       ADD D0H            ;nižji del drugega števila
4F          LD C, A            ;spravi rezultat v C
3E 03       LD A, 03H          ;višji del prvega števila
CE 07       ADC 07H            ;višji del drugega števila
47          LD B, A            ;spravi rezultat v B.
```

Po prvem seštevanju (E8 + D0) bo zastavica prenosa dvignjena (ker je rezultat večji od FF), A pa bo vseboval B8 (preverite še sami!). Drugo seštevanje (3 + 7) ne bo dalo 0AH (= 10), kot bi dejali na prvi pogled, temveč 0BH (= 11) zaradi prenosa. Končni rezultat je zato 0BB8H (= 3000)! Verižno povezovanje sicer ni preveč praktično (kasneje bomo tudi videli, da imamo na voljo še druge ukaze), vendar ga lahko nadaljujemo in tako seštevamo večja števila.

Odštevanje 8-bitnih vrednosti je povsem enako seštevanju. Obstajata dva niza ukazov, eden za enostavno odštevanje in drugi za seštevanje s prenosom:

SUB i - odštej i (= SUBTRACT)
SBC i - odštej s prenosom i (SUBTRACT WITH CARRY)

"i" označuje iste operande kakor jih ima ukaz ADD. Vrednost operanda ostane nespremenjena.

PRIMERJANJE DVEH 8-BITNIH ŠTEVIL

Pustimo za trenutek strojni jezik in poglejmo, kako primerjamo dve števili (za primerjanje sta vedno potrebni dve števili. CP primerja število s tem, kar ima v registru A).

Za števili, ki predstavljata isto vrednost, vemo, da sta enaki. Če bi to hoteli izraziti računsko, bi rekli, da je razlika med takima številoma nič. Kaj pa, če je število, ki ga primerjamo, večje od tistega, s katerim ga primerjamo ? V tem primeru bo rezultat po odštevanju negativen. Podobno je, če je število, ki ga primerjamo, manjše. V tem primeru bo razlika pozitivna.

S temi ugotovitvami lahko izdelamo način za primerjanje v strojnem jeziku. Vse, kar potrebujemo, so zastavice in operacija odštevanja. Recimo, da želimo primerjati neko število s 5:

```
LD A, 5    ;število, s katerim primerjamo
SUB n      ;število, ki ga primerjamo.
```

Dobili bomo takšne izide:

```
n = 5  --->  ničelna zastavica 1, zastavica prenosa 0;
n < 5  --->  ničelna zastavica 0, zastavica prenosa 0;
n > 5  --->  ničelna zastavica 0, zastavica prenosa 1.
```

Test za enakost bo torej ničelna zastavica. Če bo število, ki ga primerjamo, večje, bo dvignjena zastavica prenosa. Če bo število manjše od vrednosti registra A, bosta obe zastavici spuščeni. Vendar bi bil ta način precej neuporaben, ker se pri odštevanju spremeni vsebina registra A.

Na srečo imamo na voljo ukaze

```
CP i
```

(COMPARE = primerjaj). Operandi so isti kot za seštevanje. Ukaz "primerjaj" je enak ukazu "odštej", le da ne spreminja vsebine registra A. Edini učinek ima torej na zastavice. Primerjalni ukazi so zelo praktični in se zelo pogosto uporabljajo.
PRESKUS: V naslednjih vrsticah uporabite namesto XX različne vrednosti in tako preskusite ukaze ADD, ADC, SUB in SBC.

ADD

```
00          NOP
3E XX       LD A, XX
06 XX       LD B, XX
80          ADD B
06 00       LD B, 00
4F          LD C, A
C9          RET
```


Za preskus ukaza

```
ADC         zamenjajte prvo vrstico s   37    SCF
            in četrto vrstico z         88    ADC B,
SUB         zamenjajte četrto vrstico z 90    SUB B,
SBC         zamenjajte prvo vrstico s   37    SCF
            in četrto vrstico z         98    SBC B.
```

Poskusite pri odštevanju program še razširiti. Namesto zadnje vrstice (RET) nadaljujte

```
3E 00       LD A, 00
CE 00       ADC 00
32 00 7E    LD (7E00H), A
C9          RET
```

Tako boste prenesli vrednost zastavice prenosa najprej v register A, od tam pa v pomnilniško celico 7E00H (= 32256). Po izvajanju strojnega programa boste lahko pogledali, kakšna je bila vrednost zastavice prenosa, in sicer z ukazom PRINT PEEK 32256.

RAČUNANJE Z DVEMA ROKAMA

Poglavju bi se bolj podal naslov "Računanje na dve pesti". Kljub prednostim, ki jih daje raba 16-bitnih števil, je aritmetika na dveh rokah precej okorna. Na voljo imamo namreč le malo ukazov.

Podobno kot lahko z 8-bitnimi števili računamo le v registru A, lahko s 16-bitnimi vrednostmi računamo le v registrskem paru HL. Zato imenujemo par HL tudi prednostni par 16-bitne aritmetike. Prednost pa ni tako izrazita kot pri registru A, zato imena HL v ukazih ne izpuščamo.

SEŠTEVANJE

Ukazi so le štirje:

```
ADD HL, BC
ADD HL, DE
ADD HL, HL
ADD HL, SP
```

Kot vidite, k HL ne moremo prištevati absolutnih števil - tj. ukaza ADD HL, nn ni. Napraviti moramo takole

```
LD DE, nn
ADD HL, DE .
```

Ker takšno zaporedje zasede štiri od sedmih 8-bitnih registrov, ni kaj dosti v rabi. Prav tako ni seštevanja med HL in indeksnima registroma. Ker tudi ukazov za neposreden prenos IX oz. IY v BC oz. DE ni, je treba za takšno seštevanje uporabiti sklad (z ukazom PUSH porinemo vrednost registra na sklad, s POP pa jo snamemo)

```
PUSH IX
POP DE
ADD HL, DE
```

Opazili ste ukaz ADD HL, SP. SP je poseben 16-bitni register - kazalec sklada. Doslej se z njim še nismo srečali, natančneje pa ga boste spoznali v naslednjem poglavju. Povejmo le, da so ukazi za 16-bitno aritmetiko eni redkih, ki upoštevajo SP kot običajen register.

UKAZI 16-BITNE ARITMETIKE

Krajšava	Zlogov	Čas stanj T	Učinek na zastavice C	Z	PV	S	N	H
ADD HL, rr	1	11	*	-	-	-	0	?
ADC HL, rr	1	11	*	-	-	-	0	?
ADD HL, SP	2	11	*	-	-	-	0	?
ADC HL, SP	2	15	*	*	*	*	0	?
ADD IX, BC ali DE	2	15	*	-	-	-	0	?
ADD IX, IX	2	15	*	-	-	-	0	?
ADD IX, SP	2	15	*	-	-	-	0	?
ADD IY, BC ali DE	2	15	*	-	-	-	0	?
ADD IY, IY	2	15	*	-	-	-	0	?
ADD IY, SP	2	15	*	-	-	-	0	?
SBC HL, rr	2	15	*	*	*	*	1	?
SBC HL, SP	2	15	*	*	*	*	1	?

Oznake:

rr = registrski par
* = zastavica se ob operaciji spremeni
0 = zastavica se spusti (postane 0)
1 = zastavica se dvigne (postane 1)
\- = zastavica ostane, kot je bila
? = učinek ni znan

Procesor Z80A: 7 T stanj = 2 mikrosekundi

Tudi s 16-biti imamo na voljo le omejen obseg števil (čeprav večjega kot z 8-biti). Zato lahko seštevanje, podobno kot pri 8-bitnem računanju, verižno povezujemo. Ukaz "prištej s prenosom" - krajšava je ADC (ADD WITH CARRY) - je zelo podoben ukazu ADD in uporablja iste registrske pare:

```
ADC HL, BC
ADC HL, DE
ADC HL, HL
ADC HL, SP .
```

Računanje z indeksnima registroma je omejeno le na seštevanje brez prenosa. Uporabljamo lahko le štiri registre oz. registrske pare:

```
ADD IX, BC          ADD IY, BC
ADD IX, DE          ADD IY, DE
ADD IX, IX          ADD IY, IY
ADD IX, SP          ADD IY, SP .
```

ODŠTEVANJE

Tudi 16-bitno odštevanje je enostavna operacija, vendar je možno le odštevanje s prenosom. Kadar ne veste natančno ali ima zastavica prenosa vrednost 1 ali 0, je zato najboljše pred 16-bitno odštevanje postaviti ukaz, ki bo spustil zastavico (najpogosteje logični ukaz AND A). Ukazi odštevanja s prenosom (SUBTRACT WITH CARRY) so:

```
SBC HL, BC
SBC HL, DE
SBC HL, HL
SBC HL, SP .
```

Če pogledate v priročnik za Spectrum, boste videli, da je na naslovih 23635 in 23636 shranjen naslov začetka programa v basicu, na 23627 in 23628 pa naslov njegovega konca. To bomo uporabili v naslednjem programu (v katerega je vloženega že kar precej pridobljenega znanja):


```
2A 4B 5C     LD HL, (23627)  ;naloži v HL naslov konca programa
ED 5B 53 5C  LD DE, (23635)  ;naloži v DE naslov začetka
ED 52        SBC HL, DE      ;odštej DE od HL
44           LD B, H         ;prenesi dobljeno vrednost v
4D           LD C, L          registrski par BC
C9           RET             ;vrni se
```

Že slutite, kaj izračuna ta program? Kot rezultat boste dobili dolžino (v zlogih) programa v basicu (ob prvem izvajanju bo to kar dolžina urejevalnika). Ta podatek pa je pogosto dragocen!

UČINEK NA ZASTAVICE

V 16-bitni aritmetiki ima zastavica prenosa precejšnjo veljavo. Razen manj pomembne zastavice odštevanja je namreč edina, na katero vpliva ukaz ADD. Zastavica bo dvignjena le, če operacija povzroči prenos iz registra H (prenos iz L se avtomatično prišteje k H med izvajanjem ukaza).

Operaciji ADC in SBC prev tako vplivata na zastavico prenosa, poleg tega pa še na ničelno zastavico, zastavico prepolnjenja in zastavico znaka.

RAVNANJE S SKLADOM

Na začetku knjige smo govorili o skladu. Predstavili smo ga kot kup, kamor CP odlaga lističe s podatki. Seveda je to le prispodoba. V resnici je sklad nekaj zlogov obsegajoče področje pomnilnika, kjer CP shranjuje informacije, ne da bi si mu bilo potrebno zapomniti njihov naslov. Ta naslov je shranjen v posebnem 16-bitnem registru, ki ga imenujemo kazalec sklada, označujemo pa SP (angleško Stack pointer). Kazalec sklada vedno vsebuje naslov "vrhnje" vrednosti na skladu.

Na sklad lahko porinemo in snamemo le 16-bitne vrednosti, ker je namenjen zlasti začasnemu shranjevanju naslovov. Splošni ukazi, s katerimi porinemo informacijo na sklad, imajo obliko

```
PUSH rr
PUSH IX
PUSH IY ,
```

splošni ukazi, s katerimi jo vzamemo s sklada pa

```
POP rr
POP IX
POP IY .
```

To sta zelo preprosta niza ukazov in ne potrebujeta nobenega naslova. Čeprav vedno porinemo na sklad *vrednost* registra ali registrskega para, navadno na kratko rečemo, da smo porinili na sklad register ali registrski par.

Za registrske pare so ti ukazi dolgi po en zlog, za indeksna registra pa po dva zloga. Tako so tudi izredno ekonomični v izrabi pomnilniškega prostora. Ukazi PUSH niso rušilni: register ali registrski par po ukazu PUSH še vedno vsebuje isto vrednost.

Ker lahko porinemo in snamemo katerikoli registrski par, je lahko par, ki ga snamete, drug kot par, ki ste ga porinili. Na primer

```
PUSH BC
POP HL.
```


UKAZI ZA RAVNANJE S SKLADOM

Krajšava	Zlogov	Čas stanj T	Učinek na zastavice C	Z	PV	S	N	H
PUSH, rr	1	11	-	-	-	-	-	-
PUSH IX ali IY	2	15	-	-	-	-	-	-
POP rr	1	10	-	-	-	-	-	-
POP IX ali IY	2	14	-	-	-	-	-	-
LD SP, nn	3	10	-	-	-	-	-	-
LD SP, (nn)	3	20	-	-	-	-	-	-
LD SP, HL	1	6	-	-	-	-	-	-
LD SP, IX ali IY	2	10	-	-	-	-	-	-
EX (SP), HL	1	19	-	-	-	-	-	-
EX (SP), IX	2	23	-	-	-	-	-	-
EX (SP), IY	2	23	-	-	-	-	-	-

Oznake:

rr = registrski par
nn = 16-bitno število
* = zastavica se ob operaciji spremeni
0 = zastavica se spusti (postane 0)
1 = zastavica se dvigne (postane 1)
\- = zastavica ostane, kot je bila

Procesor Z80A: 7 T stanj = 2 mikrosekundi

S tema ukazoma vrednosti para BC ne spremenimo, par HL pa dobi vrednost, ki jo je imel BC tedaj, ko smo ga porinili na sklad. Tako pridobimo, čeprav posredno, še ukaz

LD rr, rr₁

"napolni registrski par z vrednostjo nekega drugega registrskega para". Ker porabimo za to le dva zloga, je ta način zelo uporaben in priljubljen.

Posebnost ukazov PUSH in POP je, da lahko porinemo in snamemo tudi registrski par AF. To sta dva od redkih ukazov, ki upoštevajo AF kot registrski par. Vsekakor je to smiselno, saj pogosto želimo ohraniti stanje zastavic.

V pravilnem programu mora biti končni seštevek PUSH-ev in POP-ov enak, ne glede na to, po kateri poti teče program. Napake navadno vodijo v polom.

V Spectrumu se začenja sklad dva zloga pod vrhom naključnega pomnilnika (naslov vrha označujemo RAMTOP). Če ste v basicu uporabili ukaz CLEAR nn, se sklad spušča od naslova nn-2 navzdol. Izbrišite v vrstici 100 našega urejevalnika vse ukaze razen prvega. Napravite zdaj npr. CLEAR 32000 in si s pomočjo urejevalnika oglejte sklad.

Ne spreminjajte vsebine sklada!

Skoraj vsaka sprememba bo povzročila polom in računalnik boste morali izključiti in ponovno vključiti. Nadzorni program namreč hrani na skladu mnogo pomembnih podatkov in spremembe povzročijo razpad. Med drugim spravi CP ob klicu podprograma na sklad naslov, na katerega se vrne ob izteku podprograma. To lahko izkoristimo in z ukazi PUSH ter POP spreminjamo naslov za povratek. Vendar mora biti sprememba dobro premišljena, sicer se bo za program končala katastrofalno. Z naslednjim programom lahko pogledate, na kateri naslov v nadzornem programu se CP vrne iz našega strojnega programa:

```
C1      POP BC
C5      PUSH BC
C9      RET
```


KAZALEC SKLADA

Ko smo govorili o registrih, kazalca sklada nismo omenili. Ta register ima namreč med 16-bitnimi registri posebno mesto, podobno kot register F med 8-bitnimi. Rabi izključno shranjevanju naslova sklada. Zaradi svojega posebnega pomena je skoraj "nedotakljiv". Kljub temu ste lahko že zasledili ukaze, ki vključujejo SP. Eden takšnih je bil na primer SBC HL, SP. Skušajte ga zdaj, ko poznate tudi ukaze PUSH in POP uporabiti za reševanje zanimivega problema.

OREH (nekoliko trši!): V sistemski spremenljivki E_LINE (naslov 23641 in 23642) je spravljen naslov konca programa v basicu. Ker je sklad v pomnilniku Spectruma vedno tik pod vrhom naključnega pomnilnika, nam da razlika med HL in SP število nezasedenih zlogov pomnilnika. Poskusite napisati program, katerega rezultat bo to število! Primerjajte svojo rešitev s tisto na koncu poglavja.

Čeprav je spreminjanje vrednosti kazalca sklada tvegano (zlasti če si človek ni povsem na jasnem, kaj počenja), včasih le želimo spremeniti položaj sklada v pomnilniku. V ta namen imamo pet ukazov:

```
LD SP, nn
LD SP, (nn)
LD SP, HL
LD SP, IX
LD SP, IY
```

Omenimo še tri redko rabljene ukaze, ki pa včasih pridejo prav:

```
EX (SP), HL
EX (SP), IX
EX (SP), IY
```

Ti ukazi ne vplivajo na kazalec sklada ampak izmenjajo vrednost HL (oz. IX ali IY) ter vrednost na skladu. Pravzaprav so to ukazi, ki v nekaterih primerih nadomestijo ukaze PUSH in POP. Denimo, da je na skladu ena vrednost, v paru HL druga in ju želimo izmenjati. Lahko bi naredili tako

```
POP BC      ;shrani vrednost s sklada začasno v BC
PUSH HL     ;porini vrednost HL na sklad
PUSH BC     ;spravi prejšnjo vrednost...
POP HL      ;...in jo poberi v HL.
```

Isto dosežemo z enim samim ukazom

```
EX (SP), HL  ,
```

kar je precej bolj enostaven način.

UPORABA: Vrednosti v registrih B in C bi želeli zamenjati med seboj. Najhitrejšo rešitev nam ponuja uporaba sklada (natančneje: ukazov PUSH, POP ter INC SP - "povečaj kazalec sklada"). Poskusite jo najti sami, preden pogledate, kaj smo napisali!

```
C5          PUSH BC   ;na sklad vedno porine najprej visoki in
                       nato nizki zlog
C5          PUSH BC   ;SP kaže zdaj naslov vrednosti C na skladu
33          INC SP    ;SP kaže zdaj naslov vrednosti B
C1          POP BC    ;vedno se najprej sname nizki in nato
                       visoki zlog
33          INC SP    ;tako uravnamo sklad
C9          RET
```

REŠITEV NALOGE O PROSTEM POMNILNIKU

Naslov, do katerega segajo program in spremenljivke v basicu, je 16-bitno število, spravljeno v E_LINE. Če torej napolnimo HL s tem naslovom, smo že na pol pota

```
LD HL, (E_LINE).
```

Zaradi prenosa moramo spustiti zastavico prenosa (najlaže z ukazom AND A, o katerem bomo govorili v naslednjem poglavju). Dobili ste petico, beri odlično, če ste pomislili na spuščanje zastavice, a niste vedeli kako:

```
AND A .
```

Zdaj odštejemo še kazalec sklada

```
SBC HL, SP.
```


Ker je naslov v kazalcu sklada višji kot naslov konca programa, bo rezultat negativen. Število moramo torej pretvoriti v pozitivno obliko. Uporabili bomo par BC (DE bi bil ravno tako dober) in vanj prestavili vrednost HL

```
PUSH HL
POP BC.
```

HL še vedno vsebuje isto vrednost (HL = BC). Da bi dobili HL = -BC, moramo BC dvakrat odšteti od HL. Ne pozabite, da je naše prvo odštevanje dvignilo zastavico prenosa in jo moramo zato znova spustiti

```
AND A
SBC HL, BC
SBC HL, BC .
```

HL zdaj vsebuje število prostih zlogov v pozitivni obliki. Da bi to število dobili kot rezultat funkcije USR, ga moramo prestaviti v BC

```
PUSH HL
POP BC
```

in se vrniti

```
RET .
```

Je bila pot do rešitve zelo težka?

LOGIČNE OPERACIJE

Tri operacije so v strojnem jeziku prav tako pomembne kot osnovne računske operacije v vsakdanjem življenju. Imenujemo jih Boolove, po velikanu angleške matematične logike, ki je oblikoval njihova pravila. Te operacije so

in (AND),
ali (OR) ter
ne (CPL).

Operacije, ki uporabljajo celotne vrednosti, so vam že domače. Logične operacije so pomembne ravno zato, ker delujejo na posamezne bite števila.

Oglejmo si prvo, "in" (AND):

bit A	bit B	rezultat "A in B"
0	0	0
1	0	0
0	1	0
1	1	1

Rezultat operacije "in" je 1, *le* če imata *tako* A *kot* B, vsak vrednost 1. Morda se sprašujete: "Kakšen smisel ima takšna operacija ?" "In" je izredno koristen, ker lahko z njim spremenimo posamezen zlog tako, da ima "prižgane" le določene bite (t.i. da imajo le določeni biti vrednost 1). Primer: želimo, da bi imelo 8-bitno število vrednost med 0 in 7. To pomeni, da mora imeti zgornjih pet bitov vrednost 0. Vrednost 1 smejo imeti le biti 0, 1 in 2 (če bi jo imel tudi bit 3, bi bilo število med 0 in najmanj 8):

```
        0 0 0 0 0 1 0 1      =  5
        <-------->
        Ti biti morajo imeti vrednost 0.
```

Če torej vzamemo neko število (katerega vrednosti navadno niti ne poznamo) in naredimo "in" z vrednostjo 7, bo rezultat število med 0 in 7. Na primer:

```
        0 1 1 0 1 0 0 1      =  105
"in"    0 0 0 0 0 1 1 1      =    7
        0 0 0 0 0 0 0 1      =    1 (število med 0 in 7!)
```

Takšen postopek imenujemo "maskiranje". Z80 dovoljuje le "in"

UKAZI ZA LOGIČNE OPERACIJE

Krajšava	Zlogov	Čas stanj	T	Učinek na zastavice C	Z	PV	S	N	H
AND r	1	4		0	*	*	*	0	0
AND n	2	7		0	*	*	*	0	0
AND (HL)	1	7		0	*	*	*	0	0
AND (IX +dis)	3	19		0	*	*	*	0	0
AND (IY +dis)	3	19		0	*	*	*	0	0
OR r	1	4		0	*	*	*	0	0
OR n	2	7		0	*	*	*	0	0
OR (HL)	1	7		0	*	*	*	0	0
OR (IX +dis)	3	19		0	*	*	*	0	0
OR (IY +dis)	3	19		0	*	*	*	0	0
XOR r	1	4		0	*	*	*	0	0
XOR n	2	7		0	*	*	*	0	0
XOR (HL)	1	7		0	*	*	*	0	0
XOR (IX +dis)	3	19		0	*	*	*	0	0
XOR (IY +dis)	3	19		0	*	*	*	0	0

Oznake:

r = 8 bitni register
n = 8-bitno število
* = zastavica se ob operaciji spremeni
0 = zastavica se spusti (postane 0)
1 = zastavica se dvigne (postane 1)
- = zastavica ostane, kot je bila

Procesor Z80A: 7 T stanj = 2 mikrosekundi

z registrom A, ki ga zato v ukazu sploh ni treba omenjati. Navesti moramo le drugi operand. Ta pa je lahko katerikoli drug 8-bitni register, število ali (HL). Enako velja za "ali" in "izključni ali". Krajšave so

```
AND r
AND n
AND (HL).
```

Pravila operacije "ali" (OR) so

bit A	bit B	rezultat "A ali B"
0	0	0
1	0	1
0	1	1
1	1	1

Rezultat je torej 1, če sta *ali* A *ali* B enaka 1. Tudi "ali" je zelo uporaben, ker lahko z njim damo kateremukoli bitu 8-bitnega števila vrednost 1. Če bi npr. želeli, da je neko število liho, bi moral imeti bit 0 tega števila vrednost 1. To dosežemo takole

```
LD A, število
OR 1 .
```

Tretja osnovna operacija, "ne" (CPL), je zelo preprosta:

bit A	rezultat "ne A"
1	0
0	1

Ukaz CPL (oz. operacijo "ne") lahko izvršimo le na registru A, ki ga zato sploh ni treba omeniti.

Ukaz XOR - izključni ali - je lahko razumljiv, v programih pa ga redkeje uporabljamo. To je sestavljena operacija, katere rezultat je 1 le, kadar ima vrednost 1 *ali samo* A *ali samo* B. Dobljena vrednost je torej enaka rezultatu OR v vseh primerih, razen kadar imata in A in B vrednost 1:

XOR = OR - AND

bit A	bit B	rezultat "A in B"
0	0	0
1	0	1
0	1	1
1	1	0

UČINEK NA ZASTAVICE

Logične operacije vplivajo na vse zastavice.

Ničelna zastavica	se dvigne (postane 1), če je rezultat 0.
Zastavica znaka	se dvigne (postane 1), če ima bit 7 rezultata vrednost 1.
Zastavica prenosa	se spusti po vsaki logični operaciji.
Zastavica parnosti	se dvigne, če je v rezultatu sodo število "prižganih" bitov, na primer:

rezultat		stanje zastavice
0 1 1 0 1 1 1 0	-->	spuščena (= 0),
0 1 1 0 1 0 1 0	-->	dvignjena (= 1).

Zastavica polprenosa in zastavica odštevanja	sta po logičnih operacijah spuščeni.

Zelo priročne so operacije registra A na samem sebi:

ukaz	učinek
AND A	A ostane nespremenjen, zastavica prenosa se spusti.
OR A	A nespremenjen, zastavica prenosa se spusti.
XOR A	A postane 0, zastavica prenosa se spusti.

Ti ukazi so priljubljeni, ker z njimi z enim zlogom dosežemo učinek, za katerega bi sicer potrebovali daljši ukaz. Tako namesto LD A, 0 (kod 3E 00) zelo pogosto uporabljamo XOR A. Prav tako zelo pogosto uporabljamo AND A, da bi spustili zastavico, npr. pred operacijama ADC (prištej s prenosom) ali SBC (odštej s prenosom).

UPORABA

```
78          LD A, B
A1          AND C       ;vrstico nadomestite kasneje še z OR C
06 00       LD B, 0       (kod B2) in XOR C (kod A9).
4F          LD C, A
C9          RET
```

Program vam da rezultat logične operacije med vrednostma registrov B in C (se pravi med visokim in nizkim zlogom naslova START ZA USR). Program lahko na začetku dopolnite z ukazom LD BC, XX XX. Tako boste lahko izvajali logične ukaze s poljubnimi pari števil.

V sistemski spremenljivki ATTR_P (naslov 23693) so med drugim spravljeni podatki o barvi črnila (v bitih 0, 1 in 2) ter o barvi papirja (v bitih 3, 4 in 5). Z naslednjim programom boste spremenili barvo črnila (učinek je enak ukazu INK XX v basicu):

```
3A 8D 5C   LD A, (ATTR_P)   ;naloži v A zlog s podatki o barvi
                             papirja in črnila
E6 F8      AND F8           ;izbriši barvo črnila rabljeno
                             doslej in ...
F6 XX      OR XX            ;..vpiši novo (XX je številka barve)
32 8D 5C   LD (ATTR_P), A   ;shrani novo vrednost v sistemsko
C9         RET               spremenljivko.
```

Podobno spremenimo barvo papirja, le da namesto XX ne moremo neposredno vstaviti številke barve. V našem primeru bo barva papirja postala modra

```
3A 8D 5C   LD A, (ATTR_P)
E6 C7      AND C7
F6 0F      OR 0F
32 8D 5C   LD (ATTR_P), A
C9         RET
```

Poskusite ugotoviti, kakšne vrednosti je treba uporabiti namesto 0F za različne barve papirja. Ni težko, le z dvojiškim zapisom si je treba pomagati!

SKOKI IN ZANKE

Skoki in zanke dajejo računalniškemu programu njegovo pravo moč. Z njimi lahko - tudi glede na rezultate prejšnjih ukazov - izvajate različne dele programa. Zato so v programerjevem delu nepogrešljivo orodje. Vendar svoboda, ki jo ponujajo, povzroča tudi težave, ustvarja programe, ki jim je težko slediti in jih je nemogoče popravljati. Toplo vam priporočamo, da svoje programe pazljivo oblikujete, preden jih pretvorite v strojni kod. Skoki so namreč tisti ukazi, ki vas bodo pogosto zvabili od načel dobrega programiranja.

"GO TO" V STROJNEM JEZIKU

Iz basica poznate ukaz "GO TO vrstica", ki nadaljuje izvajanje programa v določeni vrstici. Podobno in precej preprosto je preusmerjanje v strojnem jeziku. Le določiti je treba naslov, na katerem bo CP našel naslednji ukaz - s tem je stvar že skoraj opravljena. Najpreprostejši je enostavni ukaz "skoči" (angleško JUMP), ki ima obliko

```
JP nn
JP (HL)
JP (IX)
JP (IY)
```

Ukaz JP nn lahko preoblikujemo tako, da postane odvisen od stanja določene zastavice. Ukaz za pogojni skok je

```
JP cc, nn .
```

cc je pogoj, ki mora biti izpolnjen, da bo skok izveden (gl. tabelo). Če bi npr. srečali ukaz

```
JP Z, 0000
```

bi to prebrali "skoči, če je dvignjena ničelna zastavica, na naslov 0000" (Na tem naslovu začne CP svoje delo, ko vklopite računalnik. Ta ukaz bi uporabili, če bi želeli povsem očistiti pomnilnik in začeti, kot da ste računalnik ravnokar vključili).

UKAZI ZA SKOKE IN ZANKE

Krajšava	Zlogov	Čas stanj T	Učinek na zastavice C	Z	PV	S	N	H
JP nn	3	10	-	-	-	-	-	-
JP (HL)	1	4	-	-	-	-	-	-
JP (IX)	2	8	-	-	-	-	-	-
JP (IY)	2	8	-	-	-	-	-	-
JP cc, nn	3	10	-	-	-	-	-	-
JR dis	2	12	-	-	-	-	-	-
JR cc, dis	2	7/12	-	-	-	-	-	-
DJNZ dis	2	8/13	-	-	-	-	-	-

Oznake:

dis = odmik
nn = 16-bitno število (naslov)
\- = zastavica ostane, kot je bila
Procesor Z80A: 7 T stanj = 2 mikrosekundi

cc pomeni določen pogoj. Skok bo izveden, če je pogoj izpolnjen.

pogoj cc	skok bo izveden, če ...
Z (Zero)	...je ničelna zastavica dvignjena (...je rezultat 0);
NZ (Not Zero)	...je ničelna zastavica spuščena (...rezultat ni 0);
C (Carry)	...je zastavica prenosa dvignjena;
NC (Not Carry)	...je zastavica prenosa spuščena;
M (Minus)	...je zastavica znaka dvignjena (...je rezultat negativen);
P (Positive)	...je zastavica znaka spuščena (...je rezultat pozitiven);
PE (Parity Even)	...je zastavica parnosti dvignjena;
PO (Parity Odd)	...je zastavica parnosti spuščena.

Z ukazom JR cc, dis so dovoljeni le pogoji Z, NZ, C in NC.
Kjer sta v tabeli navedeni dve časovni vrednosti, velja prva le tedaj, ko pogoj ni izpolnjen.

Bodite pozorni - CP ne dovoljuje napak! Če mu ukažete "skoči!", bo skočil in se ne bo oziral na to, kje je pristal. Tudi če bo doskočil med podatke ali na drugi zlog dvozložnega ukaza, ga bo prebral in ga - ker vsak kod pomeni nek ukaz izvedel. Za pojasnitev vzemimo skok na ukaz LD A, (03C3H). V strojnem kodu ima ta ukaz obliko 3A 03 C3. Po skoku bo CP zagledal kod 3A in bo pravilno izvedel ukaz. Lahko pa se po pomoti zgodi, da bo pristal na drugem zlogu. V tem primeru bo najprej povečal BC (03 je kod ukaza INC BC), prebral C3 (kar je kod ukaza JP nn), prebral dva zloga naslednjega ukaza kot naslov ... in tako odskakal neizbežnemu polomu nasproti.

CP ima poseben registrski par, ki ga imenujemo programski števec. Ta mu pove, kje bo našel naslednji ukaz, ki ga mora izvesti. V programu, kjer ni skokov, pogleda CP ukaz, poveča vrednost programskega števca za toliko, kolikor je ukaz dolg in ga izvede. Če torej naleti na ukaz, ki zaseda 2 zloga, poveča programski števec za dva, če naleti na ukaz s 4 zlogi, poveča števec za štiri. Kadar CP dobi ukaz "skoči", preprosto nadomesti vrednost v programskem števcu z vrednostjo, ki ste jo določili. Zato ne smete dovoliti, da bi se vam v ukaz JP prikradla napaka.

DOLGI IN KRATKI SKOKI

Ukaze, o katerih smo ravnokar govorili, označujemo "dolgi skoki", ker nam 16-bitna vrednost v ukazu omogoča skok na katerikoli naslov. Omeniti velja dve pomanjkljivosti dolgih skokov:
- pogosto ne želimo skočiti daleč, a moramo za naslov še vedno uporabiti 16-bitno število;
- programov ne moremo brez težav premakniti drugam v pomnilnik, ker vsebujejo absolutne naslove.

Te težave premosti procesor Z80 s "kratkim" ali "relativnim" skokom. Ta mu omogoča skoke do 127 zlogov navzgor in do 128 zlogov navzdol od trenutnega naslova v programskem števcu. Dolžino skoka lahko tako določimo z enim samim zlogom! Splošna oblika relativnega skoka je

JR dis .

JR pomeni "skoči relativno" (angleško JUMP RELATIVE), dis pa je

odmik od trenutnega naslova, tj. dolžina skoka. Tudi ta ukaz lahko preoblikujemo tako, da postane odvisen od določenega pogoja:

```
JR cc, dis .
```

cc je pogoj, ki mora biti izpolnjen. Pri dolgih skokih lahko uporabimo osem različnih pogojev, pri relativnih pa le štiri: Z, NZ, C ali NC.

Vrednost odmika (dis) se *prišteje* programskemu števcu (ga torej ne nadomesti, tako kot pri dolgem skoku). Pri odmiku CP vedno upošteva pravilo "iztegnjen palec - negativno število". Zato je odmik lahko pozitiven (skok naprej) ali negativen (skok nazaj) in ima lahko le vrednosti od -128 do +127.

Pozor ! Ko CP začne izvajati relativni skok, programski števec že kaže na ukaz, ki neposredno sledi skoku. Kadar namreč CP pride do ukaza JR, ve da ta ukaz zaseda dva zloga. Zato poveča programski števec za 2 - tako da ta kaže na ukaz za skokom. Poglejmo primer

```
naslov      ukaz
 ...         ....
30000       ADD A, B
30001       JR Z, 02H
30003       LD B, 0
30005       LD HL, 4000H
 ...         ....
```

CP izvaja ta del programa takole:

1. korak Poglej vsebino pomnilniške celice na naslovu 30000. Ker je to en zlog dolg ukaz, povečaj programski števec na 30001. Izpolni ukaz.
2. korak Poglej vsebino celice na naslovu 30001. Ker je to del ukaza, dolgega dva zloga, povečaj programski števec na 30003, poglej še vsebino naslednje celice in izpolni ukaz.

Tu se mora CP odločiti, kaj naj stori s programskim števcem:
- če je ničelna zastavica dvignjena, prišteje k programskemu števcu 2 in tako skoči na naslov 30005;
- če je ničelna zastavica spuščena, ne spreminja programskega števca in nadaljuje z naslednjim ukazom (na naslovu 30003).

Glede na stanje zastavice se bo izvršil ali ukaz na naslovu 30005 ali na 30003.

Relativni skok je lahko usmerjen tudi nazaj. V tem primeru je odmik negativna vrednost.

OREH: Relativni skok zaseda dva zloga, programski števec pa kaže na ukaz, ki mu neposredno sledi. Kakšen bi bil torej učinek ukaza JR FEH (= JR -2) ?

ZANKE V STROJNEM JEZIKU

Iz basica dobro poznamo "FOR ... NEXT" zanke:

```
FOR I = 1 TO 6
LET C = C + 1
NEXT I
```

Premislimo, kako bi v strojnem jeziku napisali takšno zanko, če bi uporabili aritmetične operacije in relativne skoke:

```
        LD B, 1         ;postavi števec na 1
        LD A, 7         ;mejna vrednost
Zanka   INC C           ;C = C + 1
        INC B           ;povečaj števec
        CP B            ;je B enak A ?
        JR NZ, Zanka    ;če ni, ponovi zanko
```

Program bo deloval, vendar zaseda tri registre: enega za število, drugega za povečevanje števca in tretjega za shranjevanje mejne vrednosti. Poleg tega ukaz, ki povečuje števec, ne spremeni nobene zastavice, ko je naloga izpolnjena (tj. ko števec doseže mejno vrednost).

Mnogo bolje bi bilo šteti navzdol!

Vemo, da moramo zanko ponoviti šestkrat. Zakaj ne bi dali registru B vrednosti 6 in šteli navzdol ?

```
0606        LD B, 6         ;nastavi števec
0C   Zanka  INC C           ;C = C + 1
05          DEC B           ;zmanjšaj števec
20FC        JR NZ, Zanka    ;če še ni konec, ponovi zanko
C9          RET             ;vrni se
```

To je dosti bolj učinkovit način.

Kot vidite, smo v programu označili naslov "Zanka" in nato zapisali ukaz "JR NZ, Zanka". To je običajni način zapisa v zbirni obliki in olajšuje prebiranje ter razumevanje programa. Še vedno pa morate v ukazu navesti odmik (in ne naslova, kot bi rekli na prvi pogled). Ob tem še uporaben nasvet, ki vam bo olajšal izračunavanje odmikov v krajših programih. Pri skoku nazaj začnite pri *drugem* zlogu ukaza JR šteti od 255 (= FFH) nazaj. Odmik je vrednost, ki jo boste dobili, ko boste prišli do zloga, na katerem naj se skok konča. Preizkusite to pravilo v našem primeru (kjer smo dobili vrednost odmika FCH = 252). Pri skoku naprej začnete pri *prvem* zlogu *naslednjega* ukaza šteti od 0. Ko dospete do naslovnega zloga, imate izračunano vrednost odmika.

Procesor Z80 pozna poseben ukaz, ki opravi delo zadnjih dveh vrstic prejšnjega programa. To je ukaz

```
DJNZ dis ,
```

ki ga preberemo "zmanjšaj B in skoči, če ničelna zastavica ni dvignjena, za odmik dis" (angleško DECREASE AND JUMP IF NOT ZERO). Ukaz zaseda dva zloga in nam torej prihrani en zlog v primerjavi z našim prejšnjim zapisom. Zaradi tega ukaza se register B običajno uporablja kot števec. Omejitev ukaza DJNZ (tako kot vseh relativnih skokov) je v štetju do 255. Seveda pa lahko ukaze DJNZ "vgnezdimo", če je potrebno:

```
        LD B, 10H       ;B = 16
Zanka1  PUSH BC         ;spravi vrednost B na sklad
        LD B, 0         ;s  tem pravzaprav postavimo B na 256
Zanka2  -----
                        ;vmesni izračuni
        -----
        DJNZ Zanka2     ;opravljeno 256-krat ?
        POP BC          ;vrni prvotno vrednost B
        DJNZ Zanka1     ;ponovi večjo zanko 16-krat.
```

Seveda raba ukaza DJNZ v tem primeru ni obvezna. Prav lahko bi dali paru BC vrednost 1000H ter dodali nekaj ukazov, ki bi zmanjševali BC in testirali, ali je že enak 0.

ČAKALNE ZANKE

Včasih želimo v programu iz tega ali onega razloga narediti kratek odmor. V ta namen uporabljamo čakalne zanke, ki jih napravimo s pomočjo ukaza DJNZ:

```
        LD B, n
Čakaj DJNZ Čakaj .
```

Ukaz DJNZ Čakaj bo vrnil CP na samega sebe tolikokrat, da bo vrednost B postala 0. Šele potem bo nadaljeval z izvajanjem programa. To je obenem odgovor na naš OREH (str. 75), kaj se zgodi ob ukazu

```
Čakaj JR Čakaj.
```

Te zanke CP ne bo nikoli zapustil. Morali mu boste pomagati in izključiti računalnik. Najdaljši premor, ki ga povzroči enkratna uporaba čakalne zanke z ukazom DJNZ, traja nekaj manj kot eno tisočinko sekunde (natančneje: 952 mikrosekund).

Ker smo ravno spregovorili o "odmorih" v programu, naj omenimo še ukaz, ki sicer ne sodi v to poglavje. To je

```
HALT ,
```

ki prekine izvajanje strojnega programa do naslednje prekinitve (angleško interrupt). Njegov učinek je enak ukazu PAUSE 1 v basicu. A bodite previdni! Če so prekinitve onemogočene (ukaz DI), bo ukaz HALT nepreklicno ustavil izvajanje - in spet bo treba izvleči vtič iz vtičnice.

UPORABA: Pripravljeni program je zasnovan tako, da boste z njim lahko preizkusili dolge in kratke skoke. Spreminjati bo treba četrto (za kratke skoke) ali četrto in peto vrstico (za dolge skoke). Potrebne kode boste našli v tabeli na koncu knjige. V programu spreminjajte vrednost v registru A (prva vrstica), tako boste vplivali na zastavice. Kadar bo CP izvedel skok, bo rezultat programa 1, sicer pa 0.

```
3EXX        LD A, XX
B8          CP B
010000      LD BC, 0000
2002        JR NZ, Skok
00          NOP
```

```
C9          RET             ;mesto za povratek, če ni bilo skoka.
03     Skok INC BC
C9          RET
```

Naslednji primer vključuje ukaz DJNZ (ukazu RST 10, ki tudi nastopa, se bomo posvetili v kasnejših poglavjih). Na zaslon bo izpisal Spectrumov znakovni niz, z grafičnimi znaki vred.

```
0670        LD B, 70H       ;zanka naj se ponovi 112-krat
3E90 Zanka  LD A, 90H
90          SUB B
D7          RST 10          ;izpiši znak
10FA        DJNZ Zanka
C9          RET
```

Podrobnosti programa vam (najbrž) še niso povsem razumljive, pa si zato ne belite glave. Ko preberete poglavje "Izpisovanje na zaslon", o težavah z razumevanjem ne bo ne duha ne sluha.

PODPROGRAMI

Ko načrtujemo program, opazimo, da so nekatera opravila ali izračuni potrebni večkrat na različnih mestih v programu. Dele programa, ki opravljajo takšne naloge, izluščimo iz programa in jih izdelamo neodvisno od njega. Imenujemo jih podprogrami. Njihova vloga je podobna vlogi podprogramov ali definiranih funkcij v basicu. Podobna je tudi njihova uporaba. Celo funkcija USR pravzaprav obravnava naše strojne programe kot podprograme, saj moramo na koncu vedno uporabiti ukaz RET (RETURN = vrni se). Dobro in lepo napisan program pogosto uporablja podprograme (učeno rečemo, da je strukturiran). V takšnem programu lahko tudi marskikateri dolg skok nadomestimo z uporabo podprograma.

Iste podprograme lahko uporabljamo skupaj z različnimi glavnimi programi. To je celo zelo v navadi. Tako si ustvarimo zalogo podprogramov (navadno ji pravimo knjižnica podprogramov), ki je zelo dragocena, če ti programi opravljajo pogosto potrebno delo. Vzemimo nekoliko poenostavljen primer: če bi se na debelo ukvarjali s pisanjem iger v strojnem jeziku, bi pogosto rabili zaključen del programa, ki bi ob koncu igre na zaslon izpisal "KONEC" ter število doseženih točk. Takšen del bi oblikovali kot podprogram in bi ga lahko priključili vsem igram. Če bi se ukvarjali s statistiko, bi bil npr. eden od podprogramov namenjen izračunu poprečne vrednosti in podobno.

Ker so podprogrami samostojni deli, jih lahko tudi preizkušamo neodvisno od glavnega programa. Tako z uporabo podprogramov skrajšamo čas pisanja programov, njihovo preizkušanje in prevajanje v strojni kod. Takšni programi so tudi bolj čitljivi in laže razumljivi. Vse to precej olajša delo.

Kot vsi dobri programi, je tudi Spectrumov nadzorni program strukturiran. Njegove podprograme (z njimi se boste seznanili npr. s pomočjo knjige I. Logana in F. O'Hare The Complete Spectrum ROM Disassembly, Melbourne House Pubishers, Cheddington 1983) lahko uporabite v svojih programih, kar pomeni poleg omenjenih prednosti tudi precej krajši glavni program.

Ukaz za skok v podprogram imenujemo klic podprograma in ima splošno obliko

CALL nn

KLICI, POVRATKI IN PONOVNI ZAGONI

Krajšava	Zlogov	Čas stanj T	Učinek na zastavice C	Z	PV	S	N	H
CALL nn	3	17	-	-	-	-	-	-
CALL cc, nn	3	10/17	-	-	-	-	-	-
RET	1	10	-	-	-	-	-	-
RET cc	1	5/11	-	-	-	-	-	-
RST p	1	11	-	-	-	-	-	-

Oznake:

- = zastavica ostane, kot je bila
nn = 16-bitno število (naslov)
p = ima lahko vrednost 0, 8, 10, 18, 20, 28, 30 ali 38 (H)
cc = pogoj. Skok se izvede, če je pogoj izpolnjen.

pogoj cc	skok se izvede, če ...
Z (Zero)	...je ničelna zastavica dvignjena (...je rezultat 0);
NZ (Not Zero)	...je ničelna zastavica spuščena (...rezultat ni 0);
C (Carry)	...je zastavica prenosa dvignjena;
NC (Not Carry)	...je zastavica prenosa spuščena;
M (Minus)	...je zastavica znaka dvignjena (...je rezultat negativen);
P (Positive)	...je zastavica znaka spuščena (...je rezultat pozitiven);
PE (Parity Even)	...je zastavica parnosti dvignjena;
PO (Parity Odd)	...je zastavica parnosti spuščena.

Kjer sta v tabeli navedeni dve časovni vrednosti, velja prva v primeru, da pogoj ni izpolnjen.

Procesor Z80A: 7 T stanj = 2 mikrosekundi

(CALL = kliči). Kot vidite, je treba navesti naslov - absolutno število - kjer se program začenja. Splošni ukaz za vrnitev iz podprograma že dobro poznate:

```
RET
```

Ko CP naleti na ukaz CALL nn, najprej poveča programski števec (kot pri vsakem ukazu), tako da ta kaže na prvi zlog za ukazom CALL. Nato porine vrednost programskega števca na sklad, vrednost v programskem števcu pa nadomesti z nn. Tako skoči na določen naslov. Ko sreča ukaz RET, ravna podobno, vendar v obratni smeri - sname zadnjo vrednost s sklada in jo naloži v programski števec. To omogoča preprosto spreminjanje naslova za povratek (ki pa je tvegano, če ni dobro premišljeno) z ukazi ravnanja s skladom. Primer takšnega spreminjanja boste našli v poglavju "Izpisovanje na zaslon".

Podobno kot pri skokih lahko tudi podprograme kličemo in zaključujemo pogojno. Splošna ukaza sta

```
CALL cc, nn
```

in

```
RET cc
```

.

cc označuje iste pogoje kot pri dolgih skokih - tj. stanje ene od štirih zastavic (ničelne, prenosa, parnosti in znaka - gl. tabelo). Stanje zastavic je odvisno od zadnje izvršene operacije. Zato je dobro, da postavimo pogojne klice in povratke takoj za ukaz, ki vpliva na zastavico. Primer:

```
LD A, r
CP 1
CALL Z, ena
CP 2
CALL Z, dva
CP 3
CALL Z, tri
```

Takšen del programa omogoča klic različnih podprogramov glede na vrednost v registru r. Če vemo, da je v registru lahko le eno od števil 1, 2 ali 3, zapišemo še krajše

```
        LD A, r
        CP 2
        CALL Z, dva     ;A = 2
        CALL C, ena     ;A < 2  -->  A = 1
        CALL tri        ;A > 2  -->  A = 3
```

Ukaz CP n vpliva tako na ničelno zastavico kot na zastavico prenosa - ju spremeni glede na rezultat - ukazi CALL pa na zastavice ne vplivajo.

Podobno velja za povratke iz podprogramov, kjer lahko prav tako s pridom uporabimo ukaz RET cc.

UPORABA: Tokrat bomo pogledali le uporabo treh preprostih podprogramov v ROM-u. Prvi naj bo bolj za šalo. Nadomestite v vrstici 210 urejevalnika ukaz PRINT z RANDOMIZE.

```
CD 95 12    CALL 1295H
C9          RET
```

Popravite zdaj vrstico 210 in preizkusite program še enkrat.

Drugi program prinaša le okus po uporabi zvoka v strojnih programih

```
21 6A 06    LD HL, 066A
11 05 01    LD DE, 0105
CD B5 03    CALL 03B5
C9          RET
```

Izbrišite zdaj ukaz CLS v vrstici 200 urejevalnika in vnesite

```
CD 6B 0D    CALL 0D6B
C9          RET
```

To je že popolnoma uporabno. Poklicali smo podprogram, ki obriše zaslon. V vaših programih vam bo marsikdaj prišel prav.

PONOVNI ZAGONI

Skupina en zlog dolgih ukazov RST (RESTART) opravlja isto nalogo kot brezpogojni ukaz CALL, vendar lahko z njimi kličemo le 8 naslovov v začetku ROM-a. Z80 pozna osem ukazov RST. Opisali bomo, kakšen učinek ima njihova raba v Spectrumu (pri drugih računalnikih je učinek istih ukazov drugačen). Ukazi RST so:

RST 00 učinkuje kot izklop in ponoven vklop računalnika;

RST 08 služi za sporočila o napakah v basicu. Sledi mora 8-bitno število - kod napake;

RST 10H na zaslon izpiše znak, katerega kod najde v registru A;

RST 18H register A napolni z vrednostjo zloga, katerega naslov je v sistemski spremenljivki CH-ADD;

RST 20H se uporablja pri tolmačenju programov v basicu;

RST 28H se uporablja za računanje s plavajočo vejico (tj. pri številih z decimalkami). Zlogi, ki sledijo ukazu, določijo operacijo;

RST 30H preuredi delovni prostor tako, da dobimo v njem toliko prostih zlogov, kolikor je vrednost v BC;

RST 38H preleti tipkovnico (tj. kontrolira ali je pritisnjena kakšna tipka) in poveča števec časa.

Za uporabnike Sinclairjevega vmesnika 1 je zlasti zanimiv ukaz RST 08. Z njim v strojnem jeziku upravljamo mikrotračne enote, vmesnik RS232 in prenos podatkov v omrežju Spectrumov. Več o tem boste našli v knjigi I. Logana Spectrum microdrive book (Melbourne House Publishers, Tring 1983). Ukaza, ki ju najpogosteje rabimo v nerazširjenem Spectrumu, sta RST 10 in RST 28. Bolj kot zanimivost lahko preizkusite naslednji ukaz

```
CF          RST 08
XX          XX              ;vnesite različne vrednosti
```

SKUPINSKE OPERACIJE

Skupinske operacije predstavljajo zadnji niz zelo pogosto rabljenih ukazov. Naslednja poglavja bomo posvetili ukazom, ki so v primerjavi z doslej obravnavanimi uporabljeni manjkrat. Vsekakor vam je jezik, ki ga razume centralni procesor Z80 zdaj že precej domač in bi s tem, kar veste, že znali pisati programe.

Skupinski ukazi premagujejo visoke gore z enim samim skokom, hitro kot blisk. Manj pesniško povedano gre za ukaze, ki delujejo na cele skupine pomnilniških celic, ne le na en 8-bitni zlog. Vseh ukazov je osem in jih delimo v dve skupini: v ukaze (razširjenih) primerjav in prelaganja. V vsaki skupini sta po dva avtomatična in dva neavtomatična ukaza. Avtomatični ukazi so bolj uporabni in zato bolj pogosti.

RAZŠIRJENE PRIMERJAVE

Prvi v tej skupini je

```
CPI  ,
```

ki ga preberemo "primerjaj in povečaj" (COMPARE AND INCREASE). CPI primerja vrednost v registru A z (HL) in poveča HL. Po ukazu CPI torej HL že kaže na naslednji naslov, pripravljen na ponovitev. S tem ukazom lahko na primer napišemo del programa, ki bo preiskal pomnilnik, da bi našel iskano vrednost:

```
Išči  CPI
      JR NZ, Išči
```

Zanka se bo ponavljala, dokler ne bo našla iskane vrednosti (kot pri vseh primerjalnih ukazih se bo v takšnem primeru dvignila ničelna zastavica). Kaj pa če iskane vrednosti ni v pomnilniku ? Na srečo so ustvarjalci našega procesorja mislili na to, zato CPI ob vsaki ponovitvi zmanjša BC! Iskanje se konča
- ko procesor najde iskano vrednost ali
- ko BC doseže vrednost 0.

Tako z BC določimo dolžino (v zlogih) dela pomnilnika, ki ga želimo preiskati, s čimer določimo tudi konec iskanja.

SKUPINSKE PRIMERJAVE IN PRELAGANJA

Krajšava	Zlogov	Čas stanj T	Učinek na zastavice C	Z	PV	S	N	H
LDI	2	16	-	-	*	-	0	0
LDD	2	16	-	-	*	-	0	0
LDIR	2	21/16	-	-	0	-	0	0
LDDR	2	21/16	-	-	0	-	0	0
CPI	2	16	-	*	*	*	1	*
CPD	2	16	-	*	*	*	1	*
CPIR	2	21/16	-	*	*	*	1	*
CPDR	2	21/16	-	*	*	*	1	*

Oznake:

* = zastavica se ob operaciji spremeni
0 = zastavica se spusti (postane 0)
1 = zastavica se dvigne (postane 1)
- = zastavica ostane, kot je bila

Pri avtomatičnih ukazih velja krajši čas v primeru, ko se operacija konča (BC = 0 ali A = (HL)).

Procesor Z80A: 7 T stanj = 2 mikrosekundi

Zgornja dva ukaza lahko nadomestimo z enim samim, tako imenovanim avtomatičnim ukazom

CPIR ,

kar pomeni "primerjaj, povečaj in ponovi" (COMPARE, INCREASE AND REPEAT). Pred izvajanjem moramo
- v register A naložiti iskano vrednost,
- v par HL naložiti začetni naslov dela pomnilnika, ki ga želimo preiskati ,
- v par BC pa dolžino tega dela (število zlogov).

Ta ukaz omogoča procesorju, da avtomatično ponavlja primerjavo, dokler ne najde iskane vrednosti ali ne pride do konca preiskovane skupine. Če iskano vrednost najde, se ničelna zastavica dvigne, zastavica znaka spusti, medtem ko HL kaže en naslov za zlogom z iskano vrednostjo. Če iskane vrednosti ne najde, bo v registru BC vrednost 0, ničelna zastavica ter zastavici znaka in prepolnjenja pa bodo spuščene.

Avtomatični način iskanja je mnogo udobnejši, zato je tudi precej več v rabi. Poglejmo še primer, ki dobro pokaže razliko

```
Išči  CPI                    CPIR
      JR Z, Našel            LD A, B
      INC C                  OR C
      DEC C                  JR NZ, Našel
      JR NZ, Išči      Nič   -----
 Nič  -----                  -----
      -----          Našel   -----
Našel -----
```

Račun nam pokaže, da je program, ki uporablja avtomatični ukaz, za dva zloga krajši (kar ni tako pomembno) in *pri vsakem ciklu kar za 17 T stanj hitrejši* - kar je bistvena razlika. Iz "počasnega" programa si velja zapomniti zaporedje ukazov INC in DEC za preizkus, ali je vrednost registra enaka 0. Ker oba ukaza učinkujeta na ničelno zastavico, bo ta po njuni uporabi dvignjena le, če je bila vrednost registra enaka 0. Dodatna prednost je, da s tem ne spreminjamo nobenega drugega registra.

Če želimo del pomnilnika preiskati od konca proti začetku, imamo na voljo ukaz

CPD ,

ki ga preberemo "primerjaj in zmanjšaj" (COMPARE AND DECREASE). Zmanjšaj se seveda nanaša na HL, učinek na BC pa je enak kot pri prejšnjih dveh ukazih. Ustrezni avtomatični ukaz je

CPDR ,

"primerjaj, zmanjšaj in ponovi" (COMPARE, DECREASE AND REPEAT).

PRELAGANJE

Tudi v tej skupini so štirje ukazi:

LDI napolni in povečaj (LOAD AND INCREASE);
LDD napolni in zmanjšaj (LOAD AND DECREASE);
LDIR napolni, povečaj in ponovi (LOAD, INCREASE AND REPEAT) in
LDDR napolni, zmanjšaj in ponovi (LOAD, DECREASE AND REPEAT).

Vzemimo za primer LDI. Ta ukaz
napolni (DE) z (HL),
poveča DE in HL ter
zmanjša BC.

(Mimogrede: ti ukazi so edini, ki napolnijo eno pomnilniško celico z vsebino druge, ne da bi jo bilo potrebno prej prenesti v register). Razumljivo je, da moramo pred izvajanjem vseh ukazov prelaganja dati parom DE, HL in BC ustrezne vrednosti. Avtomatični ukaz LDIR je najpogosteje rabljeni ukaz za skupinske operacije. Opravlja enako delo kakor LDI, le da ga avtomatično ponavlja, dokler BC ne doseže vrednosti nič. Ob koncu operacije DE in HL vsebujeta naslov prvega zloga za preloženo skupino, zastavica prepolnjenja pa je spuščena.

Somerna ukaza LDD in LDDR sta povsem enaka, le da zmanjšujeta DE in HL.

Razlika med ukazoma LDI in LDD (oz. LDIR in LDDR) je pomembna, kadar se skupina zlogov, iz katere informacije jemljemo, prekriva s tisto, v katero jih prenašamo. Recimo, da ukaza uporabljamo pri obdelavi besedila; iz stavka želimo odstraniti neko besedo:

Laž ima zelo kratke nožice.

Če hočemo odstraniti besedo "zelo", moramo samo premakniti ostanek stavka za 5 znakov v levo:

DE = cilj = znak 9
HL = vir = znak 14
BC = dolžina = 14 znakov.

Uporabimo LDI! Po prvem koraku ima DE vrednost 10, HL 15, BC 13, stavek pa je

Laž ima kelo kratke nožice.

Po še dveh ponovitvah imamo

Laž ima krao kratke nožice.

In na koncu

Laž ima kratke nožice.žice.

(Če bi hoteli izbrisati še ostanek za piko, bi morali stavku na začetku dodati dovolj presledkov in povečati BC).

Če bi zdaj hoteli spet napraviti prostor za besedo "zelo", ne bi smeli dati DE vrednost 14, HL 9 in BC 14 ter uporabiti LDI. S tem bi namreč že na prvem koraku začeli uničevati podatke, ki jih želimo prenest (prepričajte se sami!). Pomagati si moramo z LDD, DE mora kazati na konec stavka (22), HL pa mora imeti vrednost 17. (Če bi preostanek prvotnega stavka z ukazom LDI izbrisali, bi morali dati DE vrednost 27 in HL 22). Dolžina seveda ostaja enaka, BC = 14.

Še boljša bi bila v takšnem primeru avtomatična ukaza, ki sta zelo učinkovita. Za premik dvajset tisoč znakov (zlogov) kar je približno 10 tipkanih strani - porabita 12 stotink sekunde.

UPORABA: Prvi primer bo pokazal delovaje CPIR. Njegov rezultat bo naslov prvega zloga v ROM-u, ki vsebuje izbrano vrednost (to vrednost vstavite namesto XX v prvi vrstici in jo pri ponovnih izvajanjih programa spreminjajte). S programom se lahko prepričate, da se v ROM-u pojavljajo vse vrednosti med 0 in 255, vendar nekatere redko (poskusite 154!). Spremenite lahko tudi

vrednosti BC ter HL v drugi in tretji vrsti.

```
3E XX       LD A, XX        ;iskana vrednost
01 FF 3F    LD BC, 3FFF     ;konec ROM-a
21 00 00    LD HL, 0000     ;začetek ROM-a
ED B1       CPIR            ;išči !
44          LD B, H
4D          LD C, L
0B          DEC BC          ;pokaži na naslov iskane vrednosti
C9          RET
```

Izbrišite zdaj v vrstici 200 urejevalnika ukaz CLS. Nato uporabite naslednji program, ki bo z ukazom LDIR preložil vsebino prvih 2048 zlogov zaslonske datoteke v naslednjih 2048 zlogov se pravi, da bo zgornjo tretjino slike na zaslonu prepisal v srednjo tretjino:

```
21 00 40    LD HL, 4000H    ;začetek zaslonske datoteke ima
                             naslov 16384
11 00 48    LD DE, 4800H    ;začetek druge tretjine je 18432
01 00 08    LD BC, 0800H    ;skupina ima 2048 zlogov
ED B0       LDIR            ;premakni skupino
C9          RET
```

MENJAVE REGISTROV

Na začetku knjige smo omenili CP-jeve rokavice, s katerimi lahko CP shrani nekatere podatke na bolj dostopno mesto kot so pomnilniške celice. Rekli smo tudi, da na izmenjalne registre ne moremo vplivati neposredno. Prispodoba o rokavicah je torej blizu resnici. Čeprav zmorejo ohraniti obliko, ne morejo same ne šteti ne računati. Zamenljivi registrski niz najpogosteje uporabljamo za "ohranjanje stanja", npr. pred klicem podprograma, ki bo uporabljal povsem druge vrednosti. Treba pa je biti pazljiv. Poleg povratnega naslova, ki ga procesor ob začetku izvajanja strojnega programa porine na sklad, je za vrnitev v basic potrebna pravilna vrednost v paru H'L' (ta mora biti ob zaključku strojnega programa 2758H oz. 10072). Če je ta vrednost spremenjena, pride do poloma.

Prvi ukaz iz te skupine je

```
EX AF, A'F'  ,
```

ki izmenja para AF in A'F' (EXCHANGE = izmenjaj). Če bi še vedno govorili o rokavicah, bi rekli "zamenjaj rokavice na paru rok AF".

Naslednji je ukaz splošne zamenjave rokavic

```
EXX  ,
```

ki izmenja vse ostale 8-bitne registre:

```
BC <---> B'C'
DE <---> D'E'
HL <---> H'L'
```

To je zelo močan ukaz, a ga ravno to omejuje v uporabi. Ker deluje na vse registre naenkrat, ne moremo obdržati nobene vrednosti (razen v registru A, na katerega EXX ne vpliva). Včasih si pomagamo s skladom:

```
PUSH BC
EXX
POP BC.
```


MENJAVE REGISTROV, UKAZI SET, RESET IN BIT

Krajšava	Zlogov	Čas stanj T	Učinek na zastavice C	Z	PV	S	N	H
EX AF, A'F'	1	4	*	*	*	*	*	*
EXX	1	4	-	-	-	-	-	-
EX DE, HL	1	4	-	-	-	-	-	-
SET b, r	2	8	-	-	-	-	-	-
SET b, (HL)	2	15	-	-	-	-	-	-
SET b, (IX+ dis)	4	23	-	-	-	-	-	-
SET b, (IY+ dis)	4	23	-	-	-	-	-	-
RES b, r	2	8	-	-	-	-	-	-
RES b, (HL)	2	15	-	-	-	-	-	-
RES b, (IX+ dis)	4	23	-	-	-	-	-	-
RES b, (IY+ dis)	4	23	-	-	-	-	-	-
BIT b, r	2	8	-	*	?	?	0	1
BIT b, (HL)	2	12	-	*	?	?	0	1
BIT b, (IX+ dis)	4	20	-	*	?	?	0	1
BIT b, (IY+ dis)	4	20	-	*	?	?	0	1

Oznake:

b = številka bita v operandu (med 0 in 7)
r = 8-bitni register
* = zastavica se ob operaciji spremeni
0 = zastavica se spusti (postane 0)
1 = zastavica se dvigne (postane 1)
- = zastavica ostane, kot je bila
? = učinek na zastavico ni znani

Procesor Z80A: 7 T stanj = 2 mikrosekundi

Tako zamenjamo registre, še vedno pa imamo za delo na voljo staro vrednost BC.

Naslednji ukaz iz te skupine ne vključuje zamenljivih registrov, temveč para DE in HL

```
EX DE, HL .
```

Z njim damo DE vrednost HL in obratno. Ukaz je zelo uporaben. Kot veste, je HL prednostni registrski par, vrednost, s katero želimo računati, pa je pogosto v DE.

Zadnji v tej skupini so ukazi

```
EX (SP), HL
EX (SP), IX
EX (SP), IY   ,
```

ki smo jih že spoznali v poglavju o skladu.

UKAZI SET, RESET IN BIT

Ukazi v teh skupinah so, razen logičnih ukazov, edini, ki omogočajo spreminjanje ter preverjanje vrednosti posameznih bitov v registrih ali pomnilniških celicah. Ker je brskanje po bitih precej puščobno delo, so ti ukazi redkeje v rabi. Poleg tega porabi CP pogosto več časa za spreminjanje vrednosti posameznih bitov kot za spremembo celotnega zloga.

Pridejo pa trenutki, ko moramo vedeti, kakšna je vrednost nekega bita, ali jo moramo celo spremeniti. Le obdržati je treba v mislih, da je marskidaj mogoče doseči isto z logičnimi operacijami.

Skupina "Set, Reset in Bit" omogoča, da bite "prižigamo" (jim damo vrednost 1), jih "ugašamo" (jim damo vrednost 0) ali preverjamo njihovo vrednost.

Prvi niz so ukazi "prižgi"

```
SET b, r
SET b, (HL)
SET b, (IX+ dis)
SET b, (IY+ dis) .
```

Spomnite se, da bite v zlogu oštevilčimo od 0 do 7. Ukaz SET da bitu b v registru ali pomnilniški celici vrednost 1. Ob tem ne vpliva na nobeno zastavico.

Podobno velja za ukaze "ugasni" (RESET), ki uporabljajo iste operande, le da dajo bitu b vrednost 0

```
RES b, r
RES b, (HL)
RES b, (IX+ dis)
RES b, (IY+ dis) .
```

Ukaz BIT bi moral pravzaprav imeti obliko "BIT ?", ker z njim preverjamo vrednost posameznega bita:

```
BIT b, r
BIT b, (HL)
BIT b, (IX+ dis)
BIT b, (IY+ dis) .
```

Vsebina registrov ali pomnilniških celic se ob tem ne spremeni. Edini rezultat tega niza ukazov je stanje ničelne zastavice:
- če je vrednost preiskovanega bita 0, je zastavica dvignjena;
- če je vrednost preiskovanega bita 1, je zastavica spuščena.

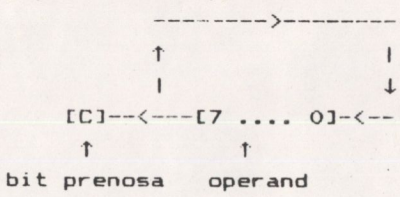
To človeka nekoliko zbega, a si boste zlahka zapomnili takole: če ima bit vrednost nič, je ničelna zastavica dvignjena, sicer ne.

Primer uporabe teh ukazov boste našli na koncu naslednjega poglavja.

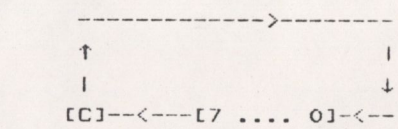
KROŽNI IN DRUGI POMIKI

Pomiki so naslednja skupina ukazov. Na voljo jih je kar nekaj vrst, ki pa jih morate dobro razlikovati, da jih boste znali pravilno uporabljati. Kot operande lahko povsod pri teh ukazih uporabimo vse registre ter (HL), (IX+ dis) in (IY+ dis), tj. vsebine naslovov HL, IX+ dis in IY+ dis. Krožni pomik je možen v levo ali v desno ter lahko vključuje 8 ali 9 bitov. Pri slednjem je v premikanje vključen tudi bit prenosa Carry). Za primer si oglejmo skici 8- in 9-bitnega krožnega pomika v levo (v desno je povsem enak, razlikuje se le po smeri):

8-bitni krožni pomik (brez prenosa)



9-bitni krožni pomik



Če označimo operand z "s", zapišemo ukaze za krožni pomik

RR s (ROTATE RIGHT s) - 9-bitni v desno;
RL s (ROTATE LEFT s) - 9-bitni v levo;
RRC s (ROTATE RIGHT WITHOUT CARRY s) - 8-bitni (brez prenosa) v desno;
RLC s (ROTATE LEFT WITHOUT CARRY s) - 8-bitni (brez prenosa) v levo.

Kot ste videli, se pri 9-bitnem krožnem premiku prenese najvišji bit (bit 7) operanda v bit prenosa, bit prenosa v bit 0 operanda itn. Pri 8-bitnem krožnem pomiku se bit 7 operanda prenese v bit prenosa in v bit 0 operanda. Vsi omenjeni ukazi

KROŽNI IN DRUGI POMIKI

Krajšava	Zlogov	Čas stanj	T	Učinek na zastavice C	Z	PV	S	N	H
RL r	2	8		*	*	*	*	0	0
RL (HL)	2	15		*	*	*	*	0	0
RL (IX+ dis)	4	23		*	*	*	*	0	0
RL (IY+ dis)	4	23		*	*	*	*	0	0
RLC r	2	8		*	*	*	*	0	0
RLC (HL)	2	15		*	*	*	*	0	0
RLC (IX+ dis)	4	23		*	*	*	*	0	0
RLC (IY+ dis)	4	23		*	*	*	*	0	0
RR r	2	8		*	*	*	*	0	0
RR (HL)	2	15		*	*	*	*	0	0
RR (IX+ dis)	4	23		*	*	*	*	0	0
RR (IY+ dis)	4	23		*	*	*	*	0	0
RRC r	2	8		*	*	*	*	0	0
RRC (HL)	2	15		*	*	*	*	0	0
RRC (IX+ dis)	4	23		*	*	*	*	0	0
RRC (IY+ dis)	4	23		*	*	*	*	0	0
RLA	1	4		*	-	-	-	0	0
RRA	1	4		*	-	-	-	0	0
RLCA	1	4		*	-	-	-	0	0
RRCA	1	4		*	-	-	-	0	0
SLA r	2	8		*	*	*	*	0	0
SLA (HL)	2	15		*	*	*	*	0	0
SLA (IX+ dis)	4	23		*	*	*	*	0	0
SLA (IY+ dis)	4	23		*	*	*	*	0	0
SRA r	2	8		*	*	*	*	0	0
SRA (HL)	2	15		*	*	*	*	0	0
SRA (IX+ dis)	4	23		*	*	*	*	0	0
SRA (IY+ dis)	4	23		*	*	*	*	0	0
SRL r	2	8		*	*	*	*	0	0
SRL (HL)	2	15		*	*	*	*	0	0
SRL (IX+ dis)	4	23		*	*	*	*	0	0
SRL (IY+ dis)	4	23		*	*	*	*	0	0

Oznake:
r = 8-bitni register 0 = zastavica se spusti (postane 0)
* = zastavica se spremeni - = zastavica ostane, kot je bila
Procesor Z80A: 7 T stanj = 2 mikrosekundi

vplivajo na vse zastavice.

Kadar želimo zavrteti vsebino registra A, lahko uporabimo krajšo in hitrejšo skupino ukazov. To so:

RRA, ki napravi isto kot RR A (presledek je bistven!);
RLA, isto kot RL A;
RRCA, isto kot RRC A ter
RLCA, ki napravi isto kot RLC A.

Zelo pomembna razlika med to skupino ukazov in prej omenjenimi pa zadeva zastavice. Ukazi "brez presledka" ne vplivajo na ničelno zastavico ter na zastavici prepolnjenja in znaka. Ponovim pa naj, da lahko z ukazi "brez presledka" obdelujemo samo register A.

Druga skupina, ki smo jo omenili v začetku, so tudi pomiki. Na voljo so nam tri skupine, ki uporabljajo iste operande kot ukazi pri krožnih pomikih:

SLA s (SHIFT LEFT ARITHMETIC s) - aritmetični pomik v levo

```
[C]<---[7 .... 0]<--- 0
 ↑          ↑
bit prenosa  operand
```

Pri tem ukazu se vsebina bita prenosa izgubi, bit 0 pa dobi vrednost 0. Z ukazom v bistvu pomnožimo operand z 2 (kaj se zgodi, če je v operandu 80H ?).

SRA s (SHIFT RIGHT ARITHMETIC s) - aritmetični pomik v desno

```
-->-
↑ ↓
--[7 .... 0]--->[C]
```

Ta ukaz deli vrednost v operandu z 2. Pri tem ohrani predznak, tako da lahko delimo števila v obsegu od - 128 do + 127.

SRL s (SHIFT RIGHT LOGICAL s) - logični pomik v desno.

```
0 --->[7 .... 0]--->[C]
```

Tudi to je - podobno kot SRA - deljenje z 2, ki pa ne ohrani predznaka (delimo lahko torej števila od 0 do 255).

UPORABA: Naslednji program bo na ekran izpisal število, ki ga boste naložili v register HL, v dvojiški obliki. Uporabljena

sta ukaz BIT in RL. BIT prebere vrednost bita 7 registra H, RL pa pomakne vrednost v levo. Če je ima bit 7 vrednost 1, se na zaslon izpiše 1, sicer 0. Pred klicem strojnega programa moramo uporabiti ukaz PRINT, zato ima izvajanje programa lepotno napako. Poleg tega moramo število, ki ga pretvarjamo, najprej neposredno naložiti v par HL. Poskusite napisati program v basicu (dolg bo le nekaj vrst), s katerim se bo pokazala prava uporabna vrednost strojnega dela (morda bo treba nekoliko spremeniti tudi sam strojni program).

```
21 XX XX        LD HL, XXXX     ;pretvarjana vrednost
06 10           LD B, 10H       ;število ima 16 bitov
CB 7C    Zanka  BIT 7, H        ;testiraj najvišji bit
3E 30           LD A, 30H       ;pripravi se na izpis "0"
28 01           JR Z, Piši      ;skoči, če je bit 7 enak 0
3C              INC A           ;sicer pripravi "1"
D7       Piši   RST 10          ;izpiši ustrezni znak
3E 20           LD A, 20H       ;20H je koda presledka
D7              RST 10          ;izpiši presledek
CB 15           RL L            ;pomakni L
CB 14           RL H            ;pomakni H in poberi prenos
10 EF           DJNZ Zanka      ;ponavljaj, dokler niso
C9              RET              obdelani vsi biti.
```


PREKINITVENI UKAZI

CP vsako 1/50 sekunde pogleda stanje posebne notranje zastavice IFF1 (Interrupt Flip Flop 1). Če je njena vrednost 1, prekine (interrupt) svoje delo in izvrši ukaz RST 38, s katerim kontrolira tipkovnico. S tem se delovna hitrost seveda nekoliko zmanjša. V posameznih delih strojnega programa pogosto ne potrebujemo kontrole tipkovnice, želimo pa hitro izvajanje. Pomagamo si z ukazom DI, ki pomeni "onemogoči prekinitve" (DISABLE INTERRUPTS) in ki postavi IFF1 na vrednost 0. Ponovno lahko prekinitve vključimo z EI, "omogoči prekinitve" (ENABLE INTERRUPTS). Ne pozabite dela programa, ki ste ga začeli z DI, končati z EI, sicer bo program izgubljen, ker nanj ne bo več mogoče vplivati!

Procesor Z80 pozna še nekaj ukazov (IM0, IM1, IM2), povezanih s prekinitvami. Njihova raba zahteva natančnejše poznavanje procesorja Z80, zato se ob njih ne bomo zadrževali. Vedoželjni bralec bo našel razlago teh ukazov v knjigi Mikroračunala (G. Smiljanič: Mikroračunala, Školska knjiga, Zagreb 1983).

PREKINITVE TER VHODNO / IZHODNI UKAZI

Krajšava	Zlogov	Čas stanj T	Učinek na zastavice C	Z	PV	S	N	H
DI	1	4	-	-	-	-	-	-
EI	1	4	-	-	-	-	-	-
IM 0	2	8	-	-	-	-	-	-
IM 1	2	8	-	-	-	-	-	-
IM 2	2	8	-	-	-	-	-	-
IN A, (n)	2	11	-	-	-	-	-	-
IN r, (C)	2	12	-	*	*	*	0	*
OUT A, (n)	2	11	-	-	-	-	-	-
OUT r, (C)	2	12	-	-	-	-	-	-
INI	2	16	-	*	?	?	1	?
IND	2	16	-	*	?	?	1	?
INIR	2	21/16	-	1	?	?	1	?
INDR	2	21/16	-	1	?	?	1	?
OUTI	2	16	-	*	?	?	1	?
OUTD	2	16	-	*	?	?	1	?
OTIR	2	21/16	-	1	?	?	1	?
OTDR	2	21/16	-	1	?	?	1	?

Oznake:

r = 8-bitni register
n = 8-bitno število (številka vhodno/izhodnih vrat)
* = zastavica se ob operaciji spremeni
0 = zastavica se spusti (postane 0)
1 = zastavica se dvigne (postane 1)
- = zastavica ostane, kot je bila
? = učinek ni znan

Pri avtomatičnih ukazih velja krajši čas v primeru, ko se operacija konča (B = 0)
Procesor Z80A: 7 T stanj = 2 mikrosekundi

VHOD IN IZHOD

Pridejo trenutki, ko CP potrebuje podatke od zunaj (na primer s tipkovnice ali s kasetofona) ali jih želi oddati navzven. Za CP je zunanji svet popolnoma tuje področje, na katerega se ne bo podal. Pripravljen je le odpreti vrata, skozi katera bo sprejemal ali oddajal. Vsi podatki, ki prihajajo ali odhajajo skozi vrata, so v obliki 8-bitnih števil.

Število vhodno-izhodnih naslovov (na kratko: v/i naslovov), s katerih lahko CP sprejema in na katere lahko oddaja, je veliko - 65535. A ker tudi različni naslovi mnogokrat pomenijo isto, je za naše delo pomembnejše dejansko število vhodno-izhodnih vrat (angleško input/output port). Teh pa je 256. Skoznje je CP povezan s tipkovnico, z vtičnicama EAR in MIC (ter preko njiju s kasetofonom), skoznje lahko vpliva na zvočnik in barvo roba ekrana. Preko robnega priključka pa se odpira pot do tiskalnika, vmesnikov (interfaceov) in drugih možnih priključkov.

Za vhod in izhod imamo dve skupini ukazov - "noter" (IN) ter "ven" (OUT), ki imajo splošno obliko:

IN A, (n) OUT A, (n)
IN r, (C) OUT r, (C)

"n" pomeni številko v/i vrat, ki jih želimo uporabiti. "r" pomeni katerikoli 8-bitni register; pri uporabi ukazov z "r" moramo številko vhodno-izhodnih vrat shraniti v register C.

Poglejmo zdaj, kako lahko dobimo podatke s tipkovnice. Do nje vodijo vrata FEH (= 254), zato bomo ukaz zapisali

IN A, (FE)

Morda se sprašujete, kako prikažemo 40 tipk računalnika v 8-bitnem zlogu. Odgovor je nekoliko presenetljiv: kot rezultat našega ukaza dobimo vedno podatek o le 5 tipkah naenkrat. Za katerih 5 tipk gre, pa določa vrednost, ki jo je imel register A ob začetku izvrševanja ukaza.

Tipkovnica je razdeljena v 8 skupin po 5 tipk:

3 =	(	1	2	3	4	5)	(	6	7	8	9	0	)	=	4
2 =	(	Q	W	E	R	T)	(	Y	U	I	O	P	)	=	5
1 =	(	A	S	D	F	G)	(	H	J	K	L	enter	)	=	6
0 =	(Caps shift	Z	X	C	V)	(	B	N	M	Sym shift	space	)	=	7	

Teh 8 skupin lahko predstavimo kot bite registra A - skupini 0 pripada bit 0, skupini 1 bit 1 itn. Bitu skupine, ki jo želimo čitati, damo vrednost 0, vsem ostalim bitom pa vrednost 1. Če želimo brati skupino (1 2 3 4 5), se pravi tretjo skupino, mora imet bit 3 registra A vrednost 0. Z drugimi besedami: pred branjem mora biti vrednost v registru A F7H

(A) = 1 1 1 1 0 1 1 1 = F7H (= 247).

Po izvršenem ukazu dobimo rezultat v registru A. Biti 5, 6 in 7 niso uporabljeni, vrednost bitov od 0 do 4 pa je odvisna od tega, katera tipka je pritisnjena. Najnižji bit ustreza najbolj zunanji tipki; če je tipka pritisnjena, ima bit vrednost 0, sicer 1. V našem primeru vpliva

tipka "1" na bit 0,
tipka "2" na bit 1,
tipka "3" na bit 2 in tako naprej.

Če bi se odločili za branje skupine 4, bi morali dati registru A vrednost EFH (= 239), tipke pa bi bile predstavljene takole:

"0" - bit 0 rezultata,
"9" - bit 1 rezultata itn.

Pustimo za trenutek tipkovnico in poglejmo pomembno dejstvo. Rekli smo, da ima CP na voljo 65535 v/i naslovov. Če dobro pogledate, moramo v naših ukazih v resnici vedno določiti 16-bitni naslov! Pri ukazu

IN A, (n)

je njegov višji del v registru A, nižji pa je številka v/i vrat (dejanski v/i naslov = 256 * A + n). Enako velja za niz

IN r, (C).

Tu je nižji zlog naslova (številka v/i vrat) v registru C, višji zlog pa moramo pred ukazom določiti v registru B (dejanski v/i naslov = 256 * (B) + (C)). Če bi torej v našem prejšnem primeru želeli uporabiti ta niz ukazov, bi dali registru C vrednost FEH, z registrom B bi pa navedli, katero skupino želimo čitati.

Res je, da višji del naslova v teh ukazih mnogokrat nima nobenega vpliva in je odločilna le številka v/i vrat (posledico smo že omenili: različni naslovi lahko pomenijo isto). Vendar kljub temu ne pozabite: v bistvu morate vedno določiti 16-bitni naslov.

Vrnimo se zdaj k tipkovnici. Marsikdaj bi (predvsem pri

igrah) želeli naenkrat prečitati celo zgornjo vrsto (skupini 3 in 4). To lahko storimo tako, da damo registru A vrednost

1 1 1 0 0 1 1 1 = E7H (= 231).

Kot vidite, sta tako bit 3 kot bit 4 enaka 0. Na ta način se podatki sicer pomešajo in ne morete vedeti, ali je bila pritisnjena tipka "1" ali "0", "2" ali "9" itn., ker ti pari vplivajo na iste bite. Uporaben pa je ta način, kot smo rekli, zlasti za igre, ker lahko npr. tipki "5" in "8" služita za pomikanje v levo in desno, čeprav pripadata različnima skupinama.

Uporaba ukazov IN ter OUT je najhitrejši način branja tipkovnice. Priznam pa, da se vam po pravici zdi precej okoren. Zato za branje tipk pogosto uporabljamo podprograme iz ROM-a, ki vam bodo gotovo bolj všeč. O njih bomo govorili v poglavju "Uporaba tipkovnice".

Ukaza IN in OUT nam dajeta veliko moč. Z njima lahko pošiljate in/ali sprejemate tudi podatke (se pravi električne impulze) z robnega priključka. Na ta način lahko upravljate vse, kar je priključeno na vaš računalnik - najsi bo to električna železnica, gospodinjski aparati ali vaš domači disko klub.

V tabeli na začetku poglavja ste opazili še ukaze

INI (IN AND INCREASE = noter in povečaj),
IND (IN AND DECREASE = noter in zmanjšaj),
INIR (IN, INCREASE AND REPEAT = noter, povečaj in ponovi),
INDR (IN, DECREASE AND REPEAT = noter, zmanjšaj in ponovi),
OUTI (OUT AND INCREASE = ven in povečaj),
OUTD (OUT AND DECREASE = ven in zmanjšaj),
OTIR (OUT, INCREASE AND REPEAT = ven, povečaj in ponovi) ter
OTDR (OUT, DECREASE AND REPEAT = ven, zmanjšaj in ponovi).

To so skupinski ukazi za vhod in izhod, in sicer štirje navadni (INI, IND, OUTI, OUTD) in štirje avtomatični (INIR, INDR, OTIR, OTDR). Ti ukazi napravijo IN (HL), (C) oz. OUT (C), (HL), povečajo oz. zmanjšajo HL ter zmanjšajo B. Avtomatični ukazi se ponavljajo, dokler B ne doseže vrednosti 0. Skupinski ukazi za vhod in izhod so zelo malo v rabi, zato se jim ne bomo natančneje posvečali.

UPORABA: Z rabo v/i ukazov se bomo pozabavali v poglavju Uporaba tipkovnice ter v igri "Ključar Martin in vražji metulj.

PRIPRAVA PROGRAMA

Dobro smo se že seznanili s strojnim jezikom - orodjem za izdelavo naših programov. A to ni vse! Tudi za gradnjo ni dovolj, da poznamo orodje in znamo zlagati opeke. Hiša, zidana brez načrta bo hladna, grda in se bo kmalu podrla. Enako je s programi! Nikoli ne smemo graditi brez načrta. Kako torej zastaviti delo, da bo program dober, pravilen in razumljiv?

Pisanje programov v basicu vam je gotovo domače. Programe pogosto vnašamo kar "iz glave", a so navadno kljub temu dovolj uspešni. Nadzorni program nas opozori na napake, dolžina programov pa je navadno tolikšna, da so še pregledni in hitro razumljivi. V strojnem jeziku je drugače. O napakah ni nobenih obvestil, spodrsljaji se končajo slabo. Programi so dolgi in (brez komentarja) zelo težko razumljivi. Zaradi tega priprava strojnih programov ne sme biti le "pisanje programov", temveč pravo programiranje. A kaj pravzaprav je programiranje? To je zbirka opravil, ki jih navadno razdelimo v tri stopnje:

- načrtovanje programa in logično preizkušanje načrta,
- kodiranje ter
- preizkušanje in popravljanje.

Najpomembnejša in časovno najdaljša je prva stopnja. Kolikor bolje je izdelan načrt, toliko manj dela in časa porabimo kasneje za ostala opravila.

NAČRTOVANJE

Program je nemogoče oblikovati naenkrat. Tega pravila ne smemo pozabiti. Marsikdo se loti izdelave strojnega programa kar "na pamet". Tako nastajajo programi s številnimi spodrsljaji, ki jih nato množica popravkov in dopolnitev spremeni v nepregledno gmoto ukazov. Zato ne moremo dovolj toplo priporočiti vestne priprave načrta na način, ki ga imenujemo *od vrha navzdol*. Nalogo (naj bo igra ali uporabni program), ki jo moramo rešiti, razdelimo na več manjših korakov. Zaporedje vseh korakov, ki vodijo do rešitve, imenujemo algoritem.

Denimo, da bi želeli izdelati program vožnje na avtomobilski dirki. Algoritem bi na prvi mah izgledal nekako tako:

Čakaj Če ni pritisnjena nobena tipka, pojdi na *Čakaj*.
Vožnja Premakni avto.
Preveri smer in hitrost.
Če smer ali hitrost nista pravi, pojdi na *Trk*.
Če vozilo še ni v cilju, pojdi na *Vožnjo*.
Cilj Izpiši doseženi čas in uvrstitev.
Pojdi na *Čakaj*.
Trk Izpiši, da se je zgodila nesreča.
Pojdi na *Čakaj*.

Kot vidite, so vsa navodila zapisana v navadni slovenščini. To je zlato pravilo: algoritem naj ostane čim dlje zapisan v običajnem jeziku. Nismo se še odločili, ali bo program v basicu, strojnem ali kakšnem drugem jeziku. Takšna odločitev bi nas trenutno le ovirala - zamisel in načrt programa nista odvisna od jezika, v katerem bomo zapisali program.

Ko imamo korake zapisane, jih še enkrat pregledamo. Smo kaj pozabili? Je kakšen korak odveč? Katere korake lahko oblikujemo kot podprograme? Tak pregled imenujemo tudi logično preizkušanje. Preglejmo naš program! Zanemarili smo navodila zato dodajmo na začetku korak, ki jih bo napisal. Če dobro pogledamo, ni nikjer koraka, v katerem bi končali program. Dodajmo torej korak *Konec*. Zaradi tega moramo spremeniti še zadnja dva koraka:

Cilj Izpiši doseženi čas in uvrstitev.
Pojdi na *Konec*.
Trk Izpiši, da se je zgodila nesreča.
Pojdi na *Konec*.
Konec Vprašaj, ali želi igralec končati igro.
Če ne želi, pojdi na *Čakaj*.
Končaj.

Z algoritmom smo zdaj zadovoljni in lahko nadaljujemo z razčlenitvijo posameznih korakov. *Konec* bi natančneje zapisali:

Konec Obriši zaslon.
Izpiši "Ali želiš končati?"
Preleti tipkovnico.
Če je pritisnjen "d", končaj.
Pojdi na *Čakaj*.

To so že precej natančna navodila, kar tudi želimo. Drugo zlato pravilo je namreč: koraki naj bodo majhni. Vsebujejo naj kar najmanj odločitev (pogojnih skokov, pogojnih klicev ipd.) - če je mogoče, le eno. Z razčlenitvijo dobimo iz posameznih korakov samostojne dele, ki jih lahko tudi samostojno preizkušamo (spomnite se podprogramov!). In še tretje pravilo: vse podrobnosti prihranimo za konec.

V načrtu oblikujemo eno glavno vejo (glavni program) in več stranskih. Glavni program naj vsebuje ukaze oziroma opravila, ki bodo največkrat izvajani. Stranske veje naj obsegajo ostala, sorazmerno redkeje potrebna opravila. Paziti je treba, da v glavnem programu ni korakov, ki jih lahko uvrstimo v stranske veje. Izvajanje programa bo tako hitrejše, razumevanje pa lažje.

Ko pripravljamo obsežnejše programe, pride prav "dnevnik". To so zapiski sprememb, pripomb, vprašanj... Delo ter kasnejše prebiranje in izdelava komentarja so na ta način lažji.

KODIRANJE

Opazili ste, da smo posameznim korakom dali imena - pravimo jim tudi simbolični naslovi (npr. Cilj, Konec). Ta kratka imena, ki povedo, kaj se dogaja v posameznem koraku, so zelo praktična že pri izdelavi algoritma. Ko kodiramo, moramo te naslove nadomestiti z absolutnimi števili - naslovi v pomnilniku. Pri delu z zbirnikom lahko nadomeščamo posredno, če pa uporabljamo urejevalnik, moramo simbolične naslove takoj (neposredno) nadomestiti z absolutnimi. Pri neposrednem nadomeščanju je treba paziti na pravi vrstni red! Nizki zlog naslova je v pomnilniku prvi, visoki zlog drugi. Pomote pri zapisu so pogost vir napak.

Druga "neprijetna" skupina so pri kodiranju relativni skoki. Pri izračunu odmikov je potrebna precejšnja pazljivost (poglejte še enkrat poglavje o skokih). Poleg tega je treba pri dopolnjevanju programa (dodajanju ali odvzemanju ukazov) preveriti, ali nismo spremenili odmika kakega kratkega skoka.

KAM SPRAVITI PROGRAM ?

Vprašanje, kam spraviti program v strojnem jeziku, je bilo pri nekaterih računalnikih (npr. ZX 81) velik problem. Za ZX Spectrum tega nikakor ne moremo trditi. Prostorov, kamor lahko shranjujemo strojni kod, je celo več.

PROSTI POMNILNIK

Z ukazom CLEAR nn rezerviramo pomnilniški prostor od naslova "nn+1" naprej. Na ta način nam je popolnoma na voljo del pomnilnika, zavarovan pred vplivi programa v basicu in pred posegi nadzornega programa. Najbolje je postaviti programe tik pod vrh pomnilnika. Tako jih ne bo potrebno premikati, kadar bomo uporabljali večje podatkovne zbirke ali kadar bomo dodajali nove programe. Takšen način shranjevanja - nad mejo basicovega področja - je najbolj enostaven in brez vsakršnih preglavic. Zato vam ga najbolj priporočamo.

Poleg te možnosti je seveda še nekaj drugih. Oglejmo si jih!

UPORABA VRSTICE REM

Prva takšna možnost so REM vrstice programa v basicu. Če je REM prva vrstica programa v basicu, lahko znake v njej nadomestimo z zlogi strojnega programa. Recimo, da je ta 40 zlogov dolg. Kot prvo vrsto v basicu napišemo

```
1  REM 1234567890123456789012345678901234567890
```

Vrsta ima 40 znakov. Če bi imel naš strojni program 60 zlogov, bi potrebovali 60 znakov; če bi imel le tri zloge, bi v vrstico lahko zapisali le tri. Popolnoma vseeno je tudi, katere znake uporabimo. Prednost našega načina je, da natančno vemo, koliko znakov je v vrstici. Tako le stežka pride do napak.

Uporaba REM vrste ima svoje prednosti in pomanjkljivosti. Poglejmo najprej pomanjkljivosti.

Prva in najhujša je, da je ta način primeren le za kratke programe, ki jih ne bomo več dopolnjevali ali popravljali. Vrste s strojnim kodom največkrat ni mogoče izpisati na zaslon z ukazom LIST, pogosto tudi ni mogoče uporabiti ukaza EDIT (Caps shift + 1). S tem je popravljanje oz. dopolnjevanje skoraj onemogočeno. Vzrok so nekateri kodi (npr. - desetiško - 10, 6, 13 in drugi), ki imajo čudne učinke na izpis.

Z začetnim naslovom programa v REM-u včasih ni bilo težav. (naslov prvega zloga za znakom REM je bil 23760). Zdaj pa, ko na Spectrum priključimo mikrotračno enoto, se program v basicu (s tem pa tudi REM vrsta ter strojni program) med obdelavo pomika v pomnilniku navzgor in navzdol. S tem je uporaba absolutnih naslovov onemogočena, programiranje pa zelo okrnjeno. Za uporabo programa v REMu si je treba v tem primeru pomagati z ukazoma DEF FN ter FN in z uporabo sistemske spremenljivke PROG izračunati dejanski naslov strojnega programa. Med manjše pomanjkljivosti prištejmo še, da bo ukaz NEW seveda izbrisal tudi REM s strojnim programom vred (kar se ne zgodi, če je program nad RAMTOP).

Prednost uporabe REM vrstice je, da je strojni program tako del programa v basicu. Zato je vedno dovolj le en LOAD, SAVE ali VERIFY ukaz. Vendar to ne odtehta omenjenih težav, zato je shranjevanje strojnega koda v REM stavke le redkokdaj ustrezna rešitev.

UPORABA PROGRAMSKEGA PROSTORA

Prednosti in pomanjkljivosti takšnega shranjevanja so skoraj enake kot za REM stavke. Razliki: ni težav pri uporabi ukaza LIST, pa tudi možnost shranjevanja nekoliko daljših strojnih programov je laže izvedljiva. Ker je v tem primeru začetni naslov strojne kode popolnoma spremenljiv (odvisen od dolžine basic programa) tudi brez uporabe mikrotračnih enot, je ta način shranjevanja še nekoliko manj priporočljiv kot raba REM vrstic. Zato ga le omenjamo in se z njim ne bomo natančneje ukvarjali.

UPORABA TISKALNIKOVEGA VMESNEGA POMNILNIKA

Tiskalnikov "buffer" je 256 zlogov dolgo področje, ki sega od naslova 23296 do 23551. Kadar ne uporabljamo tiskalnika, sameva ta prostor prazen in neizkoriščen. Zato je precej uporaben za shranjevanje - pod pogojem, da je strojni program krajši od 256 zlogov ter da ne uporabljamo tiskalnika ali paralelnega vmesnika. Vsaka uporaba ukazov LPRINT, LLIST ali COPY namreč izprazni tiskalnikov prostor. Sicer pa se ta način ne razlikuje dosti od shranjevanja v pomnilniku nad RAMTOPom. Prednost je le, da nam tako ostane na voljo ves drug pomnilniški prostor.

POSREDOVANJE PODATKOV STROJNEMU PROGRAMU

Zdaj pa nekaj popolnoma drugega. Oglejte si ukaz

```
DEF FN F(X) = USR 29340
```

Recimo, da se na naslovu 29340 začenja nek strojni program. Kadarkoli bomo v basicu uporabili ukaz FN F(X), bomo z njim pognali ta strojni program. "Lepo in prav," boste rekli, "a kakšno korist imamo od tega ? Zakaj ne bi še naprej uporabljali dobrega starega načina RANDOMIZE USR 23340 ?" No, pomislite na X v oklepaju - kaj se dogaja z njim ?

Ena od sistemskih spremenljivk (poglejte 25. poglavje priročnika za Spectrum), ki se imenuje DEFADD, shranjuje naslov spremenljivk, ki so navedene v ukazu FN. Primer:

```
DEF FN F(A,B,C,A$,B$,C$) = USR naslov
```

Uporabljenih je 6 spremenljivk in nekje v pomnilniku je spravljenih 6 skupin podatkov o teh spremenljivkah. DEFADD hrani naslov prve skupine podatkov. Poglejmo, kako so te skupine sestavljene:

```
                [ A ][   ][   1. spremenljivka    ][ , ]
1.skupina       [41 ][0E ][00 ][00 ][aa ][aa ][00 ][2C ]
                  ↑
                (DEFADD)
                [ B ][   ][   2. spremenljivka    ][ , ]
2.skupina       [42 ][0E ][00 ][00 ][bb ][bb ][00 ][2C ]
                [ C ][   ][   3. spremenljivka    ][ , ]
3.skupina       [43 ][0E ][00 ][00 ][cc ][cc ][00 ][2C ]
                [ A ][ $ ][   ][   ][ naslov ][dolžina ][ , ]
4.skupina       [41 ][24 ][0E ][?? ][   ][   ][   ][   ][2C ]
                [ B ][ $ ][   ][   ][ naslov ][dolžina ][ , ]
5.skupina       [42 ][24 ][0E ][?? ][   ][   ][   ][   ][2C ]
                [ C ][ $ ][   ][   ][ naslov ][dolžina ][ ) ]
6.skupina       [43 ][24 ][0E ][?? ][   ][   ][   ][   ][29 ]
```

Podatki o številčnih spremenljivkah obsegajo 8 zlogov. Prvi zlog vsebuje ime spremenljivke, drugi ima vedno vrednost 0E, zadnji pa 2C oz. 29H. Ti trije zlogi so za nas nepomembni. Važnih je preostalih 5 zlogov. V njih je shranjena vrednost spremenljivke. Če je to decimalno število, je zapisano v plavajoči vejici (podrobneje o tem v poglavju Aritmetika s plavajočo vejico). Če pa je vrednost celo število, so 3., 4. in 7. zlog enaki 0, v 5. zlogu je nizki, v 6. pa visoki zlog števila. Podatki o znakovnih spremenljivkah zasedajo 9 zlogov. V prvem in drugem je ime spremenljivke, prav tako so za nas neuporabni 3., 4. in 9. zlog. Važni pa so preostali 4 zlogi. V 5. in 6. je naslov znakovne spremenljivke (se pravi: naslov besede) v 5. nizki, v 6. visoki zlog naslova. V zlogih 7 in 8 pa je dolžina besede (število znakov) - v 7. nizki, v 8. visoki zlog.

Bistvo in uporabnost vsega tega ? Če v basicu napišete DEF FN F(X) = USR naslov in nato uporabite FN F(spremenljivka), boste s tem pognali strojni program. Ta pa bo lahko uporabil spremenljivko, ki ste jo navedli v oklepaju. Prebral jo bo z zaporedjem ukazov

```
2A5C0B          LD HL, (DEFADD)      ;DEFADD => 23563
23              INC HL
23              INC HL
23              INC HL
23              INC HL
4E              LD C, (HL)
23              INC HL
46              LD B, (HL)
```

Zdaj, ko smo se seznanili z zamislijo posredovanja podatkov strojnemu programu si oglejmo še drugi način, ki je pravzaprav veliko boljša možnost kot pravkar opisani. Prva vrstica programa v basicu naj bo DIM X (N), kjer je N število podatkov, ki jih moramo prenesti programu v strojnem jeziku. V sistemski spremenljivki VARS bo v tem primeru shranjen začetni naslov polja X, naslov prve vrednosti pa bo za šest večji od začetnega naslova. Če je polje znakovno (tj. če smo definirali DIM X$(N)), bo vsaka vrednost zasedala en zlog. Če je polje številčno (tj. DIM X(N)), pa je vsaka vrednost zapisana v 5-složni obliki (glej poglavje "Aritmetika s plavajočo vejico"). Izračun naslova posamezne vrednosti in dostop do nje je tako lažji in lepši. Če moramo posredovati večje število vrednosti, pa je uporaba polja tudi dosti manj zamudna.

IZPISOVANJE NA ZASLON

Prav gotovo že nestrpno čakate, da bodo vaši napori v strojnem jeziku obrodili sad. Rezultati doslej obravnavanih programov so bile v glavnem zgolj številke - in verjamem, da vam zanje kljub vsemu ni dosti mar. Zato se bomo zdaj lotili naloge, ki bo dala VIDNE plodove: izpisovanja na zaslon.

Pisanje na ekran je v strojnem jeziku precej enostavno, celo nekoliko podobno kot v basicu. Uporabljamo ukaz RST 10, ki izpiše znak, katerega kod je v registru A. Preskusite naslednji program:

```
AF              XOR A                ;= LD A, 0
323C5C          LD (TV FLAG), A
3E2A    Zanka   LD A, 2A
D7              RST 10
18FB            JR Zanka
```

Kot vidite smo morali v sistemsko spremenljivko TV FLAG (najdete jo v 25. poglavju priročnika za Spectrum) na začetku naložiti vrednost 0 (kaj če tega ne bi naredili? Preskusite program brez prvih dveh vrstic!). Ko boste pognali program, se bo ekran napolnil z zvezdicami in to zelo zelo hitro (mnogo hitreje kot z "10 PRINT "*";:GO TO 10" v basicu).

Vendar je to šele prvi korak. Mi pa želimo imeti (pod)program, ki bi nam izpisal kakršnokoli besedilo - od "DA" do "Kdor visoko leta, nizko pade", ki bi še rumeno-rdeče utripalo. Recimo, da takšen podprogram že obstaja in se imenuje S_PRINT. Uporabljati ga želimo v obliki: CALL S_PRINT s podatki "PAPIR rumen ČRNILO rdeče UTRIPANJE vključeno Kdor visoko leta, nizko pade" (besede z velikimi tiskanimi črkami uporabljamo namesto ustreznih ukazov v basicu). Poglejmo primer:

```
CDXXXX          CALL S_PRINT
5A647261766F    DEFM Zdravo
00              DEFB 00
```

DEFM (DEFINE MESSAGE = označi sporočilo) in DEFB (DEFINE BYTE = določi zlog) nista prava strojna ukaza. Uporabljamo ju le v

zbirnem zapisu, da z njima označimo podatke. Če pogledate v tabele, boste videli, da je 5A kod znaka "Z", 64 znaka "d", 72 znaka "r" itd. Z DEFB 00 smo označili konec našega "besedila" - se pravi konec podatkov, ki naj jih uporabi S_PRINT. Seveda CP podatkov ne sme obravnavati kot strojne ukaze. Zato moramo program napisati tako, da se kaj takega ne zgodi. Spomnite se, kaj se dogaja ob klicu strojnega podprograma (o tem smo govorili v poglavju "Podprogrami"). Ker beseda "Zdravo" sledi ukazu CALL, bo na skladu spravljen naslov znaka "Z". Ta naslov lahko snamemo z ukazom POP - npr. POP HL. Če zdaj povečamo HL in napravimo PUSH, se bo CP ob koncu podprograma vrnil na naslednji naslov, tj. na naslov znaka "d". V našem podprogramu se torej lahko ognemo izvajanju podatkov tako, da ponavljamo POP in PUSH dokler ne dosežemo konca podatkov, tj. zloga z vrednostjo 0 (ki nikoli ne more biti del besedila). Pazljivo si oglejte tale podprogram:

```
E1  S_PRINT  POP HL
7E           LD A, (HL)
23           INC HL
E5           PUSH HL
A7           AND A
C8           RET Z
D7           RST 10
18F7         JR S_PRINT
```

Prve štiri vrstice pogledajo znak, spravljen na trenutnem naslovu za povratek. Ob tem tudi pomaknejo ta naslov za en zlog naprej (INC HL). Naslednji dve vrstici nadzorujeta, ali je že konec podatkov. Zapomniti si velja uporabo ukaza AND A za primerjanje A z 0 (s tem prihranimo 1 zlog). Če ima A vrednost 0 (tj. če smo prišli do konca podatkov), bo CP izvedel ukaz RET Z in se vrnil na zlog, ki sledi zlogu 00. Če A še ni 0, bo CP uporabil ukaz RST 10 in izpisal ustrezni znak na zaslon. Postopek se ponavlja, dokler ni konec besedila.

Zdaj pride na vrsto naš urejevalnik. Za start uporabimo 28672 (7000H) in najprej vpišite podprogram S_PRINT, takoj za njim pa še nekoliko spremenjen glavni program:

```
AF      ST   XOR A
323C5C       LD (TV FLAG), A
```



```
CD0070       CALL S_PRINT
1106100212014B646F72207669736F6B6F206C6574612C
206E697A6B6F2070616465  DEFM "besedilo"
00           DEFB 00
C9           RET
```

"START ZA USR" naj bo ST (= 28681). Poženite program! Za začetek kar lepo, ali ne?

RAZISKOVANJE ZASLONA

Lotimo se naše naloge še z druge strani. Preizkusimo, kaj se zgodi, če v basicu zapišemo:

```
FOR I= 16384 TO 23295: POKE I, 41: NEXT I: PAUSE 0
```

Se vam zdi nenavadno? Ne le to, da smo vplivali na ekran, ampak predvsem kako se je to dogajalo. Vzrok je predstavitev zaslona, kakor jo v svojem pomnilniku hrani Spectrum - in ki se zdi bolj zapletena, kot bi bilo potrebno. Zaradi tega je koristno, da zaslon nekoliko raziščemo. Ugotovili bomo, da je njegova predstavitev v pomnilniku veliko bolj smiselna, kot je videti na prvi pogled.

Oglejmo si sliko 1. Iz nje razberemo naslov kateregakoli "PRINT" položaja na ekranu. Prav tako lahko iz nje razberemo naslove zlogov, ki določajo prilastke (barvo papirja, črnila, svetlost in utripanje) posameznih PRINT položajev (več o prilastkih na strani 116). Poskusimo s pomočjo slike določiti naslov mesta, na katerega bi pisali z ukazom PRINT AT 5,4 v basicu. Prva dva znaka naslova sta 40. Tretji znak je A (ker uporabljamo šesto vrstico), četrti pa 4. Tako pridemo do naslova 40A4. Naslov prilastkov dobimo tako, da za prvi dve številki vzamemo znaka v oklepaju. V našem primeru bi bil torej naslov zloga s prilastki 58A4. Zdaj lahko preizkusimo dva programčka:

```
21A440       LD HL, 40A4
36FF         LD (HL), FF
C9           RET
```

```
21A458       LD HL, 58A4
36C7         LD (HL), C7
C9           RET
```

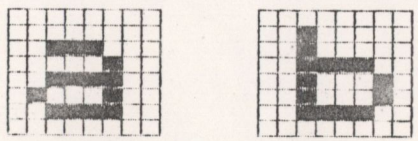
četrta številka
↓
0 1 2 3 4 5 6 7 8 9 A B C D E F 0 1 2 3 4 5 6 7 8 9 A B C D E F

0 2 4 6 8 A C E 40 (58) - prvi dve številki naslova na zaslonu ter naslova prilastkov - tretja številka	1 3 5 7 9 B D F
0 2 4 6 8 A C E 48 (59)	1 3 5 7 9 B D F
0 2 4 6 8 A C E 50 (5A)	1 3 5 7 9 B D F

SLIKA 1. ZASLONSKA DATOTEKA

Iz prej povedanega je že lažje razložiti ta dva programa in njun rezultat. S tem, zakaj smo v prvem uporabili FF in v drugem C7, pa se bomo ukvarjali malo kasneje.

Zelo zanimiv del ROM-a se skriva na njegovem koncu, med naslovoma 3D00 in 3FFF. V teh 3/4 kb niso programi, temveč dolga tabela. V njej so spravljeni točkovni vzorci vseh znakov, razen grafičnih. Za vsak znak je potrebnih 8 zlogov; "a" in "b" npr. sta shranjena takole: 00 00 38 04 3C 44 3C 00 in 00 20 20 3C 22 22 3C 00. Zapišite ta števila eno nad drugim v dvojiški obliki in nato poglejte sliko 2 ! Zdaj vidite, da je vzorec črk enak razporedu enojk med ničlami v dvojiškem zapisu. Znaki imajo na zaslonu prav takšno obliko. Naslov znaka (se pravi: njegovega vzorca), ki ima kod X ("a" ima kod 61H), izračunamo takole: 8*X + 3C00 (naslov vzorca "a" je torej 8*61 + 3C00 = 3F08).



SLIKA 2. TOCKOVNA VZORCA ČRK a IN b.

Spet je pred nami program. Poskusimo najprej ugotoviti, kaj naredi in ga šele potem praktično preizkusimo:

```
21A440          LD HL, 40A4       ;koordinate položaja 5,4
11083F          LD DE, 3F08       ;naslov vzorca znaka A
0608            LD B, 08
1A      Zanka   LD A, (DE)
77              LD (HL), A
24              INC H             ;pomembno: INC H, ne INC HL
13              INC DE
10FA            DJNZ Zanka
C9              RET
```

Ste zadeli? Če je bilo vse v redu, se je na položaju 5,4 zapisala črka a.

Poglejmo še enkrat peto vrstico programa. Rekli smo INC H. Ta kratek in hiter ukaz je vzrok za na videz nesmiselno zapleteno predstavitev zaslona v pomnilniku. Če je naslov "PRINT" položaja HL, potem ima 8 zlogov, kamor zapišemo znak, naslove HL, HL+0100, HL+0200, HL+0300 itd. do HL+0700. Z ukazom INC H dosežemo isto kot če bi imeli na voljo ukaz ADD HL, 0100. Zdaj pa si predstavljajte, kaj bi se zgodilo, če bi bil zaslon v pomnilniku predstavljen na naslovih, ki bi tekli lepo eden za drugim od levega zgornjega do desnega spodnjega kota. V tem primeru bi morale biti slike enega znaka na naslovih HL, HL+0020, HL+0040, HL+0060 itd. do HL+00E0. Ker ukaz ADD HL,nn ne obstaja, bi bil edini način za pomik na naslednji položaj PUSH DE ; LD DE, 0020 ; ADD HL, DE ; POP DE. To je precej daljše (in seveda veliko počasneje) kot INC H. Tako se razporeditev, ki jo je marsikdo imel za nespametno, izkaže kot zelo domiselna.

Kljub temu še ostane nekaj težav. Zaslon je razdeljen v tri tretjine in problem se pojavi, ko hočemo preiti iz ene tretjine v drugo. Primer: če je naslov nekega položaja HL, je je naslov naslednjega položaja **običajno** HL + 0001. Če pa ima ima HL slučajno vrednost 40FF, potem je naslednji položaj 4800 - ne 4100 (kar je druga vrstica prvega "PRINT" položaja na ekranu). Da bi se vedno pomaknili na naslednji položaj, moramo uporabiti tale program:

```
CB1C        RR H
CB1C        RR H
CB1C        RR H
23          INC HL
CB14        RL H
CB14        RL H
CB14        RL H
```

Podobno si pomagamo pri pomikih navzdol. V ta namen uporabimo program RR H; RR H; RR H; LD BC, 20H; ADD HL, BC; RL H; RL H; RL H.

Oglejmo si zdaj še zloge s prilastki in ugotovimo, kako nam lahko koristijo. Vsak "PRINT" položaj ima le en zlog za prilastke (kot se spomnite jih ima za izpis znakov osem). Zlog za prilastke shranjuje podatke o barvi papirja, črnila, svetlosti in utripanju za določen položaj. Če označimo papir s P, črnilo s Č, utripanje z U in svetlost s S, lahko zlog s prilastki v dvojiški obliki zapišemo

U S P2 P1 P0 Č2 Č1 Č0

P2, P1 in P0 predstavljajo številko barve papirja (v dvojiški obliki), Č2, Č1 in Č0 pa barve črnila (prav tako dvojiško). Utripanju in svetlosti, ki lahko zavzameta samo dve vrednosti, je namenjen po en bit. Uporabimo naše ugotovitve na preprostem podprogramu, ki bo barvo črnila na določenem položaju spremenil tako, da bo kontrastna barvi papirja. Zlog s prilastki moramo pred začetkom naložiti v register A.

```
CB6F        BIT 5, A
2803        JR Z, Bela
E6F8        AND F8              ;napravi črnilo črno
C9          RET
F607  Bela  OR 07               ;napravi črnilo belo
C9          RET
```

V bitu 5 registra A je v našem primeru spravljen drugi bit barve papirja. Če je ta temna (črna, modra, rdeča ali vijolična), bo imel bit vrednost 0, če pa je barva papirja svetla (zelena, sinja, rumena ali bela), bo vrednost bita 1 - prepričate se lahko sami: zapišite številke barv dvojiško. Gotovo vam ni težko razvozlati, kako program ugotovi potrebno barvo črnila in kako jo spremeni.

Zdaj pa k nekoliko bolj zapletenemu primeru. Gre za podprogram, ki premakne HL na nov položaj na zaslonu, DE pa na ustrezni novi zlog s prilastki. Preden podprogram pokličemo, mora HL kazati na prvo vrstico nekega položaja na ekranu, DE na ustrezni zlog s prilastki, BC pa mora vsebovati vrednost premika - FFFF za eno mesto na levo, 0001 za eno mesto na desno, FFE0 za eno mesto navzgor ter 0020 za eno mesto navzdol (tj. vrednost premika je enaka številu mest). Podprogram vsebuje zanimivo kontrolo, ki preprečuje "odhod" z zaslona:

```
7D   PREMIK  LD A, L
CB1C         RR H
CB1C         RR H
CB1C         RR H
09           ADD HL, BC
CB14         RL H
CB14         RL H
CB14         RL H
EB           EX DE, HL
09           ADD HL, BC
EB           EX DE, HL
AD           XOR L
CB67         BIT 4, A
C8           RET Z
37           SCF
CB5F         BIT 3, A
C8           RET Z
7D           LD A, L
17           RLA
AD           XOR L
CB67         BIT 4, A
C0           RET NZ
37           SCF
C9           RET
```

V programu je predvidenih nenkrat največ sedem mest premika v vsaki smeri. Varnostni dodatek, ki smo ga omenili, spusti zastavico prenosa, če je premik dopusten. Če bi premik povzročil prehod roba zaslona, se zastavica prenosa dvigne. Velja si tudi zapomniti, kako smo uporabili ukaz EX DE, HL, da smo lahko prišteli BC k DE. To pride dostikrat zelo prav.

Logika našega programa je takšna:
če je bit 4 nespremenjen, je premik dopusten. Sicer...
če je bit 3 nespremenjen, premik NI dopusten. Sicer...
če bit 3 ni enak bitu 4, je premik dopusten. Sicer...
premik NI dopusten.

Prepuščam vam, (1) da se sami poglobite v podrobnosti programa ter (2) da premislite, kje vam bo prišel prav.

Za konec še dva programa, ki vas bosta verjetno zanimala. Gre za premikanje slike (scroll); prvi program bo povzročil pomikanje navzgor, drugi navzdol. "PRINT" položaj se ob tem ne spremeni, izbrišemo pa zgornjo ali spodnjo vrstico. Slika 1 vam bo za razumevanje obeh programov v veliko pomoč.

Prvi del: pomikanje navzgor. Program lahko spravite kamorkoli v pomnilnik, START ZA USR pa mora dobiti vrednost naslova GOR. Števila, označena s #, so zapisana šestnajstiško, ostala desetiško. Kod je kot vedno šestnajstiški (čeprav je brez #).

```
C5      POMIK PUSH BC
D5            PUSH DE
E5            PUSH HL
EDB0          LDIR
E1            POP HL
24            INC H
D1            POP DE
14            INC D
C1            POP BC
C9            RET
212040 GOR    LD HL,#4020
110040        LD DE,#4000
01E01A        LD BC,#1AE0
EDB0          LDIR
21E047        LD HL,#47E0
11E040        LD DE,#40E0
0E20          LD C,#20
3E10          LD A,#10
CDXXXX ZANKA  CALL POMIK
3D            DEC A
20FA          JR NZ, ZANKA
62            LD H, D
1EE1          LD E,#E1
0B            DEC BC
```



```
3E08          LD A,#08
3600    BRISI LD (HL),#00
CDXXXX        CALL POMIK
3D            DEC A
20F8          JR NZ, BRISI
265A          LD H,#5A
54            LD D,H
3A485C        LD A,(BORDCR)
77            LD (HL),A
EDB0          LDIR
C9            RET
```

Drugi del: pomikanje navzdol. Navedite naslov DOL za izvedbo START ZA USR.

```
C5      POMIK PUSH BC
D5            PUSH DE
E5            PUSH HL
EDB8          LDDR
E1            POP HL
25            DEC H
D1            POP DE
15            DEC D
C1            POP BC
C9            RET
21DF5A DOL    LD HL,#5ADF
11FF5A        LD DE,#5AFF
01E01A        LD BC,#1AE0
EDB8          LDDR
211F50        LD HL,#501F
111F57        LD DE,#571F
0E20          LD C,#20
3E10          LD A,#10
CDXXXX ZANKA  CALL POMIK
3D            DEC A
20FA          JR NZ, ZANKA
62            LD H, D
1E1E          LD E,#1E
0B            DEC BC
3E08          LD A,#08
```

```
3600   BRISI LD (HL),#00
CDXXXX       CALL POMIK
3D           DEC A
20F8         JR NZ, BRISI
2658         LD H,#58
54           LD D, H
3A8D5C       LD A, (ATTR_P)
77           LD (HL), A
EDB8         LDDR
C9           RET
```

Priporočam, da se pri obeh programih zadržite in si vzamete toliko časa, da vam bosta razumljiva. Naslova BORDCR in ATTR_P, ki se pojavita, sta naslova sistemskih spremenljivk. V njih so shranjene vrednosti prilastkov, in sicer za zgornjo vrstico v ATTR_P, za spodnjo pa v BORDCR.

Priložen je še program v basicu, ki pokaže oba programa na delu. Ne gre za kakšno nesluteno odlično igro ali umetniško stvaritev, gre le za prikaz. Seveda pa ju lahko uporabite v kateremkoli drugem programu - je kar nekaj iger, ki bi se močno izboljšale s takšnim dodatkom. Program, bo na zaslonu narisal vzorec naključno izbranih znakov. S pritiskom na puščici "gor" (tipka 7) oz. "dol" (tipka 6) boste vzorec premikali.

```
 10 DIM A$ (22,36)
 20 FOR I = 1 TO 22 : LET B$ = CHR$ (INT (96*RND)+32)
 30 FOR J = 0 TO 5 : LET B$ = B$ + B$ : NEXT J
 40 LET A$(I) = CHR$ 16 + CHR$ INT (8*RND) + CHR$ 17 + CHR$
    INT (8*RND) + B$
 50 PRINT A$(I) : NEXT I
 60 LET A = 1
 70 PAUSE 0 : LET B$ = INKEY$
 80 LET B = A + 1: IF B = 23 THEN LET B = 1
 90 LET C = A - 1: IF C = 0 THEN LET C = 22
100 IF B$="6" THEN PRINT AT 0, USR "dol"; A$(C): LET A = C
110 IF B$="7" THEN PRINT AT 21, USR "gor"; A$(A): LET A = B
120 GO TO 70
```


UPORABA TIPKOVNICE

V poglavju o v/i ukazih smo omenili, da lahko za čitanje tipkovnice poleg ukazov IN ter OUT uporabimo tudi podprograme v ROMu. Za branje s tipkovnice imamo na voljo tri, ki si jih bomo zdaj natančneje ogledali.

Prvi nosi ime KEY SCAN in ga najdete na naslovu 028E. Uporabite ga z ukazom CALL KEY SCAN (CD8E02 šestnajstiško). Dal vam bo precej uporaben rezultat, čeprav še vedno ne takšnega, kakršnega si najbrž želite. Poglejmo kaj napravi in kako.

KEY SCAN vrne rezultat v registrih D in E. Da boste razumeli, kaj pomenita ti dve števili, najprej poglejmo skico tipkovnice na Spectrumu:

```
[ 1 ][ 2 ][ 3 ][ 4 ][ 5 ][ 6 ][ 7 ][ 8 ][ 9 ][ 0 ]
  24   1C   14   0C   04   03   0B   13   1B   23

[ Q ][ W ][ E ][ R ][ T ][ Y ][ U ][ I ][ O ][ P ]
  25   1D   15   0D   05   02   0A   12   1A   22

[ A ][ S ][ D ][ F ][ G ][ H ][ J ][ K ][ L ][ENTER]
  26   1E   16   0E   06   01   09   11   19   21

[CAPS][ Z ][ X ][ C ][ V ][ B ][ N ][ M ][SYMB][SPACE]
  27   1F   17   0F   07   00   08   10   18   20
```

Kot vidite, ima vsaka tipka svojo številko. Imenujemo jo kod tipke. Kod tipke A npr. je 26, kod tipke U je 0A. Poigrajte se in povežite številke med seboj v zaporedju. Videli boste, da so kodi razvrščeni v nekakšni spirali, ki se začne pri tipki "B" in poteka nasprotno od smeri urinega kazalca. Na ta način si boste zlahka zapomnili kode in vam ne bo vedno treba gledati v tabele.

Vrednost, ki jo KEY SCAN naloži v DE, je tesno povezana s temi kodi. Če ni pritisnjena sploh nobena tipka, bo vsebina DE enaka FFFF. Če je pritisnjena ena - in samo ena ! - tipka, bo vrednost v registru D še vedno FF, v E pa bo kod pritisnjene

tipke. Pomembno je, da se v tem primeru "caps shift" in "symbol shift" NE razlikujeta od ostalih tipk. Ko boste pritisnili samo "symbol shift", bo rezultat FF18 - ravno tako kot bi bil rezultat FF1E, če bi pritisnili samo "S".

Zdaj pa vzemimo, da sta naenkrat pritisnjeni "caps shift" in še neka druga tipka. V tem primeru vrednost v D ne bo FF temveč 27 - kar je kod tipke "caps shift". V E bo kod druge pritisnjene tipke. Podobno velja, če je istočasno pritisnjena "symbol shift" in katerakoli druga tipka - razen "caps shift". V tem primeru bo D vseboval 18 (kar je kod tipke "symbol shift"), medtem ko bo v E kod druge tipke. Če pa istočasno pritisnete "caps shift" in "symbol shift", boste v DE dobili 2718. Zapomniti si velja, da je "caps shift" pred "symbol shift"; če namreč pritisnete obe naenkrat, dobite vrednost 2718 in ne 1827.

KEY SCAN vrne še en pomemben podatek - ničelno zastavico. V vseh omenjenih primerih bo ničelna zastavica ob povratku iz podprograma dvignjena. Če pa se zgodi, da je naenkrat pritisnjenih preveč tipk, bo ničelna zastavica ob povratku spuščena. Spuščena ničelna zastavica torej sporoča, da je vrednost v DE brez pomena. Zato je pred uporabo vrednosti iz registrskega para DE vedno dobro pogledati ničelno zastavico.

Podprogram KEY SCAN ima eno pomankljivost - spremeni vrednosti registrov A,B,C,D,E,H in L ! Če želimo ohraniti vrednosti parov BC in HL, si pomagamo s skladom:

```
C5          PUSH BC
E5          PUSH HL
CD8E02      CALL KEY_SCAN
E1          POP HL
C1          POP BC
```

Zdaj pa naprej. Vzemimo, da ni bilo pritisnjenih preveč tipk. V naslednjem koraku moramo ugotoviti, ali je bila uporabljena kakšna "prava" tipka. Zavreči moramo torej rezultate FFFF (nobene uporabljene tipke), FF27 (pritisnjena samo "caps shift") in FF18 (pritisnjena le "symbol shift"). V ta namen uporabimo podprogram KEY TEST, na naslovu 031E. Če je v DE "smiselna" vrednost, bo KEY TEST najprej napravil dvoje:

- vrednost registra D bo prenesel v register B,
- v register D bo naložil 0.

Nadaljeval bo v eno od dveh smeri:

- če je v DE FFFF, FF27 ali FF18, bo v register A naložil FF, 27 ali 18 in spustil zastavico prenosa;
- če je v DE kakršnakoli druga vrednost, bo v register A naložil kod osnovnega znaka tipke in dvignil zastavico prenosa.

Osnovni znaki tipk so označeni na skici tipkovnice. Njihove kode (t.im. ASCII kode) dobite v basicu z ukazom CODE. Primer: osnovni znak prve leve zgornje tipke je "1", kod tega znaka pa je 31H. Podprogram dobi te vrednosti v posebni tabeli (KEY TABLE na naslovu 0205).

Oglejmo si nazadnje še tretji podprogram, ki se imenuje KEY CODE (naslov 0333). Ta ugotovi, kateri znak (ne tipka !) je bil uporabljen in naloži njegov ASCII kod v register A. V ta namen morajo biti izpolnjeni določeni pogoji:

1) Register E mora vsebovati kod osnovnega znaka.
2) Register B mora vsebovati
   - FF , če ni pritisnjena nobena tipka "shift",
   - 27 , če je pritisnjena "caps shift" ter
   - 18 , če je pritisnjena "symbol shift".
3) Za G-mode (grafični znaki) mora register C vsebovati 02;
   za E-mode (znaki nad tipkami) mora C vsebovati 01;
   za F-mode (znaki pod tipkami) mora C vsebovati 00 in D 00;
   za L-mode (običajni način), mora C vsebovati 00, D 08, bit 3 sistemske spremenljivke FLAGS 2 pa mora biti 0;
   za C-mode (tiskane črke) mora C vsebovati 00, D 08, bit 3 sistemske spremenljivke FLAGS 2 pa mora biti 1.

Trenutno uporabljeni znak ugotovite tako, da v register C naložite vrednost sistemske spremenljivke MODE, v D pa sistemske spremenljivke FLAGS ter uporabite KEY CODE.

S pravkar pridobljenim znanjem lahko napišemo kratek program, ki nam bo prebiral tipkovnico s pomočjo podprogramov v ROM-u:

```
CD8E02 BERI   CALL KEY SCAN
200F          JR NZ, NIC
CD1E03        CALL KEY TEST
300A          JR NC, NIC
5F            LD E, A
```

```
0E00          LD C, 00
1608          LD D, 08
CD3303        CALL KEY CODE
A7            AND A
C9            RET
37      NIC   SCF
C9            RET
```

Ta podprogram lahko naložite kamorkoli v pomnilnik in z njim nato po mili volji prebirate tipkovnico. Kod uporabljenega znaka bo v registru A. Če je podatek neuporaben (o tem smo govorili zgoraj), bo zastavica prenosa dvignjena (izhod NIC in ukaz SCF = Set Carry Flag - dvigni zastavico prenosa).

Predlagam, da ne ostanemo le pri tem programčku. Z malo več truda namreč lahko izdelamo že kar spodoben program, ki bo pokazal, kolikšno hitrost nam ponuja strojni jezik. Imenujmo program TISK; z njim bomo "tiskali" s povečanimi črkami na zaslon. Naložite ga v pomnilnik takoj za BERI. Takšen je:

```
AF      TISK  XOR A             ;A = 0
323C5C        LD (TVFLAG), A    ;izpis naj bo na gornji del zaslona
ED62          SBC HL,HL         ;HL = 0000 (koordinate PRINT AT)
E5      START PUSH HL           ;porini koordinate na sklad
CDXXXX PAVZA  CALL BERI         ;beri tipkovnico
30FB          JR NC, PAVZA      ;počakaj, da bo tipka spuščena
CDXXXX CAKAJ  CALL BERI
38FB          JR C, CAKAJ       ;počakaj novo tipko
FE20          CP #20
3848          JR C, STOP        ;končaj, če je pritisnjen kontrolni
FE80          CP #80            ;znak (ki ima kod manjši od 20H)
3045          JR NC, STOP       ;končaj, je pritisnjen ukaz (= znak
6F            LD L, A           ;s kodom, večjim od 80H)
2600          LD H, 0           ;HL = kod znaka
29            ADD HL,HL
29            ADD HL,HL
29            ADD HL,HL
11003C        LD DE, #3C00
19            ADD HL, DE        ;HL kaže na točkovni vzorec znaka
0E04          LD C, 4
D1      ZAN1  POP DE            ;DE = koordinate PRINT AT položaja
3E16          LD A,#16          ;A = kontrolni znak "AT"
```



```
D7            RST 16            ;PRINT AT ...
7A            LD A, D
D7            RST 16            ;D,...
7B            LD A, E
D7            RST 16            ;E.
14            INC D             ;naj kazalec kaže v naslednjo vrsto
D1            PUSH DE           ;daj PRINT AT koordinate na sklad
0604          LD B, 4
56            LD D, (HL)        ;prenesi dve vrstici točkovnega
23            INC HL            ;vzorca v DE
5E            LD E, (HL)
23            INC HL
3E08    ZAN2  LD A, 8           ;po enem pomiku bo to postalo 80H
CB13          RL E
17            RLA
CB13          RL E
17            RLA
CB12          RL D              ;izračunaj, kateri znak je treba
17            RLA               ;izpisati
CB12          RL D
17            RLA
D7            RST 16            ;izpiši ga
10EF          DJNZ ZAN2
0D            DEC C
20DC          JR NZ, ZAN1
E1            POP HL            ;HL = koordinate PRINT AT položaja
7D            LD A, L           ;A = številka levega stolpca prav-
FE1C          CP #1C            ;kar izpisanega znaka
2008          JR NZ, NASLD      ;skoči in pripravi koordinate za
                                ;naslednji znak, če ni pravkar
                                ;izpisani znak zadnji v vrsti
2E00          LD L,0            ;postavi številko stolpca na 0.
                                ;številka vrstice je že prava
7C            LD A, H           ;A = številka vrstice
FE14          CP #14
C8            RET Z             ;v basic, če je že izpisanih pet
18AF          JR START          ;vrst, sicer odčitaj naslednji znak
1104FC NASLD  LD DE, #FC04
19            ADD HL,DE         ;povečaj koordinate PRINT AT
18A9          JR START          ;in odčitaj naslednji znak
F1      STOP  POP AF            ;uravnaj sklad
C9            RET               ;in se vrni v basic
```

Števila z oznako # so šestnajstiška, ostala pa desetiška. Program poženite z RANDOMIZE USR "TISK" ("TISK" pomeni naslov; vstaviti morate ustrezno vrednost). Zaključil se bo, ko bo izpisanih pet vrst. Ker so ob programu opombe, naj vas posebej opozorim le na zanimiv način izpisa, povsem podobnega ukazu PRINT AT D,E v basicu. Dosežemo ga z zaporedjem LD A,16H / RST 10H / LD A,D / RST 10H / LD A,E / RST 10H. Na drug način - v našem programu ga nismo uporabljali - določimo mesta izpisa s sistemskima spremenljivkama DF_CC in S_POSN. V DF_CC je treba naložiti naslov PRINT položaja v zaslonski datoteki, v S_POSN pa koordinate položaja. Ukaz PRINT AT 0,0 bi v tem primeru zapisali LD HL,0000/ LD (DF_CC),HL/ LD HL,1821H/ LD (S_POSN),HL/ RST 10H.

Po tem, kar ste izvedeli, naj vas spomnim še na IN ukaze. Čeprav zahteva njihova raba precejšno spretnost v strojnem programiranju, so včasih nepogrešljivi. Eden takšnih primerov je več hkrati pritisnjenih tipk: če uporabljamo podprograme v ROM-u, bomo dobili neuporaben rezultat. Program, ki ga boste v tak namen napisali sami, vas bo rešil težav. Prav tako lahko z lastnim programom spremenite "pomen" tipk - po potrebi bo "symbol shift" pomenila "D", "k" bo "1" ... in podobno. Vendar so takšne stvari le redko potrebne. Zato vam priporočam, da se dobro seznanite s programi v ROM-u, ker bodo najbrž edini, ki jih boste uporabljali za branje s tipkovnice.

Za konec poglavja pa še kanček kulture. Se strinjate? Če se ne, lahko preostanek preskočite - še žal vam bo! Ogledali si bomo program, s katerim bomo tipkovnico vašega Spectruma spremenili v klaviaturo. Uporabili bomo podprogram KEY SCAN, s katerim ste se seznanili v začetku poglavja, poleg tega pa se bomo dotaknili še ene uporabne stvari: "izdelave" zvoka na Spectrumu.

Že v poglavju o vhodno-izhodnih ukazih smo rekli, da lahko z ukazi OUT vplivamo na zvočnik skozi izhodna vrata FEH (254). Na zvočnik učinkuje vrednost bita 4 v izhodnem registru. Če bi torej uporabili ukaz OUT (FEH), A, bi na zvočnik vplival bit 4 registra A. Če je vrednost 4. bita 0, je zvočnik izključen, če je 4. bit 1, je vključen. Da pa bi dobili zvok, moramo vrednost menjati. Hitrost, s katero menjamo vrednost, določa frekvenco (višino) tona. To zahteva nekaj računanja, v katerega pa se ne bomo podali. Oglejte si le končni rezultat - podprogramček, ki v našem programu nosi naslov TON. Frekvenca tona je pri tem

odvisna od vrednosti v registru B. Za zvok s frekvenco 440 nihajev na sekundo (glasbeniki ga imenujejo komorni A), mora imeti B vrednost 138. Ostale vrednosti lahko razberete iz tabele na začetku programa. Na tipkovnici je razpored takšen:

```
[   ][   ][ C# ][ D# ][    ][ F# ][ G# ][ A# ][    ][    ]

 [    ][ C  ][ D  ][ E  ][ F  ][ G  ][ A  ][ H  ][ C  ][    ]

   [    ][ C# ][ D# ][    ][ F# ][ G# ][ A# ][    ][    ][    ]

[    ][ C  ][ D  ][ E  ][ F  ][ G  ][ A  ][ H  ][ C  ][    ]
```

Kot vidite, sta vam na voljo dve oktavi, kar je zadosten obseg za kar precejšne število melodij. Program lahko uglasite po lastni želji tako, da spremenite vrednosti v tabeli NOTE. Ničla pomeni, da tipka ne bo dala zvoka. Sicer pa velja: čim višje število, tem nižji bo ton in obratno. In še nasvet: če le imate možnost, priključite vaš računalnik (preko EAR ali MIC vtičnice) na radio ali ojačevalec. Spectrumov zvok ni slab; če boste uporabili naš glasbeni program in ob tem upoštevali moj nasvet, vam ne bo žal.

Preden gremo zares, še nekaj malenkosti. Program lahko naložite kamorkoli v pomnilnik (le vstaviti morate pravilne absolutne naslove NOTE in TON. Poženete ga z naslova START (RANDOMIZE USR "START"), ustavite z BREAK (caps shift + space). Za lažje razumevanje moram pojasniti, da skozi vrata FEH vplivamo tudi na rob zaslona. Zato je v programu ukaz LD A, (5C48), ki s pomočjo sistemske spremenljivke BORDCR (na naslovu 5C48) poskrbi, da se rob zaslona ne spreminja.

Zdaj pa končno:

```
9E93464B NOTE  DEFB 9E93464B  ;toni  G    G#    G+    F#+
                              ;tipke B    H     Y     6
0050A9B4       DEFB 0050A9B4  ;toni  -    F+    F#    F
                              ;tipke 5    T     G     V
8A813D42       DEFB 8A813D42  ;toni  A    A#    A+    G#+
                              ;tipke N    J     U     7
5C5600C1       DEFB 5C5600C1  ;toni  D#+  E+    -     E
                              ;tipke 4    R     F     C
```

```
78003539        DEFB 78003539  ;toni  B    -     B+    A#+
                               ;tipke M    K     I     8
6962CFDD        DEFB 6962CFDD  ;toni  C#+  D+    D#    D
                               ;tipke 3    E     D     X
70003200        DEFB 70003200  ;toni  C+   -     C++   -
                               ;tipke sym  L     O     9
0070ECFD        DEFB 0070ECFD  ;toni  -    C+    C#    C
                               ;tipke 2    W     S     Z
00000000        DEFB 00000000  ;toni  -    -     -     -
                               ;tipke spc  ent   P     0
00000000        DEFB 00000000  ;toni  -    -     -     -
                               ;tipke 1    Q     A     caps
00        TON   NOP            ;ta podprogram povzroči premor
00              NOP            ;točno določene dolžine
00              NOP
10FB            DJNZ TON
D3FE            OUT (#FE), A   ;oddaj impulz (en element žele-
C9              RET            ;nega tona)
3A485C    START LD A,(#5C48)   ;daj v A podatek o robu zaslona
CB1F            RRA
CB1F            RRA
CB1F            RRA
E607            AND 7          ;A = barva roba
F610            OR #10         ;da bitu 4 vrednost 1
4F              LD C, A
F3              DI             ;potek naj bo brez prekinitev
C5        ZANKA PUSH BC
CD8E02          CALL #028E     ;= CALL KEY SCAN. V DE bo podatek
C5              POP BC         ;o tipkovnici
212027          LD HL, #2720   ;HL = kod "caps shift + space"
A7              AND A
ED52            SBC HL,DE
281A            JR Z, KONEC    ;končaj, če je pritisnjen BREAK
7B              LD A, E        ;podatek o pritisnjeni tipki brez
3C              INC A          ;shiftov
28EF            JR Z, ZANKA    ;nazaj, če ni pritisnjena nobena
AF              XOR A          ;tipka
57              LD D, A        ;DE = podatek o pritisnjeni tipki
                               ;(brez shiftov)
21XXXX          LD HL, NOTE
19              ADD HL, DE     ;HL = naslov note v tabeli
```



```
46              LD B,(HL)      ;B = podatek za ustrezno noto
B8              CP B           ;primerjaj B z 0
28E5            JR Z, ZANKA    ;nazaj, če tipka ne predstavlja
                               ;nobene note
79              LD A, C        ;A = barva roba ekrana, bit 4 = 1
C5              PUSH BC
CDXXXX          CALL TON       ;prva polovica ciklusa note
C1              POP BC
E607            AND 7          ;bit 4 = 0
CDXXXX          CALL TON       ;druga polovica ciklusa note
1B              JR ZANKA
FB        KONEC EI
C9              RET
```

ARITMETIKA S PLAVAJOČO VEJICO

Zadnje poglavje je namenjeno predvsem tistim, ki nameravate uporabljati strojne programe za reševanje matematičnih in tehničnih nalog. Vseeno najtopleje priporočam, da ga vsaj preletite, tudi če vaši nameni niso takšni.

Gotovo ste opazili, da smo se v knjigi ves čas ukvarjali le s celimi števili. Seveda je računalnik kos tudi realnim - navadno pravimo "decimalnim" - številom. V računalništvu imenujemo takšno aritmetiko aritmetiko s plavajočo vejico. Ukazi, ki smo jih obravnavali v prejšnih poglavjih, so ukazi procesorja Z80. Zato so enaki za vse računalnike, ki imajo ta CP. Pri delu s plavajočo vejico pa uporabljamo izključno podprograme v ROM-u. Stvari, s katerimi se boste seznanili zdaj, veljajo zato (razen ukaza RST 28H) le za Sinclairjev Spectrum.

Oglejmo si najprej, kako so realna števila zapisana v pomnilniku. Vsako število (razen 0) lahko zapišemo v obliki

$$x = m * 2\uparrow n$$

m imenujemo mantisa, n pa eksponent. V Spectrumu zavzema število v plavajoči vejici 5 zlogov. Prvi zlog je namenjenen eksponentu, natančneje: vsoti 128+n. V ostalih štirih zlogih je zložena mantisa v dvojiški obliki. Mantisa mora imeti vrednost med 1/2 in 1 (lahko je 1/2, ne more pa biti 1). Ker je manjša od 1, ima dvojiško vejico:

1/2 napišemo dvojiško 0.1
1/4 " " 0.01
3/4 " " 0.11 itd.

Ker mora biti mantisa vedno večja ali vsaj enaka 1/2, je prvi bit drugega zloga vedno 1. Ker se tega zavedamo, uporabljamo ta bit raje za označevanje predznaka - 0 pomeni pozitivno, 1 negativno število. Poskusimo zdaj po teh pravilih zapisati 0.3125.

0.3125 = 5/16
5/16 = m * 2↑n
5/16 = 5/8 * 2↑(-1)
5/8 = 1/2 + 1/8

Prvi zlog = 128 + eksponent = 128 + (-1) = 127 = 7FH
Drugi zlog = 0 0 1 0 0 0 0 0 = 20H. Zlogi 3, 4 in 5 so 0. Prvi bit drugega zloga je enak 0, ker je število pozitivno. 0.3125 je v plavajoči vejici enako 7F 20 00 00 00.

Števila v petih zlogih ne moremo, kot smo to počeli do zdaj, spraviti v en register ali registrski par. Zato realna števila spravljamo v registrski "peterček" AEDCB. S tem porabimo vse registre. Ker bomo gotovo potrebovali več kot eno realno število, moramo imeti na voljo še kakšno mesto za shranjevanje. V RAM-u je v ta namen rezerviran prostor - računski sklad.

Računski sklad je zelo podoben strojnemu skladu, ki ga že dobro poznate. V nekaterih pomembnih podrobnostih pa se oba sklada razlikujeta:
- v pomnilniku se nahajata na različnih mestih;
- računski sklad "raste" navzgor, ne navzdol;
- vsaka vrednost na računskem skladu zavzema 5 zlogov in ne le 2 kot na strojnem;
- računski sklad lahko shranjuje poleg števil tudi znake ter
- ukazi CALL in RET nanj ne vplivajo.

Ukaz RST 28H v Spectrumu označuje, da želimo izvesti operacije - eno ali več - s števili na računskem skladu. Števila, ki slede ukazu (kodi operacij), določajo za kakšne operacije gre. Niz teh kodov se mora vedno končati z 38H, ki označuje, da spet sledi strojni ukaz. Med izvajanjem ukaza RST 28 sname CP operande z računskega sklada, izvede zahtevano operacijo ter rezultat porine nazaj na sklad. Primer: zaporedje RST 28 04 38 bo snelo s sklada zgornji dve vrednosti (sklad bo tako za dve števili krajši), ju pomnožilo (kod operacije 04) ter dobljeno vrednost porinilo na sklad. V nizu je seveda lahko tudi več kodov. Kodi operacij in njihovi pomeni so zbrani v tabeli na naslednji strani. Omejili smo se le na pogosteje uporabljane in laže razumljive, ker bi bila za ostale potrebna obširnejša razlaga. A videli boste, da vam ukaz RST 28 omogoča več kot le preprosto aritmetiko.

UPORABA PODPROGRAMOV IN RAČUNSKEGA POMNILNIKA

Preden začnemo računati, moramo nekako poriniti ustrezna števila na računski sklad. Najpreprostejši način za to je uporaba treh podprogramov v ROM-u:

pomnilnika. Prav tako izbriše vrednosti v prostorih 0 in 1 ukaz RST 10, kadar izpisuje grafične znake. Zaradi tega je najbolj zanesljivo uporabljati le prostore 3, 4 in 5. Poglejmo zdaj primer: želeli bi izračunati izraz SIN X + COS X. Recimo, da je število X že na vrhu računskega sklada. Uporabimo zaporedje

```
EF      RST 28
C5      DEFB C5 (spravi_5)   ;spravi X v prostor 5. X ob tem
                               ostaja tudi na vrhu sklada.
1F      DEFB 1F (sin)        ;izračunaj SIN X. Rezultat nado-
                               mesti vrednost na vrhu sklada
                               (tj. odstrani X).
E5      DEFB E5 (vzemi_5)    ;porini vrednost iz prostora 5 na
                               sklad. Na vrhu sklada je zdaj
                               X, pod njim SIN X.
20      DEFB 20 (cos)        ;izračunaj COS X. Rezultat naj
                               nadomesti vrednost na vrhu skla-
                               da (tj. odstrani X).
0F      DEFB 0F (seštej)     ;odstrani zgornji dve vrednosti
                               (SIN X in COS X) s sklada in ju
                               nadomesti z njuno vsoto.
38      DEFB 38 (končaj)
```

Vsoto dveh kotnih funkcij smo tako izračunali v samo sedmih zlogih! Podobno je tudi z rabo drugih operacij. Opozoriti pa velja na operacije VAL, VAL$ in STR$. Njihova raba je bolj zapletena, zato naj bi se jih lotil le tisti, ki dobro pozna Spectrumov nadzorni program.

Toliko za okus. Raziskovanje številnih preostalih možnosti prepuščam vam, da boste imeli od učenja res pravo korist.

KAR SE JANEZEK NAUČI - - -

V zadnjem poglavju bomo pridobljeno znanje uporabili za izdelavo programa v strojnem jeziku. In česa bi se v tem programu lotili? Gotovo je najprivlačnejša misel o igrici. Pri oblikovanju takšnega programa smo povsem svobodni, edina omejitev je naša iznajdljivost. Odločimo se torej za igro, ki bo zahtevala nekaj programerske veščine, računalnikove hitrosti in - na koncu - igralčeve spretnosti. Program bomo vnašali z urejevalnikom, zato bomo skrbeli, da ne bo predolg.

Predlagam igro, ki bo miroljubnejše narave. Odpovejmo se streljanju in pobijanju nasprotnikov, ki brzijo čez zaslon. Raje naj igralec npr. zbira določene predmete, pri tem pa naj mu bo na voljo le omejen čas. Več predmetov ko bo zbral, večje bo število točk; hitreje ko jih bo zbral, toliko boljši bo uspeh. Da pa ne bo vse odvisno le od hitrosti, pošljimo v igro še nasprotnika. Ta naj preganja našega junaka. Če ga bo ujel, bo to pomenilo takojšen neuspešen zaključek.

Obrise igre zdaj že imamo. Lotimo se najprej izdelave načrta, nato pa bomo postopoma zapisali program. Za začetek moramo pripraviti vse, kar bomo v igri potrebovali:

```
Začetek   Razpostavi predmete.
          Pripravi uro.
          Nariši junaka.
          Nariši nasprotnika.
```

Še kaj? No da: vsakič bo treba preveriti, ali ima junak še kakšno življenje, treba bo šteti, ali so razpostavljeni že vsi predmeti... in še kaj. A to so že podrobnosti - in posebej smo poudarili, da podrobnosti vedno prihranimo za na konec. Torej lahko rečemo, da je načrt začetka igre narejen. Zdaj je na vrsti glavni, bolj zapleteni del - igra sama. Ta naj ima eno glavno zanko, iz katere bomo po potrebi klicali stranske zanke, tj. podprograme. V glavni zanki bo tekla ura, spremljali bomo premike igralca, premikali nasprotnika in kontrolirali, ali so že pobrani vsi predmeti. Takole:

```
Ura       Zmanjšaj preostali čas.
          Je čas že potekel? Če je, končaj igro.
```

```
Test      Je pritisnjena katera od tipk za premikanje
          igralca? Če je, pokliči ustrezni podprogram
          za premik.
          Premakni nasprotnika.
          So pobrani vsi predmeti? Če so, kliči podprogram
          za obračun točk.
          Vrni se na Uro.
```

Za premik bomo morali pripraviti štiri podprograme (gor, dol, desno, levo), ki pa bodo imeli enako zgradbo:

```
Premik    Premakni figuro v ustrezno smer.
          Je zadela (= je pobrala) katerega
          od predmetov? Če je, kliči ustrezni podprogram.
          Je zadela nasprotnika? Če je, kliči ustrezni
          podprogram.
          Vrni se v glavno zanko.
```

Nasprotnika bomo premikali v eni od stranskih zank. Premikati se mora tako, da bo preganjal našega junaka - se pravi, da mu bo sledil:

```
Pregon    Izračunaj, v katero smer je treba slediti.
          Premakni nasprotnika.
          Je zadel igralca? Če je, kliči ustrezni
          podprogram.
          Je zadel katerega od predmetov? Če je,
          kliči ustrezni podprogram.
          Vrni se v glavno zanko.
```

Potrebujemo še podprograme, ki smo jih že omenjali:
- igralec je pobral (zadel) predmet (Zadet),
- igralec je zadel nasprotnika ali obratno (Konec),
- nasprotnik je pobral predmet (Minus) ter
- pobrani so vsi predmeti (Obračun).

```
Zadet     Zmanjšaj število preostalih predmetov.
          Povečaj število točk.
          Vrni se.
Konec     Napravi eksplozijo.
          Zaključi igro.
          Vrni se.
```

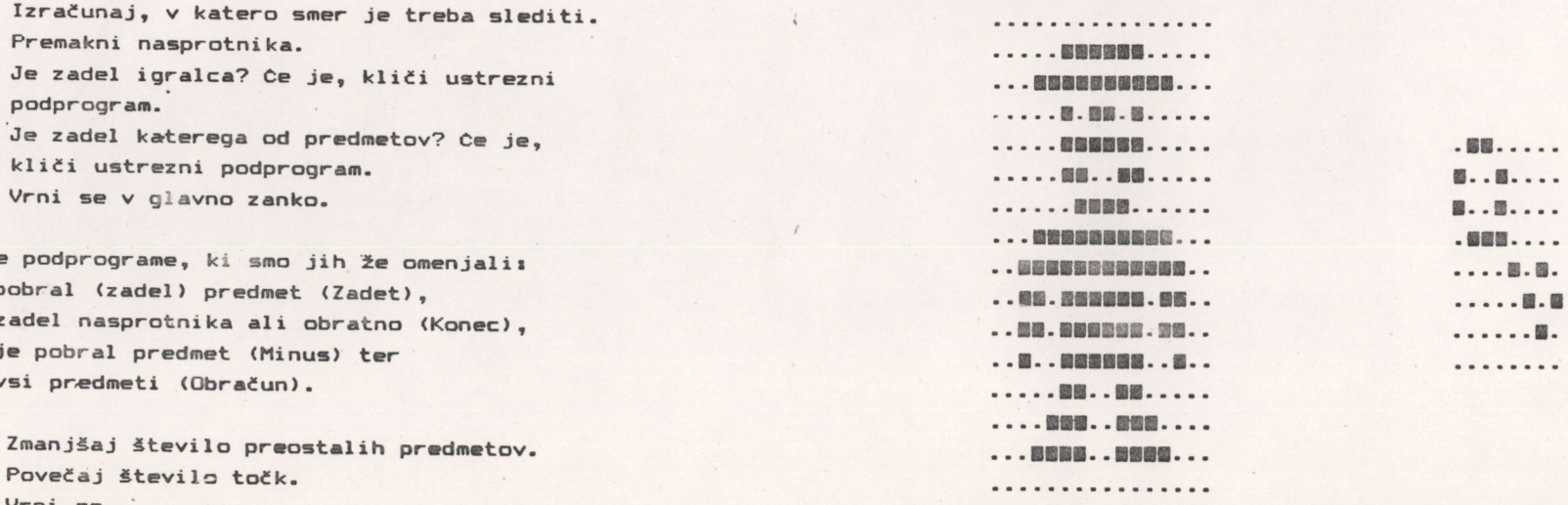


```
Minus     Zmanjšaj število preostalih predmetov.
          Vrni se.
Obračun   Obračunaj dosežene točke.
          Zaključi igro.
          Vrni se.
```

Igra se bo zaključila, ko bo igralec porabil vsa življenja.

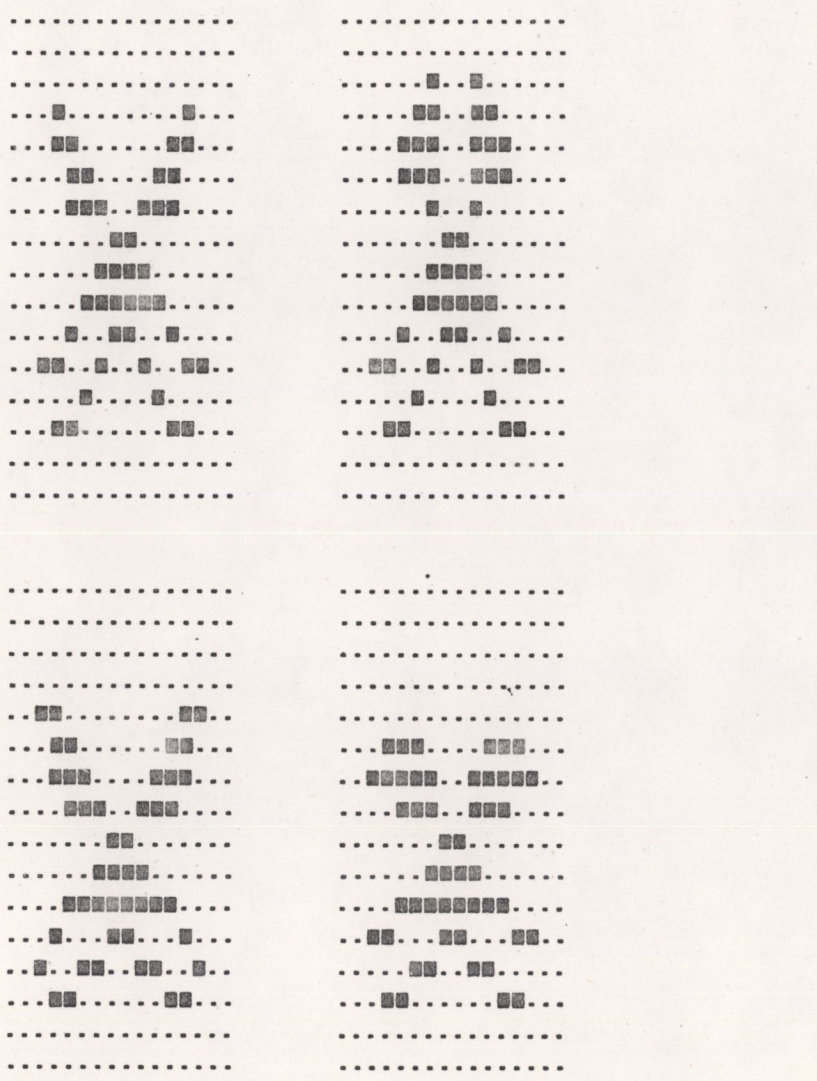
```
Zaključek Obriši zaslon.
          Izpiši dosežene točke.
          Vprašaj, ali želi igralec še eno igro.
          Če jo želi, pojdi na Začetek, sicer
          končaj.
```

Tako. Napravili smo precejšen korak - načrt je pripravljen. Oddahnimo si nekoliko in za odmor izdelajmo podobe naših likov. Kakšen naj bo junak, kakšni naj bodo predmeti, ki jih pobira, in kakšen naj bo nasprotnik? Vsak od nas bi na ta vprašanja seveda našel svojo rešitev. Predlagam naj bo glavni junak res junak, prava podoba Martina Krpana, njegov cilj naj bo, pobrati vse ključe na zaslonu:



Za nasprotnika pa naj bo velika žuželka, metulj - ljudožerec. Igro bomo popestrili s kančkom računalniške animacije: metulj bo letal po zaslonu in zamahoval s krili. Zaradi tega moramo izdelati več sličic, ki bodo v povezanem zaporedju dale vtis

metulja, ki leta. Dovolj bodo štiri, ki naj izgledajo takole:



Zdaj že imamo junaka, metulja in ključe; manjka le še ura. Naj za uro služi stolpec na levi strani zaslona, ki se bo nižal, ko bo čas tekel.

Pretehtajmo zdaj, kateri deli programa bodo zapisani v strojnem jeziku in kateri v basicu. Ključi morajo biti razpostavljeni naključno. Žal funkcija RND, ki jo v ta namen uporabljamo v basicu, v ROM-u ni zapisana kot podprogram temveč kot del večje zanke. Zato bomo ključe razpostavili iz basica. Prav tako bomo iz basica narisali stolpec ure. Vmesne dele, pri katerih je bistvena hitrost, bomo napisali v strojnem jeziku. Na koncu pa se bomo spet vrnili v basic in z njim zapisali zaključek.

Preden se lotimo programa, še dve stvari, s katerima bi se sicer srečali kasneje, a je zaradi lažje razlage bolje da ju razjasnimo zdaj. Prva je vprašanje, kako bomo uravnavali tek ure. Najenostavneje bo, če uporabimo vrednost, shranjeno v treh zlogih med sistemskimi spremenljivkami. Ta vrednost se ob vsaki prekinitvi (se pravi vsako 1/50 sekunde) poveča za eno. Mi bomo uporabili najnižji zlog na naslovu 23672 (ki se najhitreje spreminja) in po njem uravnali uro. Drugo vprašanje zadeva označevanje položaja figur in njihovo premikanje. Razporeditev naslovov v zaslonski datoteki ni povsem preprosta - in ker sta Krpan in metulj velika 16 * 16 bitov, je videti to vprašanje še za stopnjo bolj zapleteno. Lepo bi bilo, če bi mogli uporabljati običajne koordinate - denimo takšne kot z ukazom PLOT v basicu. Zaslon (natančneje: zgornjih 22 vrst) je razdeljen na 176 vrstic in 256 stolpcev. Koordinati v ukazu PLOT imata zato lahko vrednost od 0 do 175 (višina = koordinata y) oz. od 0 do 255 (širina = koordinata x). No, sreča je na naši strani: tudi v strojnih programih lahko uprabljamo te koordinate s pomočjo podprograma PIXEL_ADD, ki ga najdemo v ROM-u na naslovu 22AAH (= 8874). V register B naložimo koordinato y (oddaljenost od spodnjega roba zaslona) neke točke (bita), v register C pa pa koordinato x (oddaljenost od levega roba). Po klicu podprograma PIXEL_ADD dobimo v paru HL naslov zloga, v katerem je ta točka, v registru A pa njen položaj v zlogu (od 0 do 7). Podprogram spremeni vrednosti v parih BC in DE. Uporaba podprograma PIXEL_ADD je zelo prijetna možnost, zato se kar odločimo zanj. Zaradi tega bo najbolj priročno, če bomo imeli koordinate vseskozi spravljene v parih BC: v paru BC višino in odmik Krpanovega položaja, v paru BC izmenjalnega registrskega niza pa višino in odmik metulja. To pa nam nalaga posebno skrb za vrednosti v obeh parih BC: spraviti ju bomo morali, kadarkoli bo nevarnost, da se uničita. Zato se kar takoj odločimo za podprogramček z imenom NASLOV:

```
NASLOV    PUSH BC
          CALL PIXEL_ADD
          POP BC
          RET
```

NASLOV nam bo posredoval naslov določenega položaja, koordinate pa bodo pri tem ostale nedotaknjene.

Zdaj pa k delu. Lotimo se najprej glavne zanke:

```
URA       LD  A, (23672)     ;vrednost "n", ki jo bomo določili
          CP  n              ;kasneje, uravnava hitrost ure.
          JR  C, TEST        ;Če še ni presežena vrednost n,
          PUSH  BC           ;skoči v zanko TEST. Sicer shrani
          LD  BC, (VISURE)   ;koordinate Krpana.
```

Z imeni bomo označevali naslove rezerviranih celic. Zloga na naslovu VISURE bosta vsebovala koordinate stolpca ure. Ker bo stolpec na levi strani zaslona, bo njegova oddaljenost od levega roba (koordinata x) v registru C vedno 0, koordinata y, shranjena v registru B pa bo pravzaprav višina stolpca. Ta bo na začetku 175 in se bo zmanjševala. Ko bo dosegla 0, bo igre konec.

```
          CALL  NASLOV       ;Izračunaj naslov.
          LD  (HL), 129      ;Izbriši gornjo vrstico, vendar naj
                             ;stranici (bita 0 in 7) ostaneta.
          LD  (23672), A     ;Uri daj vrednost 0 (register A bo
                             ;imel po klicu NASLOVa vrednost 0,
                             ;ker se bodo koordinate vedno na-
                             ;našale na levi gornji bit likov).
          DEC  B             ;Znižaj višino stolpca ure, ter
          LD  (VISURE), BC   ;shrani njegove koordinate.
          POP  BC            ;S sklada vzemi podatke o položaju
                             ;junaka. Zadnji ukaz, ki je vplival
                             ;na zastavice, je bil DEC B:
          JR  NZ,'TEST       ;če višina stolpca še ni 0, pojdi v
          LD  BC, (TOCKE)    ;zanko TEST, sicer naloži v BC do-
          RET                ;sežene točke in se vrni v basic.
TEST      PUSH  BC           ;Shrani koordinate.
          CALL  KEY_SCAN     ;Preleti tipkovnico. KEY_SCAN vrne
          POP  BC            ;v registru E kod pritisnjene tipke
          LD  A, E           ;(glej poglavje o rabi tipkovnice).
```


Zdaj bomo preverjali, ali je bila pritisnjena katera od tipk za premik. Naj služita za premik levo in desno tipki O in P, za gor in dol pa Q in A. Seveda lahko izberete tudi druge tipke, le spremeniti boste morali vrednosti, ki slede.

```
          CP  #25            ;Če je pritisnjena tipka Q...
          CALL  Z, GOR       ;       ...kliči podprogram GOR.
          CP  #22            ;    -"-                P...
          CALL  Z, DESNO     ;       ...    -"-         DESNO.
          CP  #1A            ;    -"-                O...
          CALL  Z, LEVO      ;       ...    -"-         LEVO.
          CP  #26            ;    -"-                A...
          CALL  Z, DOL       ;       ...    -"-         DOL.
          CALL  Z, METULJ    ;Kliči podprogram za premik metulja
          LD  A, (KLJUC)     ;V zlogu KLJUČ shranimo število že
          CP  n              ;pobranih ključev. Če so pobrani že
          CALL  Z, OBRAC     ;vsi (na začetku je na zaslonu n
                             ;ključev) kliči podprogram za obra-
                             ;čun točk.
          JR  URA            ;Ponovi glavno zanko.
```

Če bo igralcu uspelo pobrati vse ključe, preden se bo čas iztekel, se bo vrednost doseženih točk za vse ključe pomnožila s preostankom časa. To bo opravil podprogram OBRAC.

Glavna zanka je tako pripravljena. Nadaljujmo kar po vrsti s podprogrami, ki jih bomo potrebovali. Najprej GOR:

```
GOR       LD  A, 175         ;Če je lik na zgornjem robu zaslo-
          CP  B              ;na, ne more več navzgor. Zato se v
          RET  Z             ;tem primeru vrni v glavno zanko.
```

Zdaj moramo preveriti, ali ni nad junakovo glavo metulj ali ključ. V ta namen je najprimernejši tretji (oz. četrti, če štejemo od roba klobuka) zlog nad junakovo glavo. Če je tam metulj, bo imel vrednost 1 četrti bit zloga, če bo tam ključ, bo "prižgan" bit 0. To lahko strnemo v kratek podprogram, ki ga bomo dodali na koncu. Večkrat nam bo prišel prav. Imenujmo ga BIT40:

```
BIT40     BIT  4, (HL)       ;Če je tu metulj, kliči podprogram,
          CALL  NZ, KONEC    ;ki bo zaključil igro;
          BIT  0, (HL)       ;če je tu ključ, kliči podprogram,
          CALL  NZ, ZADET    ;ki bo obračunal zadetek
          RET                ;in se vrni.
```

Teh pet vrstic bomo dodali na koncu programa, zdaj pa se vrnimo k podprogramu GOR.

```
        PUSH  BC        ;Shrani koordinate,
        INC   B         ;izračunaj
        INC   B         ;         naslov 3.
        INC   B         ;                  zloga
        CALL  NASLOV    ;                        nad junakom
        CALL  BIT40     ;in  preveri  ali  ni zadel metulja
                        ;ali ključa.
        INC   HL        ;Naslov sosednjega zloga (preveriti
        CALL  BIT40     ;moramo  oba  zloga,  ker  sta tako
        POP   BC        ;Krpan kot metulj široka 16 bitov).
```

Da bo gibanje naše figure gladko, se mora v vse smeri premikati bit za bitom, nikoli za več bitov naenkrat. Da pa bo program enostavnejši, bomo premik vsakič ponovili osemkrat. To pomeni, da se bo na vsak pritisk tipke figura pomaknila, bit za bitom, za osem bitov. V ta namen bomo naredili zanko. Če ne bi uporabljali para BC za shranjevanje koordinat, bi si lahko lepo pomagali z ukazom DJNZ. Tako pa bomo števce shranjevali v rezervirani pomnilniški celici, ki ju bomo označevali STEVC1 in STEVC2.

```
        LD    A, 8
GR      LD    (STEVC1), A ;Prvi števec.
        INC   B           ;To je zdaj koordinata 1. zloga nad
        PUSH  BC          ;figuro. Shrani vrednost.
        LD    A, 16
ZAN1    LD    (STEVC2), A ;Drugi števec.
        CALL  NASLOV      ;izračunaj naslov zloga...
        PUSH  HL          ;...in ga shrani.
        DEC   B
        CALL  NASLOV      ;Izračunaj  naslov  zloga pod njim,
        POP   DE          ;naslov  zgornjega  zloga  pa daj v
                          ;par DE.
        LD    A, (HL)     ;Prenesi vrednost spodnjega zloga..
        LD    (DE), A     ;...v zgornji zlog.
        INC   HL          ;To sta zdaj naslova...
        INC   DE          ;          ...sosednjih dveh zlogov.
        LD    A, (HL)     ;Prenesi tudi tu...
```



```
        LD    (DE), A     ;                ...vrednost navzgor.
        LD    A, (STEVC2) ;Se  je zanka že ponovila 16 krat ?
        DEC   A           ;(se pravi: se je že premaknila vsa
        JR    NZ, ZAN1    ;figura?).  Če  se ni, ponovi ZAN1,
        POP   BC
        LD    A, (STEVC1) ;sicer  zmanjšaj prvi števec in po-
        DEC   A           ;novi veliko zanko. Če se je velika
        JR    NZ, GR      ;zanka že ponovila osemkrat,
        RET               ;se vrni.
```

To je bil podprogram za pomik navzgor. Zdaj pa premik v levo:

```
LEVO    LD    A, 8        ;Če  je  figura  na  levem robu (ob
        CP    C           ;stolpcu ure), ne more na levo. Za-
        RET   Z           ;to  se  takoj vrni v glavno zanko.
        PUSH  BC
        LD    A, C        ;Najprej  bomo preverili, ali ni na
        SUB   8           ;levi strani lika metulj ali ključ.
        LD    C, A
        LD    A, B
        SUB   5           ;Za preverjanje bomo uporabili isti
        LD    B, A        ;zlog kot pri premiku navzgor. Nje-
        CALL  NASLOV      ;gove koordinate so zdaj v paru BC.
        CALL  BIT40
        LD    A, B
        SUB   8
        LD    B, A        ;Kontroliramo tudi zlog, ki je osem
        CALL  NASLOV      ;vrstic  niže (ker je figura visoka
        CALL  BIT40       ;16 vrstic).
        POP   BC
        LD    A, 8
LV      LD    (STEVC1), A ;Prvi števec.
        PUSH  BC
        LD    A, C        ;Ker  bo  pomik v levo, potrebujemo
        ADD   A, 15       ;koordinati desnega roba figure.
        LD    C, A        ;Ti sta zdaj v paru BC.
        LD    A, 16
ZAN4    LD    (STEVC2), A ;Drugi števec.
        CALL  NASLOV
        SLA   (HL)        ;Pomakni  zlog v levo. Skrajni levi
                          ;bit (bit 7) gre v bit prenosa.
```

Kodi nekaterih operacij za ukaz RST 28

01	zamenjaj	Zamenjaj zgornji dve števili na skladu.
02	odstrani	Odstrani zgornje število s sklada.
03	odštej	Odstrani zgornji dve števili in porini na sklad ... njuno razliko (odšteje zadnje število od predzadnjega).
04	pomnoži	... njun zmnožek.
05	deli	... rezultat deljenja predzadnjega števila z zadnjim.
06	potenciraj	... predzadnje število na potenco zadnjega.
0F	seštej	... njun seštevek.
17	z_seštej	podobno kot "seštej", le za seštevanje znakov.
18	VAL$	Nadomesti zgornje število z njegovim VAL$.
19	usr_z	Nadomesti zgornjo vrednost na skladu (zaporedje znakov) z vrednostjo USR tega zaporedja.
1B	neg	Nadomesti zgornje število z njegovo negativno vrednostjo.
31	podvoji	Porini na sklad še enkrat isto vrednost, ki je trenutno na vrhu.
38	končaj	Vrni se v normalni strojni program.
A0	konst_0	Porini na sklad število 0.
A1	konst_1	Porini na sklad število 1.
A2	konst_1/2	Porini na sklad število 1/2.
A3	konst_PI/2	Porini na sklad število PI/2 (1.5707963).
A4	konst_10	Porini na sklad število 10.
Cn	spravi_n	Spravi vrednost z vrha sklada v prostor n računskega pomnilnika.
En	vzemi_n	Porini na sklad vrednost v prostoru n računskega pomnilnika.

Kodi, ki slede, nadomestijo vrhnjo vrednost (oz. zaporedje znakov) na računskem skladu z rezultatom ustrezne operacije. Pomen operacij je enak kot v basicu.

1C code	20 cos	24 atn	28 sqr	2C in
1D val	21 tan	25 ln	29 sgn	2D usr št
1E len	22 asn	26 exp	2A abs	2E str$
1F sin	23 acs	27 int	2B peek	2F chr$

ime podprograma	naslov	učinek
STACK_A	2D28	pretvori celo število v registru A v 5-zložno obliko in ga porine na sklad.
STACK_BC	2D2B	enako kot STACK_A, vendar za število v registrskem paru BC.
STACK_AEDCB	2AB6	porine na sklad vrednost registrskega peterčka AEDCB.

Podprogrami za pobiranje s sklada so podobni:

ime podprograma	naslov	učinek
FP_TO_A	2DD5	naloži število z vrha računskega sklada v register A.
FP_TO_BC	2DA2	naloži število z vrha računskega sklada v registrski par BC.
FP_TO_AEDCB	2BF1	naloži število z vrha računskega sklada v registrski peterček.

Ob klicu FP_TO_AEDCB ne bo nikoli težav - 5 zlogov lahko vedno zložimo v 5 registrov. Drugače je z FP_TO_A in z FP_TO_BC. Na vrhu sklada je lahko število, ki je preveliko za v en register ali za v registrski par. To ugotavljamo z zastavicami. Če pri prelaganju ni nobenih težav, bo ničelna zastavica dvignjena, zastavica prenosa pa bo spuščena. Če je zastavica prenosa dvignjena, je bila absolutna vrednost števila prevelika. Če je ničelna zastavica spuščena, je bilo število negativno. V tem primeru bomo v registru kljub temu dobili pravilno absolutno vrednost števila.

V RAM-u Spectruma najdemo 30 zlogov prostora, ki je opisan v priročniku med sistemskimi spremenljivkami in nosi ime MEMBOT. To je računski pomnilnik. V njem je prostora za 6 vrednosti v plavajoči vejici. Z ukazom RST 28 lahko uporabljamo dva niza podatkovnih kod: Cn ter En, pri čemer ima n vrednost od 0 do 5. Niz C spravi vrednost z vrha računskega sklada v prostor n (vrednosti na skladu pri tem NE odstrani). Niz E vzame vrednost prostora n in jo porine na sklad.

Pri uporabi računskega pomnilnika je treba upoštevati dve opozorili. Nekatere funkcije (npr. SIN, COS, STR$, če omenimo samo tri) izbrišejo vrednosti v prostorih 0, 1 in 2 računskega

```
        DEC  HL          ;Naslov naslednjega zloga na levi.
        RL   (HL)        ;Pomakni zlog v levo. V desni bit
                         ;(bit 0) pride bit prenosa iz prej-
                         ;šnjega pomika, tj. bit 7 sosednje-
                         ;ga zloga. V bit prenosa se nato
                         ;prenese bit 7 zloga, ki ga zdaj
                         ;premikamo.
        DEC  HL          ;Naslednji zlog proti levi
        RL   (HL)        ;in še en pomik, nato pa...
        DEC  B           ;   ...v naslednjo vrstico navzdol.
        LD   A,(STEVC2)
        DEC  A           ;Se je že premaknila vsa figura?
        JR   NZ, ZAN4    ;Če se še ni, ponovi zanko.
        POP  BC          ;Lik se je pomaknil v levo, zato
        DEC  C           ;zmanjšaj koordinato x za 1.
        LD   A, (STEVC1) ;Se je pomik že ponovil osemkrat?
        DEC  A
        JR   NZ, LV      ;Če se še ni, ga ponovi, sicer...
        RET              ;         ...se vrni v glavno zanko.
```

Podprograma za premik v desno in navzdol sta sestavljena povsem podobno. Da se ne bomo preveč zamudili, se zato zdaj ob njiju ne bomo ustavljali. S potrebnimi opombami vred ju bomo vključili v končni zapis. Zdaj pa vzemimo v precep podprogram, ki uravnava gibanje metulja.

```
METULJ  EXX              ;Podatke o metulju smo naložili v
                         ;zamenljivi registrski niz.
        PUSH HL          ;To vrednost moramo shraniti za
                         ;nemoten povratek v basic (gl.
                         ;poglavje o menjavi registrov).
        LD   HL, GIB     ;V rezerviranem zlogu GIB bomo
                         ;šteli zamahe kril. Preden se sme
                         ;metulj premakniti drugam na ekranu
                         ;se morajo zvrstiti vse 4 sličice.
                         ;Zlogu GIB bomo dali vrednost 4 in
                         ;nato odštevali. Ko bodo opravljeni
                         ;vsi štirje gibi (tj. ko bo vred-
        DEC  (HL)        ;nost zloga GIB 0), bomo poklicali
        CALL Z, KAM      ;podprogram KAM, ki bo premikal
                         ;metulja po zaslonu.
```


Zamah metuljevih kril sestavljajo štiri slike, vsaka ima velikost 16 * 16 točk oziroma bitov. Če preračunamo, znese to 128 zlogov. Da jih bomo lahko prenašali na zaslon, jih bomo morali shraniti nekje v pomnilniku. Točen naslov bomo določili kasneje, za zdaj ga imenujmo LIKI2. Čisto na začetku bomo dali vrednost LIKI2 v par DE zamenljivega niza.

```
        LD   A, 16       ;Pripravi števec
        PUSH BC          ;in shrani koordinate metulja.
RACUN   PUSH AF          ;Števec bomo spravili kar na sklad.
        PUSH DE          ;Shrani naslov s katerega prenašamo
        CALL NASLOV      ;metuljevo podobo in izračunaj na-
        POP  DE          ;slov metuljevega zloga na zaslonu.
        LD   A, (DE)     ;Prenesi s pomočjo registra A zlog
        LD   (HL), A     ;slike iz pomnilnika na ekran.
        INC  DE          ;To sta naslova za...
        INC  HL          ;                ...desni zlog.
        LD   A, (DE)     ;Prenesi ustrezni zlog iz
        LD   (HL), A     ;pomnilnika na zaslon.
        INC  DE          ;Naslednji zlog v pomnilniku.
        DEC  B           ;Koordinata naslednje vrstice.
        POP  AF          ;Vzemi števec...
        DEC  A           ;          ...in ga zmanjšaj.
        JR   NZ, RACUN   ;Če še ni 0, ponovi zanko,
        POP  BC          ;sicer
        POP  HL          ;      "pospravi
        EXX              ;                za seboj"
        RET              ;in se vrni.
```

Naslednji korak je podprogram, ki bo premikal metulja po zaslonu. Da program ne bo postal predolg, se bomo odpovedali gladkemu gibanju; premiki bodo znašali 16 bitov naenkrat. Če pogledate začetek podprograma METULJ, boste videli, da ima ob klicu KAM registrski par HL vrednost GIB, vsebina zloga pa mora doseči 0, da pride do klica KAM. Da se bo program po vrnitvi lahko prav nadaljeval,

```
KAM     LD   A, 4        ;moramo najprej zlogu GIB
        LD   (HL), A     ;dati vrednost 4.
        PUSH BC          ;Shrani položaj (koordinati).
        LD   A, 16
```

```
ZANKA   PUSH  AF          ;Najprej bomo z ZANKO izbrisali
        CALL  NASLOV      ;sliko  metulja, tj. ustreznim zlo-
                          ;gom  na zaslonu bomo dali vrednost
        LD  (HL), A       ;0. Ker se koordinati vedno nanaša-
        INC  HL           ;ta  na  levi bit (bit 7), ima A po
        LD  (HL), A       ;klicu NASLOVa vrednost 0.
        DEC  B            ;Naslednja vrstica.
        POP  AF
        DEC  A            ;Je zanka že končana?
        JR  NZ, ZANKA     ;Če še ni, jo ponovi.
        POP  BC           ;Daj koordinatama začetno vrednost.
        EXX               ;Zamenjaj registre,
        PUSH  BC          ;shrani  koordinate Krpana na sklad
        EXX               ;in ponovno zamenjaj registre.
        POP  DE           ;Spravi koordinate Krpana v DE.
        LD  A, B          ;Primerjaj  koordinati  y junaka in
        CP  D             ;metulja.
        JR  Z, HORIZ      ;Če sta enaki, skoči naprej.
        JR  C, NAGOR      ;Če je Krpan više od metulja,
                          ;mora metulj navzgor,
NADOL   LD  A, B          ;sicer  pa  navzdol.  V tem primeru
        SUB  16           ;zmanjšaj koordinato y metulja
        LD  B, A
        JR  HORIZ         ;in skoči naprej.
NAGOR   LD  A, B
        ADD  A, 16        ;Za  dvig je treba višino povečati.
        LD  B, A
HORIZ   LD  A, C          ;Zdaj pa še primerjava koordinat x.
        CP  E             ;Če  je koordinata x metulja manjša
        JR  C, NADESN     ;od Krpanove, naj gre metulj desno,
NALEVO  LD  A, C          ;sicer pa levo. V tem primeru
        SUB  16           ;zmanjšaj oddaljenost od levega
        LD  C, A          ;roba in
        JR  VEN           ;skoči naprej.
NADESN  LD  A, C          ;Za pomik v desno...
        ADD  A, 16        ;...povečaj  oddaljenost  od levega
        LD  C, A           roba.
```

Zdaj moramo preveriti, ali ni metulj ob premiku zadel ključa ali junaka igre. Samo testiranje bo tu nekoliko bolj zapleteno, zato ga bomo oblikovali kot podprogram z naslovom METST

(MEtuljev TeST). Za testiranje bomo uporabljali četrte zloge na posameznih položajih.

```
VEN     PUSH  BC          ;Shrani koordinate.
        LD  A, B
        SUB  4            ;Koordinata četrtega zloga.
        LD  B, A
        CALL  NASLOV      ;Izračunaj njen naslov ter kliči
        CALL  METST       ;podprogram za testiranje.
        INC  HL           ;Naslov sosednjega zloga -
        CALL  METST       ;preveri njegovo vsebino.
        LD  A, B          ;Zdaj pa še dva zloga, ki sta
        SUB  8            ;osem vrstic niže.
        LD  B, A
        CALL  NASLOV      ;Ponovi isti postopek.
        CALL  METST
        INC  HL
        CALL  METST
        POP  BC           ;Poberi koordinate s sklada.
        LD  DE, LIKI2     ;Na naslovu LIKI2 so shranjene
                          ;slike metulja.
        RET               ;Vrni se v zanko METULJ.
```

S podprogramom METST ugotavljamo, ali je metulj zadel ključ ali junaka še preden se je metulj v resnici premaknil. Seveda želimo na zaslonu videt, kako metulj "požre" junaka in bomo šele potem zaključili igro. Zato bomo v podprogramu uporabili rezerviran zlog TEST1. Če bo metulj zadel Krpana, bo TEST1 dobil vrednost 1. Ta zlog bomo kontrolirali, ko bo premik že opravljen - in ustrezno ukrepali. Vendar moramo zaradi tega nekoliko dopolniti konec podprograma METULJ, in sicer bomo za ukazom EXX dodali tele vrstice:

```
        LD  HL, TEST1     ;Preveri, ali je metulj zadel
        DEC (HL)          ;Krpana. Če ga ni,...
        RET NZ            ;...se vrni.
        CALL KONEC        ;Sicer pokliči podprogram, ki bo
                          ;zaključil igro.
```

Zdaj pa k podprogramu METST. Katere zloge oz. bite testiramo v tem podprogramu boste najlaže razumeli, če boste pogledali slike naših likov.

```
METST   BIT  2, (HL)
        JR   Z, P1
        LD   A, 1
        LD   (TEST1), A
P1      BIT  5, (HL)       ;Če je "prižgan" bit 2 ali 5, to
        JR   Z, P2         ;pomeni, da je metulj zadel junaka.
        LD   A, 1          ;V tem primeru daj zlogu TEST1
        LD   (TEST1), A    ;vrednost 1. Če pa je "prižgan"
P2      BIT  3, (HL)       ;bit 3, je to znak, da je zadel
        CALL NZ, MINUS     ;ključ. V tem primeru kliči
        RET                ;podprogram MINUS ter se nato vrni.
```

Če metulj "pobere" ključ, naj preostane igralcu za kazen manj časa. Da pa bo igra tekla kot je prav, bomo ob tem povečali vrednost zloga KLJUC, ki hrani število že pobranih ključev.

```
MINUS   PUSH BC            ;Shrani koordinate.
        LD   HL, KLJUC
        INC  (HL)          ;Povečaj števec pobranih ključev.
        LD   B, 30         ;Preostali čas naj se zmanjša za
VIBA    PUSH BC            ;30. Pripravi zanko in spravi
                           ;števec v registru B na sklad.
        LD   BC, (VISURE)  ;Koordinate stolpca ure.
        CALL NASLOV        ;Izračunaj naslov vrhnjega zloga
                           ;ure.
        DEC  B             ;Zmanjšaj višino stolpca.
        JR   NZ, NAPR      ;Če višina še ni 0 (tj. ura še ni
                           ;potekla), skoči na naslov NAPR.
```

Tu pa se srečamo s težavo. Kaj storiti, če se na tem mestu izkaže, da je igralcu čas potekel? Takoj bi bilo potrebno končat igro in se vrnit v basic, a zdi se, da bi bil to precej zapleten posel. Če se spomnite, smo iz glavne zanke klicali podprogram METULJ (in tam zamenjali registrski niz), od tam KAM, iz podprograma KAM podprogram METST ter iz njega MINUS, v katerem smo zdaj. Vendar je naše znanje kos na videz zelo zapletenemu vozlu. Najprej bomo dali registrskemu paru zamenljivega niza vrednost 10072, ki je potrebna za...

```
        LD   HL, 10072     ;...nemoteno vrnitev v basic
        EXX                ;ter zamenjali registrski niz.
        LD   BC, (TOCKE)   ;V par BC naloži dosežene točke.
```


Ob klicih podprogramov so se na skladu nabirali naslovi za povratke. Da bi prišli do pravega naslova za vrnitev v basic, bi torej morali te naslove odstraniti. Lahko pa si pomagamo na preprostejši način. Shranimo na začetku programa naslov, na katerem je naslov za vrnitev v basic, v rezervirana zloga STKPT! Ko se bomo želeli vrniti, bomo dali kazalcu sklada to vrednost:

```
        LD   SP, (STKPT)
        RET                ;ter se brez težav vrnili.
```

Če pa ura še ni potekla, naj se program nadaljuje na NAPR:

```
NAPR    LD   (VISURE), BC  ;Shrani zmanjšano višino ter
        LD   (HL), 129     ;zbriši zgornjo vrstico ure.
        POP  BC            ;Vzemi števec s sklada in če še ni
        DJNZ VIBA          ;nič, ponovi zanko.
        XOR  A             ;Daj uri začetno vrednost,
        LD   (23672), A
        POP  BC            ;vzemi koordinate in
        RET                ;se vrni.
```

Zdaj pa smo že blizu konca. Manjka nam le še nekaj podprogramov, ki smo jih že "uporabili", nimamo pa jih še zapisanih. Prvi je podprogram ZADET, ki ga bomo klicali, kadar bo junak pobral ključ. KONEC bomo klicali, kadar se bo Krpan zadel v metulja ali obratno, OBRAC pa za obračun točk, kadar se bo igra uspešno končala. Vendar se še prej za trenutek ustavimo pri dveh programih iz ROM-a, ki ju bomo uporabili. Pri obračunu bomo klicali podprogram z zgovornim naslovom "HL=HL*DE" na naslovu 30A9. Ta nam vrne v registrskem paru HL zmnožek vrednosti, ki ju pred klicem vsebujeta HL in DE. Seveda morata biti števili takšni, da njun zmnožek ni večji od 65535. Če pa se vseeno zgodi, da sta vrednosti preveliki, je ob povratku zastavica prenosa dvignjena. "HL=HL*DE" spremeni vrednost v paru BC. Drugi podprogram, ki ga kanimo klicat iz ROM-a, pa je BEEPER na naslovu 03B5. Ta nadzoruje zvočnik in daje zvok, ki ga opredelimo z vrednostima v parih DE in HL. V DE mora biti zmnožek frekvence in trajanja želenega tona, v HL pa število potrebnih T stanj, deljeno s štiri (če vam ni povsem jasno, se ne vznemirjajte: ni pomembno).

Podprogram ZADET (igralec je pobral ključ):

```
ZADET   PUSH  BC          ;Shrani vrednosti v parih BC in HL.
        PUSH  HL
        LD  B, 8          ;Pripravi števec...
ZVOK    PUSH  BC          ;in ga shrani. Podrobnosti te zanke
        LD  DE, #0001     ;niso pomembne. Pripravljena je
        LD  HL, #006A     ;bila s poskušanjem, ker smo želeli
        CALL  #03B5       ;dobit kovinski zvok trka ob ključ.
        POP  BC
        DJNZ  ZVOK
        LD  HL, (TOCKE)   ;Daj v HL že dosežene točke, v
        LD  DE, 50        ;DE pa vrednost enega ključa.
        ADD  HL, DE       ;Povečaj število točk in...
        LD  (TOCKE), HL   ;...ga shrani.
        LD  HL, KLJUC
        INC  (HL)         ;Povečaj število pobranih ključev,
        POP  HL           ;Poberi vrednosti s sklada
        POP  BC
        RET               ;in se vrni.
```

Sledi podprogram za zaključek v primeru trka med Krpanom in metuljem. Najprej bomo povzročili utripanje zaslona, nato eksplozijo ter se nato vrnili v basic.

```
KONEC   LD  HL, #22AA     ;Tu bi lahko uporabili katerokoli
        LD  DE, 22528     ;vrednost, nižjo od 4000H. V DE
        LD  BC, 704       ;naslov začetka zlogov s prilastki,
                          ;v BC število zlogov, nato pa...
        LDIR              ;  ...premakni vrednosti. To bo
                          ;povzročilo utripanje na zaslonu.
        LD  DE, 31488     ;= LD D, 123 / LD E, 0 (vendar
KROZI   PUSH  DE          ;prihranimo en zlog).
MALAZN  LD  B, E          ;Tudi tu se ne bomo ukvarjali s
PAVZA   DJNZ  PAVZA       ;podrobnostmi zanke. Povejmo le,
        LD  A, (BC)       ;da s spreminjanjem vrednosti v
        OUT  (254), A     ;registru D (mi smo uporabili 123)
        INC  C            ;spreminjamo trajanje eksplozije.
        DEC  D            ;Zanko lahko uporabite za šum v
        JR  NZ, MALAZN    ;svojih programih. Če ne želite
        POP  DE           ;prog na robu, dodajte za ukazom
        INC  E            ;LD A, (BC) ukaz OR 7.
        DEC  D            ;Če zanka še ni končana,
        JR  NZ, KROZI     ;skoči nazaj.
```



```
        LD  BC, (TOCKE)   ;Naloži v BC dosežene točke,
        LD  SP, (STKPT)   ;in se vrni
        RET               ;v basic.
```

In končno še zadnji podprogram. Če igralec pobere vse ključe pred iztekom časa, se mu število točk za pobrane ključe pomnoži z vrednostjo preostalega časa (višino stolpca). Koordinate stolpca smo imeli spravljene na naslovu VISURE. Naj bo VIS naslov visokega zloga (v katerem je dejansko spravljena višina; nizki zlog ima namreč vrednost 0).

```
OBRAC   LD  HL, (TOCKE)   ;Naloži v HL dosežene točke.
        LD  A, (VIS)      ;Prenesi višino stolpca ure s
        LD  E, A          ;pomočjo registra A v register E.
        LD  D, 0
        CALL  #30A9       ;Kliči HL=HL*DE.
        PUSH  HL          ;Prenesi zmnožek v registrski
        POP  BC           ;par BC, nato pa poberi s sklada še
        POP  HL           ;naslov za povratek v glavno zanko
        RET               ;in se vrni naravnost v basic.
```

Naš program v zbirnem jeziku je končan! Na vrsti je prvi preskus, vendar zanj potrebujemo tudi del programa v basicu. Takšen je:

```
 5 LET a=0: CLS : FOR x=1 TO 3: BORDER (3+x): CLS
10 PRINT #1;"Življenj ";(4-x);"  Točk ";a: PAUSE 20
15 FOR i=1 TO 7
20 LET kx=1+INT (RND*31): LET ky=INT (RND*21)
25 IF kx<5 AND ky>18 THEN GO TO 20
30 IF (kx>14 AND kx<17) AND (ky>9 AND ky<12) THEN GO TO 20
35 PRINT AT ky,kx;"$": NEXT i
40 FOR i=0 TO 21: PRINT AT i,0; INK 2;"%": NEXT i:
   LET a=a+USR 31600
45 CLS : NEXT x: PAUSE 10: BORDER 7: CLS
50 PRINT AT 8,5;"DOSEGEL SI ";a;" TOCK.": IF a>hi THEN
   PRINT AT 10,7; BRIGHT 1;"TO JE NOV REKORD !": LET hi=a
55 PRINT AT 21,3;"ŽELIŠ ŠE ENO IGRO ? (D/N)"; PAUSE 0
60 IF INKEY$="" THEN GO TO 60
65 IF INKEY$="D" OR INKEY$="d" THEN CLS : GO TO 1
70 GO TO 9999
90 LOAD "igra"CODE : LOAD "znaki"CODE : LET hi=0: GO TO 1
```

V spremenljivki a bomo šteli dosežene točke, spremenljivka hi pa bo hranila najvišjo doseženo vrednost. Število življenj lahko uravnavamo v vrsti 5 s števcem x. Zaenkrat so določena tri življenja. Vrsta 10 napiše na spodnji vrstici zaslona število življenj ter točk. V vrstah od 15 do 35 razpostavljamo ključe. Denimo, da bo začetni položaj Krpana na "PRINT položaju" 19,2, metulja pa na 10,15 (vsak zavzame 4 mesta). Na teh mestih ne sme biti ključev - za to skrbita vrstici 25 in 30. Znak "$" v vrsti 35 bomo nadomestili z grafičnim znakom ključa. Vrsta 40 bo postavila stolpec ure (znak "%" bo treba zamenjati z grafičnim črnim kvadratkom) in poklicala glavni del - strojni program. V njem smo ob zaključkih povsod dali paru BC vrednost doseženih točk, kar nam zelo olajša delo v basicu. Ko so porabljena vsa življenja, pridejo na vrsto vrste od 50 do 70. Program shranimo z ukazom

SAVE "igra" LINE 90.

Z vrsto 90 bomo naložili strojni program ("igra" CODE) ter grafične znake - ključ ter črke č, ž in š ("znaki" CODE).

ODMOR

Med odmorom smo preizkusili igro (lepo nam je uspela!), dopolnili potrebne podrobnosti - in pred vami je zdaj dokončni izpis programa s šestnajstiškimi kodi, tako da ga boste lahko vpisali s pomočjo urejevalnika! Najprej so nanizane prave vrednosti naslovov, ki smo jih do sedaj označevali z imeni:

```
CLOCK  EQU   23672
VISURE EQU   31580
VIS    EQU   31581
STEVC1 EQU   31582
STEVC2 EQU   31583
TOCKE  EQU   31584
KLJUC  EQU   31586
TEST1  EQU   31587
STKPT  EQU   31588
GIB    EQU   31590
STVME  EQU   31591
LIKI1  EQU   31400
LIKI2  EQU   31432
```


Pri tistih delih programa, o katerih smo že obširneje govorili, se ne bomo več posebej ustavljali. Z opombami bomo pospremili le na novo dodane ali po testiranju spremenjene vrste.

```
31600 ED73647B        LD    (STKPT), SP ;Shranimo vrednost
                                        ;kazalca sklada - za
                                        ;povratke iz podpro-
                                        ;gramov v basic.
31604 D9              EXX               ;Vrednosti metulja:
31605 11C87A          LD    DE, LIKI2   ;naslov zlogov slike
31608 01785F          LD    BC, 24440   ;in koordinate (B =
31611 D9              EXX               ;95, C = 120).
31612 0100AF          LD    BC, 44800   ;Koordinate ure (B =
31615 ED435C7B        LD    (VISURE), BC;175, C = 0).
31619 010000          LD    BC, 0       ;Začetna vrednost
31622 ED43607B        LD    (TOCKE), BC ;točk.
31626 3E04            LD    A, 4        ;Števec zamahov me-
31628 32667B          LD    (GIB), A    ;tuljevih kril.
31631 AF              XOR   A           ;= LD A, 0
31632 32677B          LD    (STVME), A
31635 32637B          LD    (TEST1), A
31638 32627B          LD    (KLJUC), A
31641 32785C          LD    (CLOCK), A
31644 11A87A          LD    DE, LIKI1   ;Naslov Krpanove
31647 011017          LD    BC, 5904    ;slike ter koordina-
31650 C5              PUSH  BC          ;te: B 23, C 16.
31651 3E10            LD    A, 16       ;Zanka NAZAJ prenese
31653 F5        NAZAJ PUSH  AF          ;sliko junaka iz
31654 D5              PUSH  DE          ;pomnilnika na za-
31655 CD497E          CALL  NASLOV      ;slon (tj. nariše
31658 D1              POP   DE          ;Krpana).
31659 1A              LD    A, (DE)
31660 77              LD    (HL), A
31661 13              INC   DE
31662 23              INC   HL
31663 1A              LD    A, (DE)
31664 77              LD    (HL), A
31665 13              INC   DE
31666 05              DEC   B
31667 F1              POP   AF
```

```
31668  3D              DEC   A
31669  20EE            JR    NZ, NAZAJ
31671  CD8E02   ODMOR  CALL  #028E       ;ODMOR čaka, da pri-
31674  3EFF            LD    A, 255      ;tisnemo neko tipko
31676  BB              CP    E           ;za začetek (#028E =
31677  28F8            JR    Z, ODMOR    ;KEY_SCAN).
31679  C1              POP   BC
31680  3A785C   URA    LD    A, (CLOCK)
31683  FE03            CP    3           ;Ta vrednost uravna-
                                         ;va  hitrost ure. Iz
                                         ;basica jo spremeni-
                                         ;te s POKE 31684, n.
31685  381A            JR    C, TEST
31687  C5              PUSH  BC
31688  ED4B5C7B        LD    BC, (VISURE)
31692  CD497E          CALL  NASLOV
31695  3681            LD    (HL), 129
31697  32785C          LD    (CLOCK), A
31700  05              DEC   B
31701  ED435C7B        LD    (VISURE), BC
31705  C1              POP   BC
31706  2005            JR    NZ, TEST
31708  ED4B607B        LD    BC, (TOCKE)
31712  C9              RET
31713  C5       TEST   PUSH  BC
31714  CD8E02          CALL  #028E       ;Kliči KEY_SCAN.
31717  C1              POP   BC
31718  7B              LD    A, E
31719  FE22            CP    #22
31721  CC4E7C          CALL  Z, DESNO
31724  FE1A            CP    #1A
31726  CC927C          CALL  Z, LEVO
31729  FE25            CP    #25
31731  CC127C          CALL  Z, GOR
31734  FE26            CP    #26
31736  CCDA7C          CALL  Z, DOL
```

Pri preizkušanju smo ugotovili, da je gibanje metulja prehitro, če kličemo podprogram METULJ ob vsakem izvajanju zanke. Zato smo dodali števec STVME, ki uravnava hitrost metulja. Kot bomo videli, lahko tudi to vrednost spremenimo iz basica.


```
31762  3EAF     GOR    LD    A, 175
31764  B8              CP    B
31765  C8              RET   Z
31766  C5              PUSH  BC
31767  04              INC   B
31768  04              INC   B
31769  04              INC   B
31770  CD497E          CALL  NASLOV
31773  CD4F7E          CALL  BIT40
31776  23              INC   HL
31777  CD4F7E          CALL  BIT40
31780  C1              POP   BC
31781  3E08            LD    A, 8
31783  325E7B   GR     LD    (STEVC1), A
31786  04              INC   B
31787  C5              PUSH  BC
31788  3E10            LD    A, 16
31790  325F7B   ZAN1   LD    (STEVC2), A
31793  CD497E          CALL  NASLOV
31796  E5              PUSH  HL
31797  05              DEC   B
31798  CD497E          CALL  NASLOV
31801  D1              POP   DE
31802  7E              LD    A, (HL)
31803  12              LD    (DE), A
31804  23              INC   HL
31805  13              INC   DE
31806  7E              LD    A, (HL)
31807  12              LD    (DE), A
31808  3A5F7B          LD    A, (STEVC2)
31811  3D              DEC   A
31812  20E8            JR    NZ, ZAN1
31814  C1              POP   BC
31815  3A5E7B          LD    A, (STEVC1)
31818  3D              DEC   A
31819  20DA            JR    NZ, GR
31821  C9              RET
```

Podprogramov za premik v desno in navzdol nismo posebej obravnavali. Vendar je njuna zgradba enaka kot za pomik v levo ali navzgor. Opaznejša je le razlika v testiranju: v GOR in

LEVO smo preverjali bit 4 in bit 0 izbranega zloga, v DESNO in DOL pa pogledamo stanje bitov 3 in 0 oz. 3, 4 in 0.

```
31822  3EF0       DESNO  LD    A, 240
31824  B9                CP    C
31825  C8                RET   Z
31826  C5                PUSH  BC
31827  79                LD    A, C
31828  C610              ADD   A, 16
31830  4F                LD    C, A
31831  78                LD    A, B
31832  D605              SUB   5
31834  47                LD    B, A
31835  CD497E            CALL  NASLOV
31838  CD5A7E            CALL  BIT30
31841  78                LD    A, B
31842  D608              SUB   8
31844  47                LD    B, A
31845  CD497E            CALL  NASLOV
31848  CD5A7E            CALL  BIT30
31851  C1                POP   BC
31852  3E08              LD    A, 8
31854  325E7B     DS     LD    (STEVC1), A
31857  C5                PUSH  BC
31858  3E10              LD    A, 16
31860  325F7B     ZAN3   LD    (STEVC2), A
31863  CD497E            CALL  NASLOV
31866  CB3E              SRL   (HL)
31868  23                INC   HL
31869  CB1E              RR    (HL)
31871  23                INC   HL
31872  CB1E              RR    (HL)
31874  05                DEC   B
31875  3A5F7B            LD    A, (STEVC2)
31878  3D                DEC   A
31879  20EB              JR    NZ, ZAN3
31881  C1                POP   BC
31882  0C                INC   C
31883  3A5E7B            LD    A, (STEVC1)
31886  3D                DEC   A
31887  20DD              JR    NZ, DS
31889  C9                RET
```



```
31890  3E08       LEVO   LD    A, 8
31892  B9                CP    C
31893  C8                RET   Z
31894  C5                PUSH  BC
31895  79                LD    A, C
31896  D608              SUB   8
31898  4F                LD    C, A
31899  78                LD    A, B
31900  D605              SUB   5
31902  47                LD    B, A
31903  CD497E            CALL  NASLOV
31906  CD4F7E            CALL  BIT40
31909  78                LD    A, B
31910  D608              SUB   8
31912  47                LD    B, A
31913  CD497E            CALL  NASLOV
31916  CD4F7E            CALL  BIT40
31919  C1                POP   BC
31920  3E08              LD    A, 8
31922  325E7B     LV     LD    (STEVC1), A
31925  C5                PUSH  BC
31926  79                LD    A, C
31927  C60F              ADD   A, 15
31929  4F                LD    C, A
31930  3E10              LD    A, 16
31932  325F7B     ZAN4   LD    (STEVC2), A
31935  CD497E            CALL  NASLOV
31938  CB26              SLA   (HL)
31940  2B                DEC   HL
31941  CB16              RL    (HL)
31943  2B                DEC   HL
31944  CB16              RL    (HL)
31946  05                DEC   B
31947  3A5F7B            LD    A,(STEVC2)
31950  3D                DEC   A
31951  20EB              JR    NZ, ZAN4
31953  C1                POP   BC
31954  0D                DEC   C
31955  3A5E7B            LD    A, (STEVC1)
31958  3D                DEC   A
31959  20D9              JR    NZ, LV
31961  C9                RET
```

```
31962  3E0F      DOL   LD    A, 15
31964  B8              CP    B
31965  C8              RET   Z
31966  C5              PUSH  BC
31967  78              LD    A, B
31968  D615            SUB   21
31970  47              LD    B, A
31971  CD497E          CALL  NASLOV
31974  CB5E            BIT   3, (HL)
31976  C4127E          CALL  NZ, KONEC
31979  CD4F7E          CALL  BIT40
31982  23              INC   HL
31983  CB5E            BIT   3, (HL)
31985  C4127E          CALL  NZ, KONEC
31988  CD4F7E          CALL  BIT40
31991  C1              POP   BC
31992  3E08            LD    A, 8
31994  325E7B    DL    LD    (STEVC1), A
31997  05              DEC   B
31998  C5              PUSH  BC
31999  78              LD    A, B
32000  D60F            SUB   15
32002  47              LD    B, A
32003  3E10            LD    A, 16
32005  325F7B    ZAN2  LD    (STEVC2), A
32008  CD497E          CALL  NASLOV
32011  E5              PUSH  HL
32012  04              INC   B
32013  CD497E          CALL  NASLOV
32016  D1              POP   DE
32017  7E              LD    A, (HL)
32018  12              LD    (DE), A
32019  23              INC   HL
32020  13              INC   DE
32021  7E              LD    A, (HL)
32022  12              LD    (DE), A
32023  3A5F7B          LD    A, (STEVC2)
32026  3D              DEC   A
32027  20E8            JR    NZ, ZAN2
32029  C1              POP   BC
32030  3A5E7B          LD    A, (STEVC1)
```



```
32033  3D              DEC   A
32034  20D6            JR    NZ, DL
32036  C9              RET
32037  D9        METULJ EXX
32038  E5              PUSH  HL
32039  21667B          LD    HL, GIB
32042  35              DEC   (HL)
32043  CC537D          CALL  Z, KAM
32046  3E10            LD    A, 16
32048  C5              PUSH  BC
32049  F5        RACUN PUSH  AF
32050  D5              PUSH  DE
32051  CD497E          CALL  NASLOV
32054  D1              POP   DE
32055  1A              LD    A, (DE)
32056  77              LD    (HL), A
32057  13              INC   DE
32058  23              INC   HL
32059  1A              LD    A, (DE)
32060  77              LD    (HL), A
32061  13              INC   DE
32062  05              DEC   B
32063  F1              POP   AF
32064  3D              DEC   A
32065  20EE            JR    NZ, RACUN
32067  3E64            LD    A, 100      ;Hitrost metulja. Iz
32069  32677B          LD    (STVME), A  ;basica jo spremeni-
32072  C1              POP   BC          ;mo s POKE 32068, n.
32073  E1              POP   HL
32074  D9              EXX
32075  21637B          LD    HL, TEST1
32078  35              DEC   (HL)
32079  C0              RET   NZ
32080  CD127E          CALL  KONEC
32083  3E04      KAM   LD    A, 4
32085  77              LD    (HL), A
32086  C5              PUSH  BC
32087  3E10            LD    A, 16
32089  F5        ZANKA PUSH  AF
32090  CD497E          CALL  NASLOV
32093  77              LD    (HL), A
```

```
32094  23              INC   HL
32095  77              LD    (HL), A
32096  05              DEC   B
32097  F1              POP   AF
32098  3D              DEC   A
32099  20F4            JR    NZ, ZANKA
32101  C1              POP   BC
32102  D9              EXX
32103  C5              PUSH  BC
32104  D9              EXX
32105  D1              POP   DE
32106  78              LD    A, B
32107  BA              CP    D
32108  280C            JR    Z, HORIZ
32110  3806            JR    C, NAGOR
32112  78        NADOL LD    A, B
32113  D610            SUB   16
32115  47              LD    B, A
32116  1804            JR    HORIZ
32118  78        NAGOR LD    A, B
32119  C610            ADD   A, 16
32121  47              LD    B, A
32122  79        HORIZ LD    A, C
32123  BB              CP    E
32124  3806            JR    C, NADESN
32126  79        NALEVO LD   A, C
32127  D610            SUB   16
32129  4F              LD    C, A
32130  1804            JR    VEN
32132  79        NADESN LD   A, C
32133  C610            ADD   A, 16
32135  4F              LD    C, A
32136  C5        VEN   PUSH  BC
32137  78              LD    A, B
32138  D604            SUB   4
32140  47              LD    B, A
32141  CD497E          CALL  NASLOV
32144  CDAA7D          CALL  METST
32147  23              INC   HL
32148  CDAA7D          CALL  METST
32151  78              LD    A, B
```



```
32152  D608            SUB   8
32154  47              LD    B, A
32155  CD497E          CALL  NASLOV
32158  CDAA7D          CALL  METST
32161  23              INC   HL
32162  CDAA7D          CALL  METST
32165  C1              POP   BC
32166  11C87A          LD    DE, LIKI2
32169  C9              RET
32170  CB56      METST BIT   2, (HL)
32172  2805            JR    Z, P1
32174  3E01            LD    A, 1
32176  32637B          LD    (TEST1), A
32179  CB6E      P1    BIT   5, (HL)
32181  2805            JR    Z, P2
32183  3E01            LD    A, 1
32185  32637B          LD    (TEST1), A
32188  CB5E      P2    BIT   3, (HL)
32190  C4C27D          CALL  NZ, MINUS
32193  C9              RET
32194  C5        MINUS PUSH  BC
32195  21627B          LD    HL, KLJUC
32198  34              INC   (HL)
32199  061E            LD    B, 30
32201  C5        VIBA  PUSH  BC
32202  ED4B5C7B        LD    BC, (VISURE)
32206  CD497E          CALL  NASLOV
32209  05              DEC   B
32210  200D            JR    NZ, NAPR
32212  215827          LD    HL, 10072
32215  D9              EXX
32216  ED4B607B        LD    BC, (TOCKE)
32220  ED7B647B        LD    SP, (STKPT)
32224  C9              REI
32225  ED435C7B  NAPR  LD    (VISURE), BC
32229  3681            LD    (HL), 129
32231  C1              POP   BC
32232  10DF            DJNZ  VIBA
32234  AF              XOR   A
32235  32785C          LD    (CLOCK), A
32238  C1              POP   BC
32239  C9              RET
```

```
32240  C5          ZADET  PUSH  BC
32241  E5                 PUSH  HL
32242  0608               LD    B, 8
32244  C5          ZVOK   PUSH  BC
32245  110100             LD    DE, #0001
32248  216A00             LD    HL, #006A
32251  CDB503             CALL  #03B5        ;Kliči BEEPER.
32254  C1                 POP   BC
32255  10F3               DJNZ  ZVOK
32257  2A607B             LD    HL, (TOCKE)
32260  113200             LD    DE, 50
32263  19                 ADD   HL, DE
32264  22607B             LD    (TOCKE), HL
32267  21627B             LD    HL, KLJUC
32270  34                 INC   (HL)
32271  E1                 POP   HL
32272  C1                 POP   BC
32273  C9                 RET
32274  21AA22      KONEC  LD    HL, #22AA
32277  110058             LD    DE, 22528
32280  01C002             LD    BC, 704
32283  EDB0               LDIR
32285  11007B             LD    DE, 31488
32288  D5          KROZI  PUSH  DE
32289  43          MALAZN LD    B, E
32290  10FE        PAVZA  DJNZ  PAVZA
32292  0A                 LD    A, (BC)
32293  D3FE               OUT   (254), A
32295  0C                 INC   C
32296  15                 DEC   D
32297  20F6               JR    NZ, MALAZN
32299  D1                 POP   DE
32300  1C                 INC   E
32301  15                 DEC   D
32302  20F0               JR    NZ, KROZI
32304  ED4B607B           LD    BC, (TOCKE)
32308  ED7B647B           LD    SP, (STKPT)
32312  C9                 RET
32313  2A607B      OBRAC  LD    HL, (TOCKE)
32316  3A5D7B             LD    A, (VIS)
32319  5F                 LD    E, A
```



```
32320  1600               LD    D, 0
32322  CDA930             CALL  #30A9        ;Kliči HL=HL*DE.
32325  E5                 PUSH  HL
32326  C1                 POP   BC
32327  E1                 POP   HL
32328  C9                 RET
32329  C5          NASLOV PUSH  BC
32330  CDAA22             CALL  #22AA        ;Kliči PIXEL_ADD.
32333  C1                 POP   BC
32334  C9                 RET
32335  CB66        BIT40  BIT   4, (HL)
32337  C4127E             CALL  NZ, KONEC
32340  CB46               BIT   0, (HL)
32342  C4F07D             CALL  NZ, ZADET
32345  C9                 RET
32346  CB5E        BIT30  BIT   3, (HL)
32348  C4127E             CALL  NZ, KONEC
32351  CB46               BIT   0, (HL)
32353  C4F07D             CALL  NZ, ZADET
32356  C9                 RET
```

Takšna je torej naša igra v - vsaj zaenkrat - dokončni obliki. Pravim: "zaenkrat". Kajti tako kot vsi primeri in programi, ki smo si jih ogledali, tudi igra "Ključar Martin in vražji metulj" čaka na vaše dopolnitve in izboljšave. Tu in tam boste program lahko skrajšali, lahko boste dodali barve ali zvok, dopolnili pravila... Ne dvomim, da boste imeli mnogo zanimivih predlogov! Vsekakor pa moramo najprej preskusiti, kar smo ravnokar pripravili. Priporočam vam, da ravnate takole:

- najprej shranite na kaseto program v basicu. Namesto ključa zaenkrat uporabite grafični znak (CAPS SHIFT + 9) črke A, namesto č grafični znak B, namesto ć C, š D, ś E, ž F ter namesto ź grafični znak G. Nato
- naložite urejevalnik in z njim vpišite strojni program (šestnajstiški kod). Ko boste končali,
- naložite naslednji pomožni program in ga poženite:

```
10 DATA 0,0,7,224,31,248,5,160,7,224,6,96,3,192,31,248,63,
   252,55,236,55,236,39,228,6,96,14,112,30,120,0,0
15 DATA 0,0,0,0,0,0,16,8,24,24,12,48,14,112,1,128,3,192,7,
   224,9,144,50,76,4,32,24,24,0,0,0,0
```

```
20 DATA 0,0,0,0,2,64,6,96,14,112,14,112,2,64,1,128,3,192,7,
   224,9,144,50,76,4,32,24,24,0,0,0,0
25 DATA 0,0,0,0,0,0,0,0,48,12,24,24,28,56,14,112,1,128,3,
   192,15,240,17,136,38,100,24,24,0,0,0,0
30 DATA 0,0,0,0,0,0,0,0,0,0,28,56,62,124,14,112,1,128,3,192,
   15,240,49,140,6,96,24,24,0,0,0,0
35 CLEAR 31399:FOR i = 31400 TO 31559: READ a: POKE i, a:
   NEXT i
40 DATA 96,144,144,112,10,5,2,0
45 DATA 20,8,28,32,32,32,28,0,24,60,66,64,64,66,60,0
50 DATA 20,8,28,32,28,2,60,0,24,60,64,60,2,66,60,0
55 DATA 40,16,124,8,16,32,124,0,24,126,4,8,16,32,126,0
60 FOR i = USR "a" TO USR "h"-1: READ a: POKE i, a: NEXT i
```

Zdaj shranite zloge z ukazoma

```
    SAVE "igra" CODE 31400, 970
```

in

```
    SAVE "znaki" CODE USR "a", 64.
```

Shranite zloge strojnega programa ter grafičnih znakov takoj za prvim programom v basicu! Le še RUN ...

Želim vam obilo zabave! In upam, da vam bo pridobljeno znanje spodbuda in opora za nove poskuse. Ta knjiga je šele začetek poti - zato vam želim tudi: "Srečno pot!".

DODATEK

UREJEVALNIK ZA VNOS STROJNEGA KODA

```
 10 DEF FN A(D$)=16*(CODE D$(1)-48-7*(CODE D$(1)>57))
    +(CODE D$(2)-48-7*(CODE D$(2)>57))
 20 DEF FN A$(A)=CHR$ (INT (A/16)+48+7*(INT (A/16)>9))
    +CHR$ (A-16*INT (A/16)+48+7*((A-16*INT (A/16))>9))
100 INPUT "START ? ";NASLOV: RANDOMIZE NASLOV:
    CLEAR (NASLOV-1): LET NASLOV=PEEK 23670+256*PEEK 23671:
    POKE 23658,8
120 PRINT "NASLOV";TAB 12;"VSEBINA"'TAB 10;"STARA  NOVA":
    LET SPOMIN=NASLOV
130 FOR J=0 TO 15: PRINT NASLOV;TAB 12;FN A$(PEEK NASLOV);:
    INPUT C$
140 IF C$="" THEN PRINT TAB 18;FN A$(PEEK NASLOV):
    LET NASLOV=NASLOV+1: NEXT J: GO TO 180
150 IF C$="C9" THEN POKE NASLOV,FN A(C$):
    PRINT TAB 18;C$: GO TO 180
170 FOR I=1 TO LEN C$ STEP 2: LET D$=C$(I TO I+1):
    POKE NASLOV,FN A(D$): PRINT TAB 18;FN A$(PEEK NASLOV):
    LET NASLOV=NASLOV+1:LET J=J+1: NEXT I: NEXT J
180 INPUT "SPREMEMBE ? ";C$: IF C$<>"N" THEN LET NASLOV=SPOMIN:
    CLS : GO TO 120
190 INPUT "NAPREJ ? ";C$: IF C$<>"N" THEN LET NASLOV=NASLOV-1:
    CLS : GO TO 120
200 CLS : INPUT "START ZA USR ? "; LINE S$:
    IF S$=" STOP " THEN GO TO 9999
210 PRINT USR VAL S$
```

STROJNI UKAZI PROCESORJA Z80 IN NJIHOVI KODI

ADC A, (HL)	8E	ADD IY, IY	FD 29
ADC A, (IX+dis)	DD 8E XX	ADD IY, SP	FD 39
ADC A, (IY+dis)	FD 8E XX	AND A, (HL)	A6
ADC A, A	8F	AND A, (IX+dis)	DD A6 XX
ADC A, B	88	AND A, (IY+dis)	FD A6 XX
ADC A, C	89	AND A, A	A7
ADC A, D	8A	AND A, B	A0
ADC A, E	8B	AND A, C	A1
ADC A, H	8C	AND A, D	A2
ADC A, L	8D	AND A, E	A3
ADC A, n	CE XX	AND A, H	A4
ADC HL, BC	ED 4A	AND A, L	A5
ADC HL, DE	ED 5A	AND A, n	E6 XX
ADC HL, HL	ED 6A	BIT 0, (HL)	CB 46
ADC HL, SP	ED 7A	BIT 0, (IX+dis)	DD CB XX 46
ADD A, (HL)	86	BIT 0, (IY+dis)	FD CB XX 46
ADD A, (IX+dis)	DD 86 XX	BIT 0, A	CB 47
ADD A, (IY+dis)	FD 86 XX	BIT 0, B	CB 40
ADD A, A	87	BIT 0, C	CB 41
ADD A, B	80	BIT 0, D	CB 42
ADD A, C	81	BIT 0, E	CB 43
ADD A, D	82	BIT 0, H	CB 44
ADD A, E	83	BIT 0, L	CB 45
ADD A, H	84	BIT 1, (HL)	CB 4E
ADD A, L	85	BIT 1, (IX+dis)	DD CB XX 4E
ADD A, n	C6 XX	BIT 1, (IY+dis)	FD CB XX 4E
ADD HL, BC	09	BIT 1, A	CB 4F
ADD HL, DE	19	BIT 1, B	CB 48
ADD HL, HL	29	BIT 1, C	CB 49
ADD HL, SP	39	BIT 1, D	CB 4A
ADD IX, BC	DD 09	BIT 1, E	CB 4B
ADD IX, DE	DD 19	BIT 1, H	CB 4C
ADD IX, IX	DD 29	BIT 1, L	CB 4D
ADD IX, SP	DD 39	BIT 2, (HL)	CB 56
ADD IY, BC	FD 09	BIT 2, (IX+dis)	DD CB XX 56
ADD IY, DE	FD 19	BIT 2, (IY+dis)	FD CB XX 56

BIT 2, A	CB 57
BIT 2, B	CB 50
BIT 2, C	CB 51
BIT 2, D	CB 52
BIT 2, E	CB 53
BIT 2, H	CB 54
BIT 2, L	CB 55
BIT 3, (HL)	CB 5E
BIT 3, (IX+dis)	DD CB XX 5E
BIT 3, (IY+dis)	FD CB XX 5E
BIT 3, A	CB 5F
BIT 3, B	CB 58
BIT 3, C	CB 59
BIT 3, D	CB 5A
BIT 3, E	CB 5B
BIT 3, H	CB 5C
BIT 3, L	CB 5D
BIT 4, (HL)	CB 66
BIT 4, (IX+dis)	DD CB XX 66
BIT 4, (IY+dis)	FD CB XX 66
BIT 4, A	CB 67
BIT 4, B	CB 60
BIT 4, C	CB 61
BIT 4, D	CB 62
BIT 4, E	CB 63
BIT 4, H	CB 64
BIT 4, L	CB 65
BIT 5, (HL)	CB 6E
BIT 5, (IX+dis)	DD CB XX 6E
BIT 5, (IY+dis)	FD CB XX 6E
BIT 5, A	CB 6F
BIT 5, B	CB 68
BIT 5, C	CB 69
BIT 5, D	CB 6A
BIT 5, E	CB 6B
BIT 5, H	CB 6C
BIT 5, L	CB 6D
BIT 6, (HL)	CB 76
BIT 6, (IX+dis)	DD CB XX 76
BIT 6, (IY+dis)	FD CB XX 76
BIT 6, A	CB 77
BIT 6, B	CB 70
BIT 6, C	CB 71
BIT 6, D	CB 72
BIT 6, E	CB 73
BIT 6, H	CB 74
BIT 6, L	CB 75
BIT 7, (HL)	CB 7E
BIT 7, (IX+dis)	DD CB XX 7E
BIT 7, (IY+dis)	FD CB XX 7E
BIT 7, A	CB 7F
BIT 7, B	CB 78
BIT 7, C	CB 79
BIT 7, D	CB 7A
BIT 7, E	CB 7B
BIT 7, H	CB 7C
BIT 7, L	CB 7D
CALL naslov	CD XX XX
CALL C, naslov	DC XX XX
CALL NC, naslov	D4 XX XX
CALL M, naslov	FC XX XX
CALL P, naslov	F4 XX XX
CALL PE, naslov	EC XX XX
CALL PO, naslov	E4 XX XX
CALL Z, naslov	CC XX XX
CALL NZ, naslov	C4 XX XX
CCF	3F
CP (HL)	BE
CP (IX+dis)	DD BE XX
CP (IY+dis)	FD BE XX
CP A	BF
CP B	B8
CP C	B9
CP D	BA
CP E	BB
CP H	BC
CP L	BD
CP n	FE XX
CPD	ED A9
CPDR	ED B9
CPI	ED A1
CPIR	ED B1

CPL	2F
DAA	27
DEC (HL)	35
DEC (IX+dis)	DD 35 XX
DEC (IY+dis)	FD 35 XX
DEC A	3D
DEC B	05
DEC C	0D
DEC D	15
DEC E	1D
DEC H	25
DEC L	2D
DEC BC	0B
DEC DE	1B
DEC HL	2B
DEC IX	DD 2B
DEC IY	FD 2B
DEC SP	3B
DI	F3
DJNZ dis	10 XX
EI	FB
EX (SP), HL	E3
EX (SP), IX	DD E3
EX (SP), IY	FD E3
EX AF, A'F'	08
EX DE, HL	EB
EXX	D9
HALT	76
IM 0	ED 46
IM 1	ED 56
IM 2	ED 5E
IN A, (C)	ED 78
IN A, (vrata)	DB XX
IN B, (C)	ED 40
IN C, (C)	ED 48
IN D, (C)	ED 50
IN E, (C)	ED 58
IN H, (C)	ED 60
IN L, (C)	ED 68
INC (HL)	34
INC (IX+dis)	DD 34 XX
INC (IY+dis)	FD 34 XX
INC A	3C
INC B	04
INC C	0C
INC D	14
INC E	1C
INC H	24
INC L	2C
INC BC	03
INC DE	13
INC HL	23
INC IX	DD 23
INC IY	FD 23
INC SP	33
IND	ED AA
INDR	ED BA
INI	ED A2
INIR	ED B2
JP (HL)	E9
JP (IX)	DD E9
JP (IY)	FD E9
JP naslov	C3 XX XX
JP C, naslov	DA XX XX
JP NC, naslov	D2 XX XX
JP M, naslov	FA XX XX
JP P, naslov	F2 XX XX
JP PE, naslov	EA XX XX
JP PO, naslov	E2 XX XX
JP Z, naslov	CA XX XX
JP NZ, naslov	C2 XX XX
JR dis	18 XX
JR C, dis	38 XX
JR NC, dis	30 XX
JR Z, dis	28 XX
JR NZ, dis	20 XX
LD (naslov), A	32 XX XX
LD (naslov), BC	ED 43 XX XX
LD (naslov), DE	ED 53 XX XX
LD (naslov), HL	22 XX XX
LD (naslov), HL	ED 63 XX XX
LD (naslov), IX	DD 22 XX XX

LD (naslov), IY	FD 22 XX XX
LD (naslov), SP	ED 73 XX XX
LD (BC), A	02
LD (DE), A	12
LD (HL), A	77
LD (HL), B	70
LD (HL), C	71
LD (HL), D	72
LD (HL), E	73
LD (HL), H	74
LD (HL), L	75
LD (HL), n	36 XX
LD (IX+dis), A	DD 77 XX
LD (IX+dis), B	DD 70 XX
LD (IX+dis), C	DD 71 XX
LD (IX+dis), D	DD 72 XX
LD (IX+dis), E	DD 73 XX
LD (IX+dis), H	DD 74 XX
LD (IX+dis), L	DD 75 XX
LD (IX+dis), n	DD 36 XX XX
LD (IY+dis), A	FD 77 XX
LD (IY+dis), B	FD 70 XX
LD (IY+dis), C	FD 71 XX
LD (IY+dis), D	FD 72 XX
LD (IY+dis), E	FD 73 XX
LD (IY+dis), H	FD 74 XX
LD (IY+dis), L	FD 75 XX
LD (IY+dis), n	FD 36 XX XX
LD A, (naslov)	3A XX XX
LD A, (BC)	0A
LD A, (DE)	1A
LD A, (HL)	7E
LD A, (IX+dis)	DD 7E XX
LD A, (IY+dis)	FD 7E XX
LD A, A	7F
LD A, B	78
LD A, C	79
LD A, D	7A
LD A, E	7B
LD A, H	7C
LD A, I	ED 57
LD A, L	7D
LD A, R	ED 5F
LD A, n	3E XX
LD B, (HL)	46
LD B, (IX+dis)	DD 46 XX
LD B, (IY+dis)	FD 46 XX
LD B, A	47
LD B, B	40
LD B, C	41
LD B, D	42
LD B, E	43
LD B, H	44
LD B, L	45
LD B, n	06 XX
LD BC, (naslov)	ED 4B XX XX
LD BC, nn	01 XX XX
LD C, (HL)	4E
LD C, (IX+dis)	DD 4E XX
LD C, (IY+dis)	FD 4E XX
LD C, A	4F
LD C, B	48
LD C, C	49
LD C, D	4A
LD C, E	4B
LD C, H	4C
LD C, L	4D
LD C, n	0E XX
LD D, (HL)	56
LD D, (IX+dis)	DD 56 XX
LD D, (IY+dis)	FD 56 XX
LD D, A	57
LD D, B	50
LD D, C	51
LD D, D	52
LD D, E	53
LD D, H	54
LD D, L	55
LD D, n	16 XX
LD DE, (naslov)	ED 5B XX XX
LD DE, nn	11 XX XX
LD E, (HL)	5E

LD E, (IX+dis)	DD 5E XX
LD E, (IY+dis)	FD 5E XX
LD E, A	5F
LD E, B	58
LD E, C	59
LD E, D	5A
LD E, E	5B
LD E, H	5C
LD E, L	5D
LD E, n	1E XX
LD H, (HL)	66
LD H, (IX+dis)	DD 66 XX
LD H, (IY+dis)	FD 66 XX
LD H, A	67
LD H, B	60
LD H, C	61
LD H, D	62
LD H, E	63
LD H, H	64
LD H, L	65
LD H, n	26 XX
LD HL, (naslov)	2A XX XX
LD HL, (naslov)	ED 6B XX XX
LD HL, nn	21 XX XX
LD I, A	ED 47
LD IX, (naslov)	DD 2A XX XX
LD IX, nn	DD 21 XX XX
LD IY, (naslov)	FD 2A XX XX
LD IY, nn	FD 21 XX XX
LD L, (HL)	6E
LD L, (IX+dis)	DD 6E XX
LD L, (IY+dis)	FD 6E XX
LD L, A	6F
LD L, B	68
LD L, C	69
LD L, D	6A
LD L, E	6B
LD L, H	6C
LD L, L	6D
LD L, n	2E XX
LD R, A	ED 4F
LD SP, (naslov)	ED 7B XX XX
LD SP, nn	31 XX XX
LD SP, HL	F9
LD SP, IX	DD F9
LD SP, IY	FD F9
LDD	ED A8
LDDR	ED B8
LDI	ED A0
LDIR	ED B0
NEG	ED 44
NOP	00
OR (HL)	B6
OR (IX+dis)	DD B6 XX
OR (IY+dis)	FD B6 XX
OR A	B7
OR B	B0
OR C	B1
OR D	B2
OR E	B3
OR H	B4
OR L	B5
OR n	F6 XX
OTDR	ED BB
OTIR	ED B3
OUT (C), A	ED 79
OUT (vrata), A	D3 XX
OUT (C), B	ED 41
OUT (C), C	ED 49
OUT (C), D	ED 51
OUT (C), E	ED 59
OUT (C), H	ED 61
OUT (C), L	ED 69
OUTD	ED AB
OUTI	ED A3
POP AF	F1
POP BC	C1
POP DE	D1
POP HL	E1
POP IX	DD E1
POP IY	FD E1
PUSH AF	F5

PUSH BC	C5
PUSH DE	D5
PUSH HL	E5
PUSH IX	DD E5
PUSH IY	FD E5
RES 0, (HL)	CB 86
RES 0, (IX+dis)	DD CB XX 86
RES 0, (IY+dis)	FD CB XX 86
RES 0, A	CB 87
RES 0, B	CB 80
RES 0, C	CB 81
RES 0, D	CB 82
RES 0, E	CB 83
RES 0, H	CB 84
RES 0, L	CB 85
RES 1, (HL)	CB 8E
RES 1, (IX+dis)	DD CB XX 8E
RES 1, (IY+dis)	FD CB XX 8E
RES 1, A	CB 8F
RES 1, B	CB 88
RES 1, C	CB 89
RES 1, D	CB 8A
RES 1, E	CB 8B
RES 1, H	CB 8C
RES 1, L	CB 8D
RES 2, (HL)	CB 96
RES 2, (IX+dis)	DD CB XX 96
RES 2, (IY+dis)	FD CB XX 96
RES 2, A	CB 97
RES 2, B	CB 90
RES 2, C	CB 91
RES 2, D	CB 92
RES 2, E	CB 93
RES 2, H	CB 94
RES 2, L	CB 95
RES 3, (HL)	CB 9E
RES 3, (IX+dis)	DD CB XX 9E
RES 3, (IY+dis)	FD CB XX 9E
RES 3, A	CB 9F
RES 3, B	CB 98
RES 3, C	CB 99
RES 3, D	CB 9A
RES 3, E	CB 9B
RES 3, H	CB 9C
RES 3, L	CB 9D
RES 4, (HL)	CB A6
RES 4, (IX+dis)	DD CB XX A6
RES 4, (IY+dis)	FD CB XX A6
RES 4, A	CB A7
RES 4, B	CB A0
RES 4, C	CB A1
RES 4, D	CB A2
RES 4, E	CB A3
RES 4, H	CB A4
RES 4, L	CB A5
RES 5, (HL)	CB AE
RES 5, (IX+dis)	DD CB XX AE
RES 5, (IY+dis)	FD CB XX AE
RES 5, A	CB AF
RES 5, B	CB A8
RES 5, C	CB A9
RES 5, D	CB AA
RES 5, E	CB AB
RES 5, H	CB AC
RES 5, L	CB AD
RES 6, (HL)	CB B6
RES 6, (IX+dis)	DD CB XX B6
RES 6, (IY+dis)	FD CB XX B6
RES 6, A	CB B7
RES 6, B	CB B0
RES 6, C	CB B1
RES 6, D	CB B2
RES 6, E	CB B3
RES 6, H	CB B4
RES 6, L	CB B5
RES 7, (HL)	CB BE
RES 7, (IX+dis)	DD CB XX BE
RES 7, (IY+dis)	FD CB XX BE
RES 7, A	CB BF
RES 7, B	CB B8
RES 7, C	CB B9
RES 7, D	CB BA

RES 7, E	CB BB
RES 7, H	CB BC
RES 7, L	CB BD
RET	C9
RET C	D8
RET NC	D0
RET M	F8
RET P	F0
RET PE	E8
RET PO	E0
RET Z	C8
RET NZ	C0
RETI	ED 4D
RETN	ED 45
RL (HL)	CB 16
RL (IX+dis)	DD CB XX 16
RL (IY+dis)	FD CB XX 16
RL A	CB 17
RLA	17
RL B	CB 10
RL C	CB 11
RL D	CB 12
RL E	CB 13
RL H	CB 14
RL L	CB 15
RLC (HL)	CB 06
RLC (IX+dis)	DD CB XX 06
RLC (IY+dis)	FD CB XX 06
RLC A	CB 07
RLCA	07
RLC B	CB 00
RLC C	CB 01
RLC D	CB 02
RLC E	CB 03
RLC H	CB 04
RLC L	CB 05
RLD	ED 6F
RR (HL)	CB 1E
RR (IX+dis)	DD CB XX 1E
RR (IY+dis)	FD CB XX 1E
RR A	CB 1F
RRA	1F
RR B	CB 18
RR C	CB 19
RR D	CB 1A
RR E	CB 1B
RR H	CB 1C
RR L	CB 1D
RRC (HL)	CB 0E
RRC (IX+dis)	DD CB XX 0E
RRC (IY+dis)	FD CB XX 0E
RRC A	CB 0F
RRCA	0F
RRC B	CB 08
RRC C	CB 09
RRC D	CB 0A
RRC E	CB 0B
RRC H	CB 0C
RRC L	CB 0D
RRD	ED 67
RST 00	C7
RST 08	CF
RST 10	D7
RST 18	DF
RST 20	E7
RST 28	EF
RST 30	F7
RST 38	FF
SBC A, (HL)	9E
SBC A, (IX+dis)	DD 9E XX
SBC A, (IY+dis)	FD 9E XX
SBC A, A	9F
SBC A, B	98
SBC A, C	99
SBC A, D	9A
SBC A, E	9B
SBC A, H	9C
SBC A, L	9D
SBC A, n	DE XX
SBC HL, BC	ED 42
SBC HL, DE	ED 52
SBC HL, HL	ED 62

SBC HL, SP	ED 72
SCF	37
SET 0, (HL)	CB C6
SET 0, (IX+dis)	DD CB XX C6
SET 0, (IY+dis)	FD CB XX C6
SET 0, A	CB C7
SET 0, B	CB C0
SET 0, C	CB C1
SET 0, D	CB C2
SET 0, E	CB C3
SET 0, H	CB C4
SET 0, L	CB C5
SET 1, (HL)	CB CE
SET 1, (IX+dis)	DD CB XX CE
SET 1, (IY+dis)	FD CB XX CE
SET 1, A	CB CF
SET 1, B	CB C8
SET 1, C	CB C9
SET 1, D	CB CA
SET 1, E	CB CB
SET 1, H	CB CC
SET 1, L	CB CD
SET 2, (HL)	CB D6
SET 2, (IX+dis)	DD CB XX D6
SET 2, (IY+dis)	FD CB XX D6
SET 2, A	CB D7
SET 2, B	CB D0
SET 2, C	CB D1
SET 2, D	CB D2
SET 2, E	CB D3
SET 2, H	CB D4
SET 2, L	CB D5
SET 3, (HL)	CB DE
SET 3, (IX+dis)	DD CB XX DE
SET 3, (IY+dis)	FD CB XX DE
SET 3, A	CB DF
SET 3, B	CB D8
SET 3, C	CB D9
SET 3, D	CB DA
SET 3, E	CB DB
SET 3, H	CB DC
SET 3, L	CB DD
SET 4, (HL)	CB E6
SET 4, (IX+dis)	DD CB XX E6
SET 4, (IY+dis)	FD CB XX E6
SET 4, A	CB E7
SET 4, B	CB E0
SET 4, C	CB E1
SET 4, D	CB E2
SET 4, E	CB E3
SET 4, H	CB E4
SET 4, L	CB E5
SET 5, (HL)	CB EE
SET 5, (IX+dis)	DD CB XX EE
SET 5, (IY+dis)	FD CB XX EE
SET 5, A	CB EF
SET 5, B	CB E8
SET 5, C	CB E9
SET 5, D	CB EA
SET 5, E	CB EB
SET 5, H	CB EC
SET 5, L	CB ED
SET 6, (HL)	CB F6
SET 6, (IX+dis)	DD CB XX F6
SET 6, (IY+dis)	FD CB XX F6
SET 6, A	CB F7
SET 6, B	CB F0
SET 6, C	CB F1
SET 6, D	CB F2
SET 6, E	CB F3
SET 6, H	CB F4
SET 6, L	CB F5
SET 7, (HL)	CB FE
SET 7, (IX+dis)	DD CB XX FE
SET 7, (IY+dis)	FD CB XX FE
SET 7, A	CB FF
SET 7, B	CB F8
SET 7, C	CB F9
SET 7, D	CB FA
SET 7, E	CB FB
SET 7, H	CB FC
SET 7, L	CB FD

SLA (HL)	CB 26
SLA (IX+dis)	DD CB XX 26
SLA (IY+dis)	FD CB XX 26
SLA A	CB 27
SLA B	CB 20
SLA C	CB 21
SLA D	CB 22
SLA E	CB 23
SLA H	CB 24
SLA L	CB 25
SRA (HL)	CB 2E
SRA (IX+dis)	DD CB XX 2E
SRA (IY+dis)	FD CB XX 2E
SRA A	CB 2F
SRA B	CB 28
SRA C	CB 29
SRA D	CB 2A
SRA E	CB 2B
SRA H	CB 2C
SRA L	CB 2D
SRL (HL)	CB 3E
SRL (IX+dis)	DD CB XX 3E
SRL (IY+dis)	FD CB XX 3E
SRL A	CB 3F
SRL B	CB 38
SRL C	CB 39
SRL D	CB 3A
SRL E	CB 3B
SRL H	CB 3C
SRL L	CB 3D
SUB (HL)	96
SUB (IX+dis)	DD 96 XX
SUB (IY+dis)	FD 96 XX
SUB A	97
SUB B	90
SUB C	91
SUB D	92
SUB E	93
SUB H	94
SUB L	95
SUB n	D6 XX
XOR (HL)	AE
XOR (IX+dis)	DD AE XX
XOR (IY+dis)	FD AE XX
XOR A	AF
XOR B	A8
XOR C	A9
XOR D	AA
XOR E	AB
XOR H	AC
XOR L	AD
XOR n	EE XX

Opomba: Pri sprotnem indeksnem naslavljanju sledi kodu ukaza najprej odmik in nato število.

STROJNI KODI PROCESORJA Z80 IN NJIHOVI UKAZI

00	NOP	24	INC H
01 XX XX	LD BC, nn	25	DEC H
02	LD (BC), A	26 XX	LD H, n
03	INC BC	27	DAA
04	INC B	28 XX	JR Z, dis
05	DEC B	29	ADD HL, HL
06 XX	LD B, n	2A XX XX	LD HL, (naslov)
07	RLCA	2B	DEC HL
08	EX AF, A'F'	2C	INC L
09	ADD HL, BC	2D	DEC L
0A	LD A, (BC)	2E XX	LD L, n
0B	DEC BC	2F	CPL
0C	INC C	30 XX	JR NC, dis
0D	DEC C	31 XX XX	LD SP, nn
0E XX	LD C, n	32 XX XX	LD (naslov), A
0F	RRCA	33	INC SP
10 XX	DJNZ dis	34	INC (HL)
11 XX XX	LD DE, nn	35	DEC (HL)
12	LD (DE), A	36 XX	LD (HL), n
13	INC DE	37	SCF
14	INC D	38 XX	JR C, dis
15	DEC D	39	ADD HL, SP
16 XX	LD D, n	3A XX XX	LD A, (naslov)
17	RLA	3B	DEC SP
18 XX	JR dis	3C	INC A
19	ADD HL, DE	3D	DEC A
1A	LD A, (DE)	3E XX	LD A, n
1B	DEC DE	3F	CCF
1C	INC E	40	LD B, B
1D	DEC E	41	LD B, C
1E XX	LD E, n	42	LD B, D
1F	RRA	43	LD B, E
20 XX	JR NZ, dis	44	LD B, H
21 XX XX	LD HL, nn	45	LD B, L
22 XX XX	LD (naslov), HL	46	LD B, (HL)
23	INC HL	47	LD B, A

48	LD C, B	71	LD (HL), C
49	LD C, C	72	LD (HL), D
4A	LD C, D	73	LD (HL), E
4B	LD C, E	74	LD (HL), H
4C	LD C, H	75	LD (HL), L
4D	LD C, L	76	HALT
4E	LD C, (HL)	77	LD (HL), A
4F	LD C, A	78	LD A, B
50	LD D, B	79	LD A, C
51	LD D, C	7A	LD A, D
52	LD D, D	7B	LD A, E
53	LD D, E	7C	LD A, H
54	LD D, H	7D	LD A, L
55	LD D, L	7E	LD A, (HL)
56	LD D, (HL)	7F	LD A, A
57	LD D, A	80	ADD A, B
58	LD E, B	81	ADD A, C
59	LD E, C	82	ADD A, D
5A	LD E, D	83	ADD A, E
5B	LD E, E	84	ADD A, H
5C	LD E, H	85	ADD A, L
5D	LD E, L	86	ADD A, (HL)
5E	LD E, (HL)	87	ADD A, A
5F	LD E, A	88	ADC A, B
60	LD H, B	89	ADC A, C
61	LD H, C	8A	ADC A, D
62	LD H, D	8B	ADC A, E
63	LD H, E	8C	ADC A, H
64	LD H, H	8D	ADC A, L
65	LD H, L	8E	ADC A, (HL)
66	LD H, (HL)	8F	ADC A, A
67	LD H, A	90	SUB B
68	LD L, B	91	SUB C
69	LD L, C	92	SUB D
6A	LD L, D	93	SUB E
6B	LD L, E	94	SUB H
6C	LD L, H	95	SUB L
6D	LD L, L	96	SUB (HL)
6E	LD L, (HL)	97	SUB A
6F	LD L, A	98	SBC A, B
70	LD (HL), B	99	SBC A, C

9A	SBC A, D	C3 XX XX	JP naslov
9B	SBC A, E	C4 XX XX	CALL NZ, naslov
9C	SBC A, H	C5	PUSH BC
9D	SBC A, L	C6 XX	ADD A, n
9E	SBC A, (HL)	C7	RST 00
9F	SBC A, A	C8	RET Z
A0	AND A, B	C9	RET
A1	AND A, C	CA XX XX	JP Z, naslov
A2	AND A, D	CB 00	RLC B
A3	AND A, E	CB 01	RLC C
A4	AND A, H	CB 02	RLC D
A5	AND A, L	CB 03	RLC E
A6	AND A, (HL)	CB 04	RLC H
A7	AND A, A	CB 05	RLC L
A8	XOR B	CB 06	RLC (HL)
A9	XOR C	CB 07	RLC A
AA	XOR D	CB 08	RRC B
AB	XOR E	CB 09	RRC C
AC	XOR H	CB 0A	RRC D
AD	XOR L	CB 0B	RRC E
AE	XOR (HL)	CB 0C	RRC H
AF	XOR A	CB 0D	RRC L
B0	OR B	CB 0E	RRC (HL)
B1	OR C	CB 0F	RRC A
B2	OR D	CB 10	RL B
B3	OR E	CB 11	RL C
B4	OR H	CB 12	RL D
B5	OR L	CB 13	RL E
B6	OR (HL)	CB 14	RL H
B7	OR A	CB 15	RL L
B8	CP B	CB 16	RL (HL)
B9	CP C	CB 17	RL A
BA	CP D	CB 18	RR B
BB	CP E	CB 19	RR C
BC	CP H'	CB 1A	RR D
BD	CP L	CB 1B	RR E
BE	CP (HL)	CB 1C	RR H
BF	CP A	CB 1D	RR L
C0	RET NZ	CB 1E	RR (HL)
C1	POP BC	CB 1F	RR A
C2 XX XX	JP NZ, naslov	CB 20	SLA B

CB 21	SLA C	CB 52	BIT 2, D
CB 22	SLA D	CB 53	BIT 2, E
CB 23	SLA E	CB 54	BIT 2, H
CB 24	SLA H	CB 55	BIT 2, L
CB 25	SLA L	CB 56	BIT 2, (HL)
CB 26	SLA (HL)	CB 57	BIT 2, A
CB 27	SLA A	CB 58	BIT 3, B
CB 28	SRA B	CB 59	BIT 3, C
CB 29	SRA C	CB 5A	BIT 3, D
CB 2A	SRA D	CB 5B	BIT 3, E
CB 2B	SRA E	CB 5C	BIT 3, H
CB 2C	SRA H	CB 5D	BIT 3, L
CB 2D	SRA L	CB 5E	BIT 3, (HL)
CB 2E	SRA (HL)	CB 5F	BIT 3, A
CB 2F	SRA A	CB 60	BIT 4, B
CB 38	SRL B	CB 61	BIT 4, C
CB 39	SRL C	CB 62	BIT 4, D
CB 3A	SRL D	CB 63	BIT 4, E
CB 3B	SRL E	CB 64	BIT 4, H
CB 3C	SRL H	CB 65	BIT 4, L
CB 3D	SRL L	CB 66	BIT 4, (HL)
CB 3E	SRL (HL)	CB 67	BIT 4, A
CB 3F	SRL A	CB 68	BIT 5, B
CB 40	BIT 0, B	CB 69	BIT 5, C
CB 41	BIT 0, C	CB 6A	BIT 5, D
CB 42	BIT 0, D	CB 6B	BIT 5, E
CB 43	BIT 0, E	CB 6C	BIT 5, H
CB 44	BIT 0, H	CB 6D	BIT 5, L
CB 45	BIT 0, L	CB 6E	BIT 5, (HL)
CB 46	BIT 0, (HL)	CB 6F	BIT 5, A
CB 47	BIT 0, A	CB 70	BIT 6, B
CB 48	BIT 1, B	CB 71	BIT 6, C
CB 49	BIT 1, C	CB 72	BIT 6, D
CB 4A	BIT 1, D	CB 73	BIT 6, E
CB 4B	BIT 1, E	CB 74	BIT 6, H
CB 4C	BIT 1, H	CB 75	BIT 6, L
CB 4D	BIT 1, L	CB 76	BIT 6, (HL)
CB 4E	BIT 1, (HL)	CB 77	BIT 6, A
CB 4F	BIT 1, A	CB 78	BIT 7, B
CB 50	BIT 2, B	CB 79	BIT 7, C
CB 51	BIT 2, C	CB 7A	BIT 7, D

CB 7B	BIT 7, E	CB A4	RES 4, H
CB 7C	BIT 7, H	CB A5	RES 4, L
CB 7D	BIT 7, L	CB A6	RES 4, (HL)
CB 7E	BIT 7, (HL)	CB A7	RES 4, A
CB 7F	BIT 7, A	CB A8	RES 5, B
CB 80	RES 0, B	CB A9	RES 5, C
CB 81	RES 0, C	CB AA	RES 5, D
CB 82	RES 0, D	CB AB	RES 5, E
CB 83	RES 0, E	CB AC	RES 5, H
CB 84	RES 0, H	CB AD	RES 5, L
CB 85	RES 0, L	CB AE	RES 5, (HL)
CB 86	RES 0, (HL)	CB AF	RES 5, A
CB 87	RES 0, A	CB B0	RES 6, B
CB 88	RES 1, B	CB B1	RES 6, C
CB 89	RES 1, C	CB B2	RES 6, D
CB 8A	RES 1, D	CB B3	RES 6, E
CB 8B	RES 1, E	CB B4	RES 6, H
CB 8C	RES 1, H	CB B5	RES 6, L
CB 8D	RES 1, L	CB B6	RES 6, (HL)
CB 8E	RES 1, (HL)	CB B7	RES 6, A
CB 8F	RES 1, A	CB B8	RES 7, B
CB 90	RES 2, B	CB B9	RES 7, C
CB 91	RES 2, C	CB BA	RES 7, D
CB 92	RES 2, D	CB BB	RES 7, E
CB 93	RES 2, E	CB BC	RES 7, H
CB 94	RES 2, H	CB BD	RES 7, L
CB 95	RES 2, L	CB BE	RES 7, (HL)
CB 96	RES 2, (HL)	CB BF	RES 7, A
CB 97	RES 2, A	CB C0	SET 0, B
CB 98	RES 3, B	CB C1	SET 0, C
CB 99	RES 3, C	CB C2	SET 0, D
CB 9A	RES 3, D	CB C3	SET 0, E
CB 9B	RES 3, E	CB C4	SET 0, H
CB 9C	RES 3, H	CB C5	SET 0, L
CB 9D	RES 3, L	CB C6	SET 0, (HL)
CB 9E	RES 3, (HL)	CB C7	SET 0, A
CB 9F	RES 3, A	CB C8	SET 1, B
CB A0	RES 4, B	CB C9	SET 1, C
CB A1	RES 4, C	CB CA	SET 1, D
CB A2	RES 4, D	CB CB	SET 1, E
CB A3	RES 4, E	CB CC	SET 1, H

CB CD	SET 1, L	CB F6	SET 6, (HL)
CB CE	SET 1, (HL)	CB F7	SET 6, A
CB CF	SET 1, A	CB F8	SET 7, B
CB D0	SET 2, B	CB F9	SET 7, C
CB D1	SET 2, C	CB FA	SET 7, D
CB D2	SET 2, D	CB FB	SET 7, E
CB D3	SET 2, E	CB FC	SET 7, H
CB D4	SET 2, H	CB FD	SET 7, L
CB D5	SET 2, L	CB FE	SET 7, (HL)
CB D6	SET 2, (HL)	CB FF	SET 7, A
CB D7	SET 2, A	CC XX XX	CALL Z, naslov
CB D8	SET 3, B	CD XX XX	CALL naslov
CB D9	SET 3, C	CE XX	ADC A, n
CB DA	SET 3, D	CF	RST 08
CB DB	SET 3, E	D0	RET NC
CB DC	SET 3, H	D1	POP DE
CB DD	SET 3, L	D2 XX XX	JP NC, naslov
CB DE	SET 3, (HL)	D3 XX	OUT (vrata), A
CB DF	SET 3, A	D4 XX XX	CALL NC, naslov
CB E0	SET 4, B	D5	PUSH DE
CB E1	SET 4, C	D6 XX	SUB n
CB E2	SET 4, D	D7	RST 10
CB E3	SET 4, E	D8	RET C
CB E4	SET 4, H	D9	EXX
CB E5	SET 4, L	DA XX XX	JP C, naslov
CB E6	SET 4, (HL)	DB XX	IN A, (vrata)
CB E7	SET 4, A	DC XX XX	CALL C, naslov
CB E8	SET 5, B	DD 09	ADD IX, BC
CB E9	SET 5, C	DD 19	ADD IX, DE
CB EA	SET 5, D	DD 21 XX XX	LD IX, nn
CB EB	SET 5, E	DD 22 XX XX	LD (naslov), IX
CB EC	SET 5, H	DD 23	INC IX
CB ED	SET 5, L	DD 29	ADD IX, IX
CB EE	SET 5, (HL)	DD 2A XX XX	LD IX, (naslov)
CB EF	SET 5, A	DD 2B	DEC IX
CB F0	SET 6, B	DD 34 XX	INC (IX+dis)
CB F1	SET 6, C	DD 35 XX	DEC (IX+dis)
CB F2	SET 6, D	DD 36 XX XX	LD (IX+dis), n
CB F3	SET 6, E	DD 39	ADD IX, SP
CB F4	SET 6, H	DD 46 XX	LD B, (IX+dis)
CB F5	SET 6, L	DD 4E XX	LD E, (IX+dis)

DD 56 XX	LD D, (IX+dis)	DD CB XX B6	RES 6, (IX+dis)
DD 5E XX	LD E, (IX+dis)	DD CB XX BE	RES 7, (IX+dis)
DD 66 XX	LD H, (IX+dis)	DD CB XX C6	SET 0, (IX+dis)
DD 6E XX	LD L, (IX+dis)	DD CB XX CE	SET 1, (IX+dis)
DD 70 XX	LD (IX+dis), B	DD CB XX D6	SET 2, (IX+dis)
DD 71 XX	LD (IX+dis), C	DD CB XX DE	SET 3, (IX+dis)
DD 72 XX	LD (IX+dis), D	DD CB XX E6	SET 4, (IX+dis)
DD 73 XX	LD (IX+dis), E	DD CB XX EE	SET 5, (IX+dis)
DD 74 XX	LD (IX+dis), H	DD CB XX F6	SET 6, (IX+dis)
DD 75 XX	LD (IX+dis), L	DD CB XX FE	SET 7, (IX+dis)
DD 77 XX	LD (IX+dis), A	DD E1	POP IX
DD 7E XX	LD A, (IX+dis)	DD E3	EX (SP), IX
DD 86 XX	ADD A, (IX+dis)	DD E5	PUSH IX
DD 8E XX	ADC A, (IX+dis)	DD E9	JP (IX)
DD 96 XX	SUB (IX+dis)	DD F9	LD SP, IX
DD 9E XX	SBC A, (IX+dis)	DE XX	SBC A, n
DD A6 XX	AND A, (IX+dis)	DF	RST 18
DD AE XX	XOR (IX+dis)	E0	RET PO
DD B6 XX	OR (IX+dis)	E1	POP HL
DD BE XX	CP (IX+dis)	E2 XX XX	JP PO, naslov
DD CB XX 06	RLC (IX+dis)	E3	EX (SP), HL
DD CB XX 0E	RRC (IX+dis)	E4 XX XX	CALL PO, naslov
DD CB XX 16	RL (IX+dis)	E5	PUSH HL
DD CB XX 1E	RR (IX+dis)	E6 XX	AND A, n
DD CB XX 26	SLA (IX+dis)	E7	RST 20
DD CB XX 2E	SRA (IX+dis)	E8	RET PE
DD CB XX 3E	SRL (IX+dis)	E9	JP (HL)
DD CB XX 46	BIT 0, (IX+dis)	EA XX XX	JP PE, naslov
DD CB XX 4E	BIT 1, (IX+dis)	EB	EX DE, HL
DD CB XX 56	BIT 2, (IX+dis)	EC XX XX	CALL PE, naslov
DD CB XX 5E	BIT 3, (IX+dis)	ED 40	IN B, (C)
DD CB XX 66	BIT 4, (IX+dis)	ED 41	OUT (C), B
DD CB XX 6E	BIT 5, (IX+dis)	ED 42	SBC HL, BC
DD CB XX 76	BIT 6, (IX+dis)	ED 43 XX XX	LD (naslov), BC
DD CB XX 7E	BIT 7, (IX+dis)	ED 44	NEG
DD CB XX 86	RES 0, (IX+dis)	ED 45	RETN
DD CB XX 8E	RES 1, (IX+dis)	ED 46	IM 0
DD CB XX 96	RES 2, (IX+dis)	ED 47	LD I, A
DD CB XX 9E	RES 3, (IX+dis)	ED 48	IN C, (C)
DD CB XX A6	RES 4, (IX+dis)	ED 49	OUT (C), C
DD CB XX AE	RES 5, (IX+dis)	ED 4A	ADC HL, BC

ED 4B XX XX	LD BC, (naslov)	ED B2	INIR
ED 4D	RETI	ED B3	OTIR
ED 4F	LD R, A	ED B8	LDDR
ED 50	IN D, (C)	ED B9	CPDR
ED 51	OUT (C), D	ED BA	INDR
ED 52	SBC HL, DE	ED BB	OTDR
ED 53 XX XX	LD (naslov), DE	EE XX	XOR n
ED 56	IM 1	EF	RST 28
ED 57	LD A, I	F0	RET P
ED 58	IN E, (C)	F1	POP AF
ED 59	OUT (C), E	F2 XX XX	JP P, naslov
ED 5A	ADC HL, DE	F3	DI
ED 5B XX XX	LD DE, (naslov)	F4 XX XX	CALL P, naslov
ED 5E	IM 2	F5	PUSH AF
ED 5F	LD A, R	F6 XX	OR n
ED 60	IN H, (C)	F7	RST 30
ED 61	OUT (C), H	F8	RET M
ED 62	SBC HL, HL	F9	LD SP, HL
ED 63 XX XX	LD (naslov), HL	FA XX XX	JP M, naslov
ED 67	RRD	FB	EI
ED 68	IN L, (C)	FC XX XX	CALL M, naslov
ED 69	OUT (C), L	FD 09	ADD IY, BC
ED 6A	ADC HL, HL	FD 19	ADD IY, DE
ED 6B XX XX	LD HL, (naslov)	FD 21 XX XX	LD IY, nn
ED 6F	RLD	FD 22 XX XX	LD (naslov), IY
ED 72	SBC HL, SP	FD 23	INC IY
ED 73 XX XX	LD (naslov), SP	FD 29	ADD IY, IY
ED 78	IN A, (C)	FD 2A XX XX	LD IY, (naslov)
ED 79	OUT (C), A	FD 2B	DEC IY
ED 7A	ADC HL, SP	FD 34 XX	INC (IY+dis)
ED 7B XX XX	LD SP, (naslov)	FD 35 XX	DEC (IY+dis)
ED A0	LDI	FD 36 XX XX	LD (IY+dis), n
ED A1	CPI	FD 39	ADD IY, SP
ED A2	INI	FD 46 XX	LD B, (IY+dis)
ED A3	OUTI	FD 4E XX	LD C, (IY+dis)
ED A8	LDD	FD 56 XX	LD D, (IY+dis)
ED A9	CPD	FD 5E XX	LD E, (IY+dis)
ED AA	IND	FD 66 XX	LD H, (IY+dis)
ED AB	OUTD	FD 6E XX	LD L, (IY+dis)
ED B0	LDIR	FD 70 XX	LD (IY+dis), B
ED B1	CPIR	FD 71 XX	LD (IY+dis), C

FD 72 XX	LD (IY+dis), D	FD CB XX 6E	BIT 5, (IY+dis)
FD 73 XX	LD (IY+dis), E	FD CB XX 76	BIT 6, (IY+dis)
FD 74 XX	LD (IY+dis), H	FD CB XX 7E	BIT 7, (IY+dis)
FD 75 XX	LD (IY+dis), L	FD CB XX 86	RES 0, (IY+dis)
FD 77 XX	LD (IY+dis), A	FD CB XX 8E	RES 1, (IY+dis)
FD 7E XX	LD A, (IY+dis)	FD CB XX 96	RES 2, (IY+dis)
FD 86 XX	ADD A, (IY+dis)	FD CB XX 9E	RES 3, (IY+dis)
FD 8E XX	ADC A, (IY+dis)	FD CB XX A6	RES 4, (IY+dis)
FD 96 XX	SUB (IY+dis)	FD CB XX AE	RES 5, (IY+dis)
FD 9E XX	SBC A, (IY+dis)	FD CB XX B6	RES 6, (IY+dis)
FD A6 XX	AND A, (IY+dis)	FD CB XX BE	RES 7, (IY+dis)
FD AE XX	XOR (IY+dis)	FD CB XX C6	SET 0, (IY+dis)
FD B6 XX	OR (IY+dis)	FD CB XX CE	SET 1, (IY+dis)
FD BE XX	CP (IY+dis)	FD CB XX D6	SET 2, (IY+dis)
FD CB XX 06	RLC (IY+dis)	FD CB XX DE	SET 3, (IY+dis)
FD CB XX 0E	RRC (IY+dis)	FD CB XX E6	SET 4, (IY+dis)
FD CB XX 16	RL (IY+dis)	FD CB XX EE	SET 5, (IY+dis)
FD CB XX 1E	RR (IY+dis)	FD CB XX F6	SET 6, (IY+dis)
FD CB XX 26	SLA (IY+dis)	FD CB XX FE	SET 7, (IY+dis)
FD CB XX 2E	SRA (IY+dis)	FD E1	POP IY
FD CB XX 3E	SRL (IY+dis)	FD E3	EX (SP), IY
FD CB XX 46	BIT 0, (IY+dis)	FD E5	PUSH IY
FD CB XX 4E	BIT 1, (IY+dis)	FD E9	JP (IY)
FD CB XX 56	BIT 2, (IY+dis)	FD F9	LD SP, IY
FD CB XX 5E	BIT 3, (IY+dis)	FE XX	CP n
FD CB XX 66	BIT 4, (IY+dis)	FF	RST 38

Opomba: Pri sprotnem indeksnem naslavljanju sledi kodu ukaza najprej odmik in nato število.

STVARNO KAZALO

MONITOR

CONTENTS

Published by:

Picturesque,
6 Corkscrew Hill,
West Wickham,
Kent BR4 9BB.

INTRODUCTION

The SPECTRUM MONITOR is a machine code entry and de-bug monitor utility, with comprehensive facilities including a Disassembler, that has been designed to allow the ZX Spectrum computer to be programmed in machine code, without the need to use any Basic commands.

Used in conjunction with one of the many books available that teach the principles and practice of Machine code programming, the Spectrum MONITOR is an ideal learning tool for the beginner.

For the more experienced user, the Spectrum MONITOR offers the commands necessary for successful Machine code programming and de-bugging.

The MONITOR allows a free interchange with Basic, and has been carefully structured so that machine code routines can be run from either Basic or from the MONITOR, and that the MONITOR can be accessed at any time without upsetting the stack. All keyboard entries are validated, and it is impossible to crash the Spectrum while using the MONITOR commands.

The program cassette that accompanies this book contains two versions of the MONITOR; one for the 16K Spectrum and one for the 48K Spectrum. If you own a 48K machine, you can use either version.

THE SPECTRUM MONITOR occupies just over 4K of memory at the top of memory and is not relocatable. On loading into your Spectrum, Ramtop is automatically reset to below the MONITOR, and should only be altered downwards. In other words, assuming you use the correct version for the memory size your Spectrum contains, you can lower Ramtop further to create space for your own machine code programs if you wish, but the MONITOR uses the 4K of memory immediately below the start of the User Definable Graphics area of RAM. If you load the 16K version into a 48K Spectrum, you can use the memory above the MONITOR but if you move Ramtop above the MONITOR, you may run the risk of corrupting the MONITOR routines with Basic commands.

LOADING THE SPECTRUM MONITOR

Connect your Spectrum to a T.V., cassette recorder, and ZX printer if you own one, as described in the Sinclair manual, and switch on.

The MONITOR loads from cassette in the same way as a Basic program. Set the volume control to about ½ volume and set the tone control to maximum treble.

Type LOAD "Monitor 16" or LOAD "Monitor 48" according to which side of the cassette you are loading, or simply type LOAD " ". Start the cassette in play and press ENTER. As with all commercially recorded cassettes, you may find that you have to experiment until you find the optimum replay level for this tape. Make a note of the volume setting you find successful, for future use. The chances of a bad load are reduced if you observe the following:

1) Regularly clean the record/replay head of your cassette recorder, using one of the proprietory cleaning kits.

2) Clean the rubber pinch wheel and the capstan spindle that it makes contact with.

3) Use a cleaning kit supplied with a liquid cleaner that you apply with a cotton wool swab. This type is far more effective than the cleaning cassette type.

4) Disconnect the 'Mic' lead to the cassette recorder when loading.

If you should experience difficulty in loading your copy of the MONITOR, or if you accidentally damage your tape, please return the cassette to Picturesque (the address is on the front page of this book). Your cassette will be re-recorded directly from our ZX Spectrum, and sent back to you by return of post, along with postage stamps to cover your postage costs. We believe such a back-up service to be an essential part of our trading standards.

ACCESS TO SPECTRUM MONITOR

Having successfully loaded, the MONITOR automatically relocates itself to the correct part of memory, resets Ramtop to below itself, and displays a message to this effect on the screen. This screen message shows the correct form of address to access the MONITOR, depending on which version is loaded.

RANDOMIZE USR 30479 (for 16K)
RANDOMIZE USR 63247 (for 48K)

You can access the MONITOR from Basic in this way at any time, which will produce the following message at the bottom of the screen, in addition to whatever was already there:

Press BREAK for Monitor

On pressing the Break key (shifted or unshifted) the screen is cleared, and the prompt and cursor appear at the bottom of the screen. The MONITOR uses its own internal stack to make its use transparent to your machine code program. These functions will be explained more fully later.

PROMPT & CURSOR

The prompt (▣) indicates that the MONITOR is waiting to be put into a command mode. It does not appear at the start of every line of the display, but only appears when a command routine has ended, and the MONITOR is waiting for a new command instruction. The flashing cursor is visible for the majority of the time, and indicates a request for a keyboard entry, and shows where the result of that keyboard entry will be displayed on the screen.

COMMAND MODES

The range of commands offered by the SPECTRUM MONITOR are as follows:

M Display a memory location and its contents, and change its contents.
X Escape from a command mode to the start of the MONITOR, (Operates on all command modes except R and K.)
A Move an area of RAM to a new location.
F Fill a specified area of RAM with a specified byte value.
I Insert up to 255 bytes into a machine code routine.
D Delete up to 255 bytes from a machine code routine.
J Jump to specified address, and start executing the routine there.
B Set a Breakpoint in a machine code routine, to return control to the MONITOR.
K Restore codes after a Breakpoint has been passed.

R Display the contents of the CPU registers.
C Continue operation of a routine after a Breakpoint.
Y Return to Basic.
P Hex dump to Printer.
$ Text entry.
Z Disassemble any area of memory into Z80 mnemonics either to the screen, or to both the screen and the ZX Printer.
N Number conversion. Hex to Decimal or Decimal to Hex.

All command codes are accessed by a single keystroke denoted by the letter in the left hand column above. All keyboard entries are checked for validity, and at any given time, only the permitted keyboard entries are accepted by the MONITOR; any other key press is rejected and the keyboard is scanned again.

All numeric inputs and displays are in Hex to facilitate the entry of the Z80 Op. Codes. The number conversion routine simplifies access to Hex addresses.

PLEASE NOTE

All references to addresses and their contents in these instructions will be in Hex, and will be shown as a two or four figure number, without the suffix 'H'.

Each command will now be dealt with in detail, with examples, to clarify its operation.

M – Display contents of memory location

The screen should still only show the prompt and cursor in the bottom left corner.

Type M

An inverse M (blue letter on a white square) will appear immediately to the right of the prompt, and the cursor will move along the bottom line by one character space.

Now type in the Hex value of the address that you wish to inspect, say 6000. (You can use the MONITOR to inspect or change any memory location in RAM or ROM, although you cannot alter the ROM values.) The address appears on the screen as you type it in,

and as soon as you have typed the fourth digit, a two character Hex number appears on the screen to the right of the address, showing the Hex value of the contents of that memory location.

```
M6000 00 -
```

The cursor has now moved on, leaving a space after the two contents digits.
Now type FF.

```
M6000 00 FF -
```

This has loaded the Hex value FF into memory location 6000. The value is loaded into the location automatically after you type the second digit.
Now type ENTER.

```
M6000 00 FF
6001 00 -
```

The original line with the prompt has scrolled up one line, and 6001 00 is now displayed on the bottom line. At all times, the MONITOR operates with a scrolling screen, and new information is always displayed on the bottom line.

Typing ENTER displays the next memory location and its contents. To enter a value in this new address, enter the two Hex digits, or type ENTER again for the next memory location.

Now let's check that FF really has been loaded into address 6000.

Type M

```
M6000 00 FF
6001 00
M-
```

The screen has scrolled up one line, and the inverse M has reappeared on the bottom line.

Typing M after a Hex address, data entry, or ENTER, allows you to re-enter the M command routine at the start. To re-enter the M command in the middle of typing a Hex address, type X, to escape, and M to enter the M command.

Now type 6000

```
M6000 00 FF
6001 00
M6000 FF -
```

The contents of 6000 are shown as FF.

To summarize the M command so far, you can sequentially step through memory locations using ENTER and change the value of the contents of each address, or, by typing M again, you can define a new starting address. To re-enter the M command with a partly typed address on the screen, type X to escape to the prompt, and then type M to enter the M command.

Now type 00 to clear the contents of address 6000. You will notice that the cursor is still visible on the bottom line of the screen to the right of the 00 just entered. Although address 6000 now contains the value 00, (the value was changed immediately you typed the second 0) you can change it again if you wish.

So, if you enter an incorrect value, and realise what you have done before you press ENTER, you can correct your mistake without having to specify the address again. In fact, you can make only two attempts at entering a value before the routine returns you to the prompt and cursor, when you will have to type M to re-enter the M command.

Remember, that in the M command, ENTER only has the effect of stepping on to the next memory location, whereas in other commands (where applicable) ENTER causes the operation to be executed.

Imagine that you have just entered a machine code routine from, say 6000 to 6200, and you realise that you need to change the value of one byte somewhere near the middle of the routine, but you do not know the precise address. You could spend a long time guessing addresses and looking at their contents, or repeatedly pressing ENTER until you find the right byte. But if you type M, to re-enter the M command, and look at the contents of an address somewhere near the beginning of the routine, then press ENTER, and hold it pressed, after about 1½ seconds the screen will start scrolling quite fast, and will rapidly display successive locations until you release ENTER. In this way you can quickly scan through a routine until you find the byte you are looking for. To effect the alteration, having released ENTER, you will again have to type M xxxx to re-enter the M command at the correct address, and then change the contents of that address.

The 'M' command, (and the $ command, described on page 18) are the only commands that use the repeating key facility.

X – Escape

The X command allows you to escape from a command mode and returns you to the monitor key scan routine.

Type X

The screen will scroll, and the prompt and cursor will reappear on the bottom line of the screen. You can now enter any of the MONITOR commands. All command routines, with the exception of R (display registers) and K (breakpoint restore), will accept X as an escape command at any time up to the point of execution.

I – Insert

If, having written a machine code routine, you find it necessary to add extra instructions in the middle of that routine, the Insert command (I) allows you to insert up to 255 bytes at any point, and automatically moves up memory a specified area of RAM by the number of bytes you wish to insert.

The Insert command takes the form 'I aaaa bbbb nn' where I is the Insert command mode, aaaa is the Hex address of the first byte of the insertion, bbbb is the Hex address of the highest byte in RAM of the block of memory to be moved, and nn is the number of bytes to be inserted, in Hex.

Example

Type X to restore the prompt and cursor to the bottom line, and then, using the M command, enter the following consecutive values into memory:

```
6000 00
6001 01
6002 02
6003 03
  etc.
   .
   .
   .
   .
```

600A 0A
600B 0B
600C 0C
600D 0D
600E 0E
600F 0F

(The values entered into these locations are purely for a demonstration of the Insert and Delete commands, and, if run, will cause the ZX SPECTRUM to crash).

In the above example, the start of the imaginary routine is 6000 and the end is 600F. We will now insert 5 bytes, the first new byte to be at 6004.

Type X	to restore the prompt and cursor.
Type I	to get into the Insert mode.
Type 6004	the address of the first byte of the insertion.
Type 600F	the address of the highest byte to be moved.
Type 05	the number of bytes to be inserted (Hex).

At any time up to this point, you can type X to escape from this command mode, as no change to RAM has occurred yet. Indeed, if you make a typing error at any time, you must type X, and start the command again.

If you have entered the Insert example correctly, you can now type ENTER to effect the insertion. The screen will scroll up one line, and the prompt and cursor will return to the bottom line. The insertion has been completed.

Using the M command, check through the 21 locations from 6000. Addresses 6004 to 6008 inclusive will now contain the value 00, and 6009 to 6014 will contain the values 04 through to 0F.

When using the Insert command, any absolute addresses in the remainder of the routine that referred to the area that has been relocated, will have to be changed to maintain correct operation of the routine.

D – Delete

This command has the opposite effect to Insert, and takes the form 'D aaaa bbbb nn', where D is the Delete command mode, aaaa is the address of the first byte to be deleted, bbbb is the address of the highest byte to be moved down RAM, and nn is the number of bytes to be deleted.

Assuming that the result of the Insert example is still in memory, let us now move the area of RAM from 6009 to 6014 back to its original place.

Type X	to restore the prompt and cursor.
Type D	to enter the Delete command.
Type 6004	the start of the area to be deleted.
Type 6014	the end of the area to be moved.
Type 05	the number of bytes to be deleted (Hex).
Type ENTER	to effect the deletion.

The prompt and cursor will reappear on the bottom line, and the deletion will be complete.

Now check through addresses 6000 to 6014, using the M command. The contents of these locations will be as they were before the Insert example, and locations 6010 to 6014 will have been loaded with the value 00.

Again, any absolute addresses relating to the area of RAM that has been moved by Delete, will now need to be changed.

A – Area Relocate

The 'A' command block moves a specified area of RAM, and takes the form: 'A aaaa bbbb cccc' where A is the Area Relocate command, aaaa is the present start address, and bbbb is the present end address of the area to be moved, and cccc is the new starting address.

Assuming that the example used for the I and D commands is still in memory, let us now move the whole area from 6000 to 600F up memory, to start at 6200.

Type X	to restore the prompt and cursor.
Type A	to enter the Area Relocate mode.
Type 6000	the start address of the area to be moved.
Type 600F	the end address of the area to be moved.
Type 6200	the new start address.
Type ENTER	to effect the move.

The screen scrolls up one line, and the prompt and cursor return to the bottom line. The move is complete.

Using the M command, check that addresses 6200 to 620F have been loaded with the same values as those still remaining in addresses 6000 to 600F.

This routine will allow you to move up or down memory, from any original starting address to any new starting address, even if the new area overlaps the original area. The original area (unless over-written by the move) is not changed.

Try moving the area from 6000 to 600F to a new start address of 6008 and then move it back again.

A word of warning: Most MONITOR commands that allow you to alter the values in memory locations, will operate on the area of RAM containing the MONITOR routines. Always check carefully that you are not about to corrupt the MONITOR, which occupies the area from EEAC to FEF9, (48K); or 6EAC to 7EF9 (16K).

F — Fill an area with a given value

The Fill routine allows you to enter the same byte value into a given area of RAM, and takes the form: 'F aaaa bbbb xx', when aaaa and bbbb are the start and end addresses respectively of the specified area, and xx is the value to be entered.

Type X	to restore the prompt and cursor.
Type F	to enter the F command.
Type 6020	the start address.
Type 6100	the end address.
Type AA	the value to be entered. (Hex).
Type ENTER	to effect the Fill.

The screen scrolls, and the prompt and cursor appear on the bottom line of the screen. The fill is complete.

Now use the M command, with ENTER kept pressed, to verify that each byte from 6020 to 6100 inclusive has a value of AA.

Y — Return

Printed on the 'Y' key is the keyword RETURN, and by pressing this key, followed by ENTER, when the prompt and cursor alone are visible on the bottom line of the screen, a return to Basic is effected so that you can use any of the Basic commands. As the MONITOR does not have its own Save and Load commands, you will need the Return command to use the Basic Save and Load.

If, having returned to Basic, you wish to access the MONITOR again the following addresses should be used with the USR function, depending upon which version of the MONITOR you are using:

16K version:	type RANDOMIZE USR 30479
48K version:	type RANDOMIZE USR 63247

followed by ENTER.

Using the MONITOR 'Return' command, will reset the stack, but will not affect the Basic listing or the variables.

To demonstrate the remaining commands of the MONITOR, use the 'M' command to enter the following short program, starting at address 6000 Hex.

```
6000  01 00 00      LD   BC,0000 ;  Clear BC
6003  11 00 00      LD   DE,0000 ;  Clear DE
6006  21 00 00      LD   HL,0000 ;  Clear HL
6009  03            INC  BC      ;  BC = BC + 01
600A  13            INC  DE      ;  DE = DE + 01
600B  23            INC  HL      ;  HL = HL + 01
600C  C9            RET          ;  Return

Start 6000
End   600C
```

Having entered Hex codes, go back to 6000 and check that the codes are correct. (Type M 6000 and check the contents of each location).

It is recommended that you would normally Save a machine code program before running it in case it crashes, which it is certainly likely to do unless you are an experienced machine code programmer. In this case, there is no real point in Saving the program, but if you wish to do so, refer to the section on "The Monitor in practice".

The last line of the routine is a return instruction which it is usual to use at the end of a machine code routine, to return you to Basic.

When you are satisfied that you have entered the above program correctly, type 'X' to restore the prompt and cursor.

B — Breakpoint

This command allows you to temporarily interrupt a machine code program at any point, and return control to the MONITOR, so that you can inspect the values in the CPU registers, and in RAM, and make corrections as necessary.

It takes the form 'B aaaa', where B is the Breakpoint command mode, and aaaa is the address of the instruction that the break will replace. (aaaa must be the address of the first byte of a multi-byte instruction).

The Op. codes in the three addresses aaaa, aaaa+1 and aaaa+2 are automatically saved in data bytes within the MONITOR, and these locations are then loaded with CD OF F7 (for the 48K version) or CD OF 77 (for the 16K version), which constitutes a CALL to the entry point of the MONITOR. It must be a CALL to maintain correct operation of the Stack.

On entering the MONITOR at this address, the values in the CPU registers are stored in data bytes within the MONITOR at the addresses shown in Appendix A; the Stack Pointer is set to the MONITOR Stack; the message "Press Break for Monitor" is displayed on the bottom line of the screen, in addition to the screen display your program has created. The MONITOR will now wait until you press 'Break' when it will display the prompt and cursor on the bottom line of the screen.

You will now be able to use any of the MONITOR commands to check or alter the routine, before returning control to the routine at the point at which the break occurred. As the MONITOR uses its own integral Stack, separate from the Program Stack, there is no danger of over-writing the Program Stack during a Breakpoint.

Before running the example just entered, enter a Breakpoint at address 6009. This will have the effect of stopping the program after the registers BC, DE and HL have been cleared, but before they are incremented.

If the prompt is not visible on the bottom line, type X, otherwise,

Type B	the breakpoint command mode
Type 6009	the breakpoint address

There is no need to type ENTER, as the Breakpoint is set after typing the fourth digit. The screen will scroll, and the prompt will appear on the bottom line.

Using the 'M' command, check that 6009 to 600B now contain CD OF F7 (for 48K) or CD OF 77 (for 16K) in place of 03 13 23, the latter having been stored for later replacement.

You are now in a position to run the routine up to the breakpoint.

J — Jump and execute

The Jump command allows you to jump out of the control of the MONITOR to the starting address of any routine that you write, and it takes the form 'J aaaa' where J is the Jump command mode, and aaaa is the start address of your program.

You can run your machine code programs either with the MONITOR 'J' command, or by returning to Basic and using the USR function. Either way, the MONITOR commands are available to you after a Breakpoint.

In this example, we will use the 'J' command.

Type X	to restore the prompt and cursor.
Type J	to enter the Jump command.
Type 6000	the start address.
Type ENTER	

The screen is cleared and the routine will run, and then return to the MONITOR, with the "Press Break for Monitor" message.

The sequence of events on executing the J command is:

i) the screen is cleared.
ii) the Stack Pointer is set to the program Stack.
iii) the start address is put into the Program Counter, and the program is executed.

The MONITOR uses its own integral Stack, which is set on entry to the monitor routine, therefore the program Stack, which is set by

the Basic initialisation routines when the ZX Spectrum is switched on, must be reset before your program can be run. Item (ii) above does this for you. The use of two stacks helps make the MONITOR invisible to your machine code program.

Having run, the program will have encountered the Breakpoint at address 6009, and have displayed "Press BREAK for Monitor". Press the BREAK/SPACE key (shifted or unshifted) to access the MONITOR.

The first operation after a Breakpoint should always be to replace the correct byte values to the addresses where the break occurred.

K – Break Restore

This command restores the correct values into the three bytes overwritten by the Breakpoint command.

Type K

The screen will show K 6009 and will scroll up one line, displaying the prompt on the bottom line. There is no need to type ENTER. Using the M command, verify that the original codes have been replaced in addresses 6009 to 600B.

Only one Breakpoint can be entered at any time, as there is only enough room to store one address and three data bytes, so a Break Restore (K) command must be executed before the next Breakpoint (B) is defined, and it is recommended that a Break Restore (K) command is keyed immediately after a Breakpoint has been encountered.

If you enter an incorrect breakpoint with the B command, type K immediately afterwards to restore the original values to the incorrect breakpoint address, and then re-type the Breakpoint.

The K command can only restore the last entered Breakpoint.

Let us now inspect the CPU registers, to make sure that the program is working as we expect.

R – Display values in CPU registers

If the prompt is not visible on the bottom line of the screen, type X, otherwise

Type R

The screen scrolls up, automatically displaying the CPU register contents thus:

```
>
IR      3F5A

A'F'    0044
B'C'    1721
D'E'    369B
H'L'    2758

AF      0F54
BC      0000
DE      0000
HL      0000

IX      03D4
IY      5C3A
SP      6E93
PC      6009
```

There is no need to type ENTER.

As you will see, the Program Counter contains 6009, the address at which the Breakpoint occurred. The BC, DE and HL register pairs will all contain 0000. In this example, these are the only registers that we are interested in.

The CPU registers are displayed with their contents shown in Hex, and the values in the registers are stored in RAM at the addresses shown in Appendix A.

When the MONITOR is entered from a Breakpoint, the values of the registers immediately prior to the Breakpoint are stored in these memory locations, so that the operation of your routine can be checked, and corrections to the routine, or the register contents can be made before continuing.

Any changes to the CPU register values stored in RAM during a Breakpoint only take effect after a Jump (J) or Breakpoint Continue (C) command has been executed.

Having a) encountered one Breakpoint, b) restored the correct values after the break, and c) verified that the CPU registers have their correct values, we will now enter another Breakpoint, and continue the routine.

If the prompt is not visible on the bottom line of the screen, type X, otherwise,

Type B 600B

This will set a new Breakpoint after the BC and DE register pair have been incremented, but before the HL pair is incremented.

C — Breakpoint Continue

This command allows you to continue from a Breakpoint, and is executed by typing C followed by ENTER. You can Escape to the monitor by typing X before ENTER. The program being run will continue as if the Breakpoint had never occurred.

The screen is cleared; the program Stack is reset; and the CPU registers are re-loaded from their data block before the Breakpoint address is put into the Program Counter, and execution is resumed.

Type C
Type ENTER

The routine will run on until it reaches the next Breakpoint, and display "Press BREAK for Monitor".

When the prompt appears, after pressing 'BREAK',

Type K	to restore the bytes occupied by the Breakpoint.
Type R	to display the registers.

You can now verify that the Program Counter contains 600B, the BC and DE register pair contain 0001, having been incremented, and the HL pair still contains 0000.

When a routine encounters a Breakpoint, it returns control to the MONITOR with a CALL operation, the return address being stored on the Program Stack, for use by the Breakpoint Continue (C) command. Having encountered a Breakpoint and studied the CPU registers and/or memory locations, one of two situations will occur:

1) Everything will be as you expect, and the program is correct to that point. In this case, you would normally restore the Breakpoint bytes ('K' command) and use the Breakpoint Continue ('C' command) to continue the program to a new Breakpoint.

or

2) An error will become evident, in which case you would track down the error and correct it, and then, leaving the current Breakpoint set, use the 'J' command to re-run the program up to the same Breakpoint, to check that your correction is successful.

The Program Stack operation of the MONITOR allows you to do this providing that, at the Breakpoint, there have been an equal number of PUSHes and POPs, or CALLs and RETs. If the Program Stack is not balanced at the Breakpoint, you will have a cumulative stack imbalance every time you use the 'J' command after a Breakpoint (but not if you use the 'C' command). In this case to restore the Stack to normal once you have traced an error, RETURN to Basic ('Y' command) and re-access the monitor from the beginning, then use the 'J' command to run your program up to the Breakpoint again.

Having set a Breakpoint in a routine, you can either use the 'J' command to run the routine, or you can use the RETURN command to go back to Basic, and run the machine code via the USR function. For example, if you have written some machine code as part of a Basic program, which is accessed by the USR function in the Basic program, you can set a Breakpoint using the MONITOR then return to Basic and run the Basic program. When the Breakpoint is reached in the machine code, the MONITOR will be accessed as shown above, and the Breakpoint Continue command will allow the machine code to resume, and eventually return to the Basic program that called it.

The MONITOR has been carefully designed to allow this free interchange between Basic and machine code, without upsetting the Stack.

P — Printer

This command allows you to produce a Hex dump of any section of memory onto the Sinclair Printer. It takes the form: 'P aaaa bbbb' where P is the Printer command, aaaa is the Hex address of the first byte to be printed, and bbbb is the Hex address of the last byte you want printed. The Printer produces a display in the format shown below.

```
0000 F3 AF 11 FF FF C3 CB 11
0008 2A 5D 5C 22 5F 5C 18 43
0010 C3 F2 15 FF FF FF FF FF
```

Each line shows the Hex contents of eight successive locations, with the Hex address of the first byte shown in the left hand column. The routine will only print complete lines, and if the end address that you specify is part of the way along a line, it will print up to the end of that line.

If the prompt is not visible on the bottom line of the screen, type X, otherwise,

Type P	to enter the Printer command.
Type aaaa	the start address.
Type bbbb	the end address.

Typing X at any point will return you to the prompt and cursor.

Type ENTER.

IMPORTANT NOTE

It is possible to stop the Printer before the routine has been completed, by pressing the Break key (shifted only) but this will return you to the Basic monitor, and you will have to re-access the MONITOR as described on Page 2.

$ — String Entry

This command operates in a similar fashion to the 'M' command, but allows you to enter text directly from the keyboard. It is by no means a word processor, but it offers a much simpler method of text entry than by converting letters to their character codes, and entering the codes individually with the 'M' command.

The command takes the form: '$ aaaa' where $ is the command mode, and aaaa is the starting address of the text block.

Let us enter a simple message into a free area of RAM.

Type $ (Symbol Shift and '4') to enter the $ command.

An inverse $ will appear on the bottom line of the display.

Type 6100

The address is displayed as normal.

Up to the point of entering the last digit of the address, the X command will return you to the prompt. But from now on, this command is slightly different from all the others. As you may well want to type 'X' as a string entry, it cannot now be reserved for the escape command. So in this case only, the escape function is accessed by typing STOP (Symbol Shift and 'A'). The word STOP should serve to remind you of which key to press.

As you type in each letter of the message, it is displayed on the screen, to the right of the address and its present contents, and the character code is stored in that address. The screen scrolls automatically, displaying the next address and its present contents. You do not need to press ENTER to access the next address.

If there is a valid character code in an address, it will be displayed between the address itself and the cursor, otherwise a question mark is displayed. All upper and lower case letters can be entered by use of the Caps Shift key; also punctuation marks and spaces by the use of the Symbol Shift key. The only exception is '$' which is reserved as the command mode.

Graphics, user defined characters, and keywords and expressions such as ** cannot be entered directly. Inverse characters must be created by accessing the colour attributes part of the memory relating to the particular screen location. In other words, any single character that appears on a key top and that can be accessed by a single key press or by one level of shift can be entered. Any character normally accessed by the use of the GRAPHICS or EXTENDED modes will have to be entered by using the 'M' command to enter the Hex character code.

Now type in the following message:

This is "Spectrum MONITOR".

(Use Symbol Shift and P for the " marks). Having typed it in, (mistakes included) now review the message:

Type $	(Symbol Shift and 4) to re-enter the $ command.
Type 6100	the start address.
Type ENTER	and hold it pressed until the whole message is on the screen.

If the message is correct, you can now type STOP (Symbol Shift and A) to return to the prompt.

If you have made a typing error, or if you want to put another message at a new starting address, type $, to re-enter the $ command at the beginning, in the same way that the 'M' command is re-entered. You will have to enter the new address before making your correction, or starting your new message.

The repeating keyboard with the fast scrolling screen works as in the 'M' command, to allow you to review a message quickly.

Remember that, having entered the starting address, the escape command is accessed by typing STOP (Symbol Shift and A).

Z – Disassembler

This command will disassemble any part of RAM or ROM, either to the screen alone, or to both the screen and the ZX Printer. It provides a display that includes the Hex address of the first byte of the instruction, the Hex values of the bytes that relate to that instruction and the Z80 mnemonic for that instruction. The full set of Z80 mnemonics can be disassembled.

The command takes the form: 'Z aaaa bbbb' where 'Z' is the command mode, 'aaaa' is the hex starting address and 'bbbb' is the hex end address of the part of memory you wish to disassemble.

Type Z	to access the command.
Type 0000	the address of the start of the ROM.
Type 0020	the end address.

Having typed in the end address, the screen will scroll and display.

PRINTER?

Your response to this is similar to the Basic "Scroll?" command. If you wish to use the printer, type ENTER; but if you only require a screen display, type 'N' (for NO).

Type N	for screen display only.

The disassembly will appear on the screen thus:

```
>Z0000 0020
PRINTER?
0000 F3       DI
0001 AF       XOR  A
0002 11FFFF   LD   DE,FFFF
```



```
0005 C3CB11   JP   11CB
0008 2A5D5C   LD   HL,(5C5D)
000B 225F5C   LD   (5C5F),HL
000E 1843     JR   0053
0010 C3F215   JP   15F2
0013 FF       RST  38
0014 FF       RST  38
0015 FF       RST  38
0016 FF       RST  38
0017 FF       RST  38
0018 2A5D5C   LD   HL,(5C5D)
001B 7E       LD   A,(HL)
001C CD7D00   CALL 007D
  ENTER for more; X for end
```

16 lines of disassembly will be displayed when using the screen only, followed by the message:

ENTER for more; X for end.

Pressing ENTER will display the next 16 lines, unless the end address is reached, when the prompt and cursor will be returned.

Typing 'X' in response to the above message will also return the prompt and cursor.

If you are disassembling to the Printer, the routine will continue, uninterrupted, until it reaches the end address. The Printer can be stopped by using the Break key in the normal way, which will return you to BASIC. You will then need to access the Monitor as described on Page 2.

If you try to disassemble to the Printer when it is not connected, the screen display will be produced on its own, the routine stopping when it has reached the end address.

All disassembled addresses and values are in Hex. Relative jumps show the address to which the jump will go, with the offset value shown with the hex coding for that instruction.

The only instructions that are displayed in a slightly different form from the published Zilog mnemonics are "JP (HL)" and the IX and IY counterparts "JP (IX)" and "JP (IY)". As these instructions jump to the address actually held in the register, the brackets are not shown in the display, which makes the action of the instruction a little clearer.

N – Number Conversion

This routine will convert Hex numbers to Decimal or vice versa.

Type N The Number command.

The screen will display:

NUMBER H/D?

Type H (for Hex) or D (for Decimal) to indicate the number system of the number you wish to convert.

Type H to convert a Hex number to Decimal.

The screen will scroll, and show 'H' (followed by the cursor. Now enter four Hex digits. (You must enter leading zeros when entering a Hex number).

Type 4000
Type ENTER

The display will now show:

H 4000 = 16384

with the prompt and cursor on the bottom line.

To convert from Decimal to Hex, Type D instead of H in response to "NUMBER H/D?", and enter your decimal number, without leading zeros. Again type ENTER to produce an answer.

The MONITOR in Practice

The purpose of this section is to explain the operations of the MONITOR that affect all commands, and which have not been covered so far, and to give some general precautions when using machine code.

1) The MONITOR display produces white characters on a blue background. After lengthy tests with various colour combinations, this gave the most readable display.

2) The loudspeaker will emit a short Beep when a key is pressed. The length of the Beep has been adjusted to give an easily audible sound, without slowing down the response time of the keyboard. The System Variable PIP does not affect the MONITOR'S keyboard Beep.

3) As has been explained earlier, the MONITOR uses its own internal Stack except when a program is running. The program Stack is reset from "SP" in the data block shown in Appendix A whenever a 'J' (Jump) or 'C' (Break Continue) command is executed. When the 'Y' (Return) command is used, the program Stack is cleared and reset to its normal Basic starting point as defined in the System Variable ERR SP.

4) The CPU register values stored in the addresses shown in Appendix A are only reloaded into the CPU when a Jump (J) or Break Continue (C) command is executed. Returning to Basic (to access your machine code via the USR function) resets the CPU register values as defined by the Basic ROM routines.

5) In addition to the precautions shown on Page 180 of the Sinclair manual advising you not to use the I or IY registers in machine code programs, it is recommended that, if you need to use the alternative BC, DE and HL register pairs and wish to return to Basic after your machine code program, you should save the values held in the alternative registers at the start of your program, and reload them before returning to Basic.

6) The Spectrum MONITOR does not have its own Save and Load routines because the Basic Save and Load routines in the Spectrum allow you to record machine code programs onto cassette. Having written your machine code, you would use the Number Conversion (N) command to convert your Hex start and end addresses into Decimal, calculate the length of your program, and use the Return (Y) command to return to Basic to Save and Verify your machine code.

CONCLUSION

All the commands of the MONITOR have now been demonstrated, and you are ready to start writing and running your own machine code programs.

The Spectrum MONITOR can be used on its own to enter Machine code programs, but to simplify the process of Machine code programming, PICTURESQUE also market an EDITOR ASSEMBLER that is fully compatible with the MONITOR, and which allows you to enter Z80 mnemonics into a listing, with line numbers and Labels. The listing is totally independent from Basic and uses a unique 40 column Screen display.

If you own a 48K Spectrum, the ASSEMBLER and the MONITOR can both reside in memory together, creating the most versatile Machine code system available for the ZX Spectrum. The ASSEMBLER is also designed to make the best use of available memory in a 16K Spectrum.

APPENDIX A — CPU REGISTERS

The values in the CPU registers are stored in the following locations after a Breakpoint, and can be altered using the M command. The alterations only take effect if the Breakpoint Continue (C) command or a Jump (J) command is used to access the machine code.

16K	48K	REG
7F3D –	FF3D –	R
7F3E –	FF3E –	I
7F3F –	FF3F –	F'
7F40 –	FF40 –	A'
7F41 –	FF41 –	C'
7F42 –	FF42 –	B'
7F43 –	FF43 –	E'
7F44 –	FF44 –	D'
7F45 –	FF45 –	L'
7F46 –	FF46 –	H'
7F47 –	FF47 –	F
7F48 –	FF48 –	A
7F49 –	FF49 –	C
7F4A –	FF4A –	B
7F4B –	FF4B –	E
7F4C –	FF4C –	D
7F4D –	FF4D –	L
7F4E –	FF4E –	H
7F4F –	FF4F –	IX (Low)
7F50 –	FF50 –	IX (High)
7F51 –	FF51 –	IY (Low)
7F52 –	FF52 –	IY (High)
7F53 –	FF53 –	SP (Low)
7F54 –	FF54 –	SP (High)
7F55 –	FF55 –	PC (Low)
7F56 –	FF56 –	PC (High)

Any changes to the I and IY registers may cause the ZX SPECTRUM to crash.

APPENDIX B SUMMARY OF COMMANDS

M aaaa nn ENTER	Memory Location & contents in Hex. aaaa = address. nn = new contents value ENTER = next location (repeating) 'M' re-enters command.
X	Escape to prompt, and wait for new command.
I aaaa bbbb nn	Insert. aaaa = address 1st. byte insertion bbbb = address highest byte to be moved nn = no. bytes to be inserted Type ENTER to execute.
D aaaa bbbb nn	Delete. aaaa = address 1st byte deletion bbbb = address highest byte to be moved nn = no. bytes to be deleted Type ENTER to execute
A aaaa bbbb cccc	Area Relocate. aaaa = present start address bbbb = present end address cccc = new start address Type ENTER to execute.
F aaaa bbbb xx	Fill. aaaa = start address of area to fill bbbb = end address of area to fill xx = value to be loaded into area Type ENTER to execute.
Z aaaa bbbb ENTER/N	Disassembler. aaaa = addr. start of disassembly bbbb = addr. end of disassembly Type ENTER for printer or N for screen. BREAK stops printer early.
Y	Return. Returns to Basic and resets the Basic stack Pointer. Type ENTER to execute.

J aaaa	Jump. aaaa = start address of program Type ENTER to execute.
B aaaa	Breakpoint. aaaa = address of Breakpoint Executes automatically on typing 4th address digit.
K	Break Restore.' Executes automatically, and restores last entered'Breakpoint.
R	Register Display. Executes automatically, displaying values in CPU registers.
C	Breakpoint Continue. Type ENTER to execute. Continues program execution after a Breakpoint.
$ aaaa letter/ENTER	String Entry. aaaa = address 1st byte of string Letter = character from keyboard ENTER = next location (repeating) '$' re-enters command. Typing a letter automatically increments address.
P aaaa bbbb	Printer. aaaa = address 1st byte to LPrint bbbb = address last byte required to LPrint Type ENTER to execute Type Break to stop early.
N H/D, Number	Number Conversion. H/D = Hex or Decimal number Type ENTER to execute.

Z80 INSTRUCTION CODES

OBJ CODE	SOURCE STATEMENT	
8E	ADC	A,(HL)
DD8E05	ADC	A,(IX+d)
FD8E05	ADC	A,(IY+d)
8F	ADC	A,A
88	ADC	A,B
89	ADC	A,C
8A	ADC	A,D
8B	ADC	A,E
8C	ADC	A,H
8D	ADC	A,L
CE20	ADC	A,n
ED4A	ADC	HL,BC
ED5A	ADC	HL,DE
ED6A	ADC	HL,HL
ED7A	ADC	HL,SP
86	ADD	A,(HL)
DD8605	ADD	A,(IX+d)
FD8605	ADD	A,(IY+d)
87	ADD	A,A
80	ADD	A,B
81	ADD	A,C
82	ADD	A,D
83	ADD	A,E
84	ADD	A,H
85	ADD	A,L
C620	ADD	A,n
09	ADD	HL,BC
19	ADD	HL,DE
29	ADD	HL,HL
39	ADD	HL,SP
DD09	ADD	IX,BC
DD19	ADD	IX,DE
DD29	ADD	IX,IX
DD39	ADD	IX,SP
FD09	ADD	IY,BC
FD19	ADD	IY,DE
FD29	ADD	IY,IY
FD39	ADD	IY,SP
A6	AND	(HL)
DDA605	AND	(IX+d)
FDA605	AND	(IY+d)
A7	AND	A
A0	AND	B
A1	AND	C
A2	AND	D
A3	AND	E
A4	AND	H
A5	AND	L

OBJ CODE	SOURCE STATEMENT	
E620	AND	n
CB46	BIT	0,(HL)
DDCB0546	BIT	0,(IX+d)
FDCB0546	BIT	0,(IY+d)
CB47	BIT	0,A
CB40	BIT	0,B
CB41	BIT	0,C
CB42	BIT	0,D
CB43	BIT	0,E
CB44	BIT	0,H
CB45	BIT	0,L
CB4E	BIT	1 (HL)
DDCB054E	BIT	1,(IX+d)
FDCB054E	BIT	1,(IY+d)
CB4F	BIT	1,A
CB48	BIT	1,B
CB49	BIT	1,C
CB4A	BIT	1,D
CB4B	BIT	1,E
CB4C	BIT	1,H
CB4D	BIT	1,L
CB56	BIT	2,(HL)
DDCB0556	BIT	2,(IX+d)
FDCB0556	BIT	2,(IY+d)
CB57	BIT	2,A
CB50	BIT	2,B
CB51	BIT	2,C
CB52	BIT	2,D
CB53	BIT	2,E
CB54	BIT	2,H
CB55	BIT	2,L
CB5E	BIT	3,(HL)
DDCB055E	BIT	3,(IX+d)
FDCB055E	BIT	3,(IY+d)
CB5F	BIT	3,A
CB58	BIT	3,B
CB59	BIT	3,C
CB5A	BIT	3,D
CB5B	BIT	3,E
CB5C	BIT	3,H
CB5D	BIT	3,L
CB66	BIT	4,(HL)
DDCB0566	BIT	4,(IX+d)
FDCB0566	BIT	4,(IY+d)
CB67	BIT	4,A
CB60	BIT	4,B
CB61	BIT	4,C
CB62	BIT	4,D

OBJ CODE	SOURCE STATEMENT	
CB63	BIT	4,E
CB64	BIT	4,H
CB65	BIT	4,L
CB6E	BIT	5,(HL)
DDCB056E	BIT	5,(IX+d)
FDCB056E	BIT	5,(IY+d)
CB6F	BIT	5,A
CB68	BIT	5,B
CB69	BIT	5,C
CB6A	BIT	5,D
CB6B	BIT	5,E
CB6C	BIT	5,H
CB6D	BIT	5,L
CB76	BIT	6,(HL)
DDCB0576	BIT	6,(IX+d)
FDCB0576	BIT	6,(IY+d)
CB77	BIT	6,A
CB70	BIT	6,B
CB71	BIT	6,C
CB72	BIT	6,D
CB73	BIT	6,E
CB74	BIT	6,H
CB75	BIT	6,L
CB7E	BIT	7,(HL)
DDCB057E	BIT	7,(IX+d)
FDCB057E	BIT	7,(IY+d)
CB7F	BIT	7,A
CB78	BIT	7,B
CB79	BIT	7,C
CB7A	BIT	7,D
CB7B	BIT	7,E
CB7C	BIT	7,H
CB7D	BIT	7,L
DC8405	CALL	C,nn
FC8405	CALL	M,nn
D48405	CALL	NC,nn
C48405	CALL	NZ,nn
F48405	CALL	P,nn
EC8405	CALL	PE,nn
E48405	CALL	PO,nn
CC8405	CALL	Z,nn
CD8405	CALL	nn
3F	CCF	
BE	CP	(HL)
DDBE05	CP	(IX+d)
FDBE05	CP	(IY+d)
BF	CP	A
B8	CP	B
B9	CP	C
BA	CP	D
BB	CP	E
BC	CP	H
BD	CP	L
FE20	CP	n

OBJ CODE	SOURCE STATEMENT	
EDB1	CPIR	
EDA1	CPI	
2F	CPL	
27	DAA	
35	DEC	(HL)
DD3505	DEC	(IX+d)
FD3505	DEC	(IY+d)
3D	DEC	A
05	DEC	B
0B	DEC	BC
0D	DEC	C
15	DEC	D
1B	DEC	DE
1D	DEC	E
25	DEC	H
2B	DEC	HL
DD2B	DEC	IX
FD2B	DEC	IY
2D	DEC	L
3B	DEC	SP
F3	DI	
FB	EI	
102E	DJNZ	e
E3	EX	(SP),HL
DDE3	EX	(SP),IX
FDE3	EX	(SP),IY
08	EX	AF
EB	EX	DE,HL
D9	EXX	
76	HALT	
ED46	IM0	
ED56	IM1	
ED5E	IM2	
ED78	IN	A,(C)
ED40	IN	B,(C)
ED48	IN	C,(C)
ED50	IN	D,(C)
ED58	IN	E,(C)
ED60	IN	H,(C)
ED68	IN	L,(C)
34	INC	(HL)
DD3405	INC	(IX+d)
FD3405	INC	(IY+d)
3C	INC	A
04	INC	B
03	INC	BC
0C	INC	C
14	INC	D
13	INC	DE
1C	INC	E
24	INC	H
23	INC	HL
DD23	INC	IX
FD23	INC	IY