

A GRAPHICS BASIC utasításait 7 egymástól jól elkülöníthető csoportba soroljuk:

- Nem grafikus utasítások
- Finomgrafikus utasítások
- Színező utasítások
- Ablak kezelő utasítások
- Képlem kezelő utasítások
- SPRITE kezelő utasítások
- Hangadó utasítások

Az egyes utasításokat is ezen csoportokon belül fogjuk ismertetni. Minden fejezet megértését mintaprogramok segítik elő. A végén pedig a haladó programozók találnak hasznos tudnivalókat.

Amit nem árt tudni!

A GRAPHICS BASIC az 54500-as memóriacímektől terjed felfelé 10868 byte hosszon, tehát a normál UDG területet (65368-65535) sérthetetlenül hagyja.

A program megírásakor a legfontosabb szempontok voltak: a gyorsaság, az egyszerű kezelhetőség és a kis memóriagigény, ebből adódóan kisebb hiányosságok is vannak. Bár a GRAPHICS BASIC (továbbiakban GB) microdrive kompatibilis, azért adódhatnak nehézségek is!

FONTOS! BÁRMILYEN MICRODRIVE MŰVELTETÉSE UTÁN A GB-T ÚJRA EL KELL INDÍTANI A RANDOMIZE UR 54500 UTASÍTÁSSAL!

A GB utasításfelismerő rendszere az aláhúzás "_" karakterre épül, vagyis minden új utasítást ezzel kell kezdeni. Ezután kell beírni az utasítást betűnként, mely betűk csak nagybetűk lehetnek, és közöttük **szóközöket elhelyezni TILOS!** A kulcsszó után következnek egy kötelező szókészlet, majd szükség esetén a paraméterek. Egy sorba továbbra is lehet több utasítást írni. Az új utasítások paramétereit vessző "," és pontosvessző ";" választja el egymástól. Ezek az elválasztójelek nem cserélhetők el egymással.

A vesszőt azonban értelmezni szokták a két koordináta vagy két hangmagasság szb., míg a pontosvesszőt két különböző értelmezni szokták a két koordináta vagy két hangmagasság szb. (id. pl. a PLACE utasítást).

A GB új utasításai két kivétellel egy új koordináta-rendszer használatának, melynek rögzítője a bal-felső sarokban van, és az egész képernyő megcímkeztése vele (255x191).

Nem grafikus utasítások

_HALT

Működése megfelel a Z80 HALT utasításának. A _HALT-ot ún. szinkronizálásra használhatjuk. Lényegében a program futását függeszti fel addig, amíg a processzor megszakításkérését nem kap. A Spectrum-ban ez 1/50-ed másodpercenként történik, és ez egybeesik azzal a pillanattal, amikor az ULA épp a képszerkesztést kezdi meg. Írjuk be a következő példát:

```
10 _HALT : BORDER 7 : BORDER 4 : BORDER 7 : BORDER 4 :
   BORDER 7 : GO TO 10
```

_DPOKE cc,dd

Dupla POKE, azaz a (cc) címűtől a memóriába írja (dd) a kétbyte-os számot.

_SPOKE cc,a\$

String POKE, azaz a (cc) címűtől a memóriába tölti az a\$ stringet.

_ONERR ss

Az ismert ON ERROR GO TO megvalósítása, ahol (ss) annak a sornak a száma, ahol hiba esetén a program futása folytatódik. Ez alól kivételt képeznek a következő hibázsetek: Ok., End of file, BREAK into program.

_OFFERR

Kikapcsolja az ONERR üzemmódot.

_BROFF

Hatástalanítja a BREAK billentyűt. Az utasítás kiadása után a BREAK billentyű megnyomása nem befolyásolja a program futását. VIGYÁZATI! A PROGRAM ILYENKOR MEGÁLLITHATATLAN! BANJUNK VELE ÓVATOSAN! Kivételt képeznek a magnókezelő utasítások, mert ezekre a BROFF hatástalan. A megoldás a BROFF és az ONERR utasítások együttes használatában rejlik.

_BRON

Visszatérés a normális BREAK kezeléshez. Ha valamilyen oknál fogva hibázsetet kapunk, és visszatérünk a szerkesztőbe, akkor a BROFF utasítás érvényét veszti, vagyis egy BRON utasítás is végrehatódhat!

Finomgrafikus utasítások

_LINE x1,y1 TO x2,y2

Egyeneset rajzol két abszolút koordináta közé. A koordináták PLOT utasítás szerintiek. Az összekötő TO szócskát a 'SYMBOL SHIFT + F'-tel érhetjük el.

_FILL x,y

Zárt alakzat kitöltése (x,y) koordinátától kiindulva. Az ATTRIBUTUM-okat nem kezel! Bonyolultabb alakzatokat, ill. nem pont megfelelően megválasztott koordináták esetén meg egyszerűbb alakzatokat is hibásan tölthet ki a rutin. Gépeljük be a következő programot:

```
10 CIRCLE 64,85,61
20 FILL 24,85
30 CIRCLE 191,85,61
40 FILL 191,85
50 STOP
```

Futtatáskor láthatjuk, hogy a baloldali kör nem töltődött ki teljesen. A megoldás: a FILL x. pozíciójának meg kell egyeznie a CIRCLE x. pozíciójával. Bővebben ld. a RUTIN-RÖL-RUTINRA c. könyvben.

_SIZE x,y;d

A SPRINT és a MAGNIFY utasításnál a nagyság mértékét határozza meg (x,y=1...255). A "d" pedig az x koordináta karakterenkénti növekménye. Lényegében a betűk szélességét jelöli ki. Használatával 32-nél több betűt is írhatunk egy sorba.

```
_SIZE 1,1;7 - 36 karakteres sor
_SIZE 1,1;6 - 42 karakteres sor
```

Újratervezett karakterkészlettel akár 64 betűs sorok is írhatók (SIZE 1,1;4).

_SPRINT x,y;s,m;a\$

A SIZE utasításban megadott méretben kiírja az a\$ stringet (x,y) képpont koordinátától kezdve. Az (s) paraméter a kiírt betűk stílusát határozza meg. Ennek a nyolcbites számnak minden bite egy-egy stílust jelent, amelyek egymással ötvözhetők is. A bitek jelentése a következő:

Bit	Helyiérték	Stílus
0	1	Balra dőlő
1	2	Jobbira dőlő
2	4	Telt betűk
3	8	Vastagított
4	16	Hullámzó
5	32	Remegő
6	64	Kettős képű
7	128	Raszter-pontos

A raszterpontos stílus csak nagyított karaktereknél ad élvezhető, és olvasható képet. Az (m) paraméter pedig a képernyő és a betűkép monitorozási technikáját határozza meg.

m = 0 - felülírás (OVER 0)
1 - Vagy (OR) kapcsolat
2 - És (AND) kapcsolat
4 - Kizáró vagy (XOR) kapcsolat (OVER 1)

Írjuk be a következő programot:

```
10 SIZE 1,1;8: FOR i=0 TO 6
20 _SPRINT 0,i;8;2;1,1;STR$(2*i)
   i) "+" helyiértéku stílus: "NEXT
   i
30 SIZE 2,2;16: _SPRINT 0,64;12
   8,1;"128-as stílus:"
40 _SPRINT 0,80;132,1;"128+4-es
   stílus:"
50 SIZE 1,1;7: _SPRINT 0,100;0,
   1;"36 karakteres kiírás."
60 SIZE 1,1;6: _SPRINT 0,110;0,
   1;"42 karakteres kiírás."
70 STOP
```

Ez a kis ízező csak töredékét mutatja az azoknak a lehetőségeknek, melyek az utasításban rejlenek. Az utasítás a színmemóriát is kezeli, az ATON-tól függően. Ez a színezés csak a tartós színparaméterekre vonatkozik, így az egész kiírt szöveg azonos színű lesz. Bonyolultabb színezést a következő utasításcsoport tagjai tesznek lehetővé.

Színező utasítások

COLOR y,x;c,r ;k

Az (y,x;c,r) paraméterlánc által meghatározott ablakot tölti fel (k) színkóddal. Az ablak adatai mind karakterben értendők, és a PRINT AT utasítás szerinti koordináta rendszeren belül kell meghatározni értéküket. Az (y,x) paraméterpár az ablak bal felső koordinátáját, míg a (c,r) paraméterpár az ablak méretét határozza meg.

- (y) = 0 ... 23
(x) = 0 ... 31
(c) = 0 ... 24
(r) = 0 ... 32

A (k) paraméter egy nyolcbites szám, melynek értelmezése a következő:

- 0,1,2-es bitek a TINTA (INK), a
3,4,5-ös bitek a PAPIR (PAPER) színét adják.
A 6-os bit a fényerő (BRIGHT), míg a
7-es bit pedig a villogást (FLASH) határozza meg.

Másképp megfogalmazva:

k = TINTASzín + 8*PAPÍRSzín + 64*FÉNYERŐ + 128*VILLOGÁS.

ALTER y,x;c,r ;l,m

Az (y,x;c,r) paraméterlánc által meghatározott ablakban az (l) színkódot kicseréli (m) színkódra. Az ablak paramétereinek és a színkódoknak az értelmezése azonos a COLOR utasításnál leírtakkal. Nézzünk erre is egy példát:

```
10 COLOR 0,0;24,32;3
20 FOR i=0 TO 23
30 SPRINT 0,i*8;0,0;"32b 0"
40 NEXT i
50 FOR i=63 TO 56: PAUSE 5
60 COLOR 0,0;24,32;1
70 NEXT i
80 FOR i=1 TO 50
90 LET x=INT (RND*(10-i/8))+1
100 LET y=INT (RND*(6-i/10))+1
110 COLOR RND*(24-y),RND*(32-x)
120 NEXT i
130 BORDER 2: PAUSE 0: BORDER 7
140 FOR i=0 TO 254
150 ALTER 0,0;24,32;i,i+1
160 NEXT i
```

A listában az egyes logikai részeket tagoltuk. Ezek külön-külön is működőképesek.

Ablak kezelő utasítások:

A GB szolgáltatásai közé tartozik a 16 db **ablak**, melyekkel különféle mutatóványokat is végrehajthatunk. Ezeket az ablakokat 1-től 16-ig számozzuk, és a későbbiekben ezekkel a számokkal hivatkozunk rájuk, a 0-ás számú ablak maga a képernyő, teljes terjedelmében.

WINDOW n;x,y,e,f

Definiálja egy (n) számú ablakot (x,y,e,f) paraméterek szerint. Az ablak maximális mérete a képernyő mérete! (x,y) az ablak bal felső sarkának koordinátáját, (e,f) az ablak méretét határozza meg. Minden vízszintes értelmi paraméter (x és e) karakterben, minden függőleges értelmi paraméter (y és f) pixelben értendő.

- (x) = 0 ... 31 (e) = 0 ... 32
(y) = 0 ... 191 (f) = 0 ... 192

CLW n

Törli az (n) ablak által kijelölt képernyőterületet.

INVERTW n

Invertálja az (n) ablak által kijelölt képernyőterületet. Mind a CLW, mind az INVERTW végrehajtása után a színmemória érintetlen marad!

SCROLL (fv.) n,r,s

ROLL (fv.) n,r,s

Ez a két utasítás az **ablakok tartalmának mozgására** használható. Szintaktikájuk teljesen azonos, működésében van egy kis eltérés. A ROLL utasításnál görgetés közben az ablak területéről kicsúszó részeket az ablak tüloldalán újra belépnek (WRAP-AROUND rendszer). Ezt a műveletet a paraméterektől függően a színnel is végre tudja hajtani. A SCROLL utasításnál ezzel

szemben a kilépő információk elvesznek, és a tüloldalon üres terület gördül be (OFF-LINE rendszer). Színek mozgása esetén pedig a belépő oldal az aktuális színeljelmzők jelennek meg (PAPER,INK,...). Az első paraméter helyén látható (fv.) nem egy számot, hanem a billentyűzetben szereplő három kijelölt függvény valamelyikét takarja. Ezek lehetnek:

POINT - A mozgítás csak a képpontokra korlátozódik, és a lépésszám képpontban értendő. 7-nél nagyobb lépésszám esetén addig végez karakteres eltolást, amíg a maradék lépésszám kisebb nem lesz 7-nél, majd a maradékot képpontonként eltolva fejezi be a műveletet.

ATTR - A mozgítás csak a színmemóriára korlátozódik, karakteres lépésszámmal.

SCREEN\$ - A képpontokat és a színmemóriát egyszerre mozgítja, karakteres léptékekkel.

Az (n) paraméter határozza meg azt az ablakot, amelyiket mozgítani akarjuk.

Az (r) a mozgítás irányát jelöli ki.

- (r) = 1 - balra
(r) = 2 - lefelé
(r) = 3 - felfelé
(r) = 4 - jobbra

Végül az (s) pedig a lépésszámot adja meg.

- (s) = 0 ... 255

Írjuk be a következő programot:

```
10 LIST
20 FOR K=1 TO 190 STEP 2
30 WINDOW 1;0,K;32,192-K
40 SCROLL POINT 1,2,1
50 NEXT K
60 FOR K=1 TO 40
70 COLOR RND*21,RND*28;3,3;RND
*255
80 NEXT K
90 PAUSE 100
100 ROLL POINT 0,3,192
110 PAUSE 100
120 FOR K=1 TO 20
130 INVERTW 0
140 NEXT K
150 PAUSE 100
160 ROLL SCREEN$ 0,1,32
170 PAUSE 100
180 CLW 0
190 SCROLL ATTR 0,4,32
```

Képlelem kezelő utasítások:

Egy új fogalmat kell bevezetnünk ezen utasításcsoport tárgyalása előtt. Ez a **KÉPELEM**. A képelemek olyan felhasználó által definiálható méretű **képdarabok**, melyeket a képernyőn bármikor, és bárhol elhelyezhetünk. Tartalmukat újra és újra beolvashatjuk a képernyőről, tukrozhajthatjuk, egymásba olvaszthatjuk, vagy akár az egész képelemet torlithatjuk a nyilvántartásból. A képelemek mérete tetszőleges lehet, de maximum akkora, mint a képernyő (256*192 képpont). Ezekből a képelemekből max. 255 db lehet, és 1-től 255-ig számozhatjuk őket. Nincsen sorrendbeli megkötés, tehát nyugodtan definiálhatjuk először a 32-es számút, majd a 126-osat, stb. Ha egy képelemet definiálunk, a számítógép nyilvántartásba veszi azt. Ez a nyilvántartás, másnéven képelemmemória mindig lefelé bővül, amíg el nem éri a RAMTOP-ot (amit CLEAR utasítással állíthatunk be). Ez alá a cím alá nem fog terjedni, ilyenkor **M RAMTOP NO GOOD** hibaezenettel megáll a program. A memória felosztása a következő:

23755	RAMTOP	54500	65368	65535
BASIC	KÉPELEMEK	GR. BASIC	UDG	

A képelemek definiálásakor a startcím lejjebb lép, és egy ötbyte-os lefelé lógallja ezt a területet. Ennek a fejlecnek az első byte-ja tartalmazza a képelem számát. A második, és harmadik byte adja a következő képelem címét, a negyedik és ötödik pedig a képelem méretét mondja meg. A képelemek szélessége mindig karakterekben, a magassága pedig képpontban értendő. Ezután következik a képelem grafikus adatai blockja, képpontsoroként tárolva, majd a színdatokat ugyancsak soroként. Miután a képelem magassága képpontban van megadva, nem biztos, hogy pont karakter méretűre lett definiálva a képelem. Ilyenkor, ha a fennmaradó bitek száma nagyobb 4-nél, akkor plusz még

egy színoszor is hozzáadódik az eredetéhez. Tehát, ha 4 bit magas a képelem, akkor nincsenek színadatok, de ha 5 bit magas, akkor egy sornyi színadatát rendelkezik.
FONTOS! A KÉPELEMMEMÓRIÁT EGY 0-ÁS BYTE-NAK KELL LEZÁRNI!

Alálapotban ez a startcím 54499-re áll be (0-ás byte). A NEW utasítás nem törli a képelemmemóriát.

Nos nézzük ezeket az utasításokat:

START cc

A képelemmemória kezdőcímét (cc) állítja be. Értéke a 16 bites határokon belül bármi lehet, még a RAMTOP alá is tehetjük, addig nem lesz belőle gond, amíg nem akarunk új képelemet definiálni.

ERECT

Ha valamilyen oknál fogva az egész képelemmemóriát áthelyez-
zük, azt fogjuk tapasztalni, hogy a program nem találja a ké-
pelemeket, még a START használatát után sem. Ennek az oka az
öbte-os fejlejekben keresendő, azon belül is a második és har-
madik byte a bűnös. Ilyenkor újra ki kell számítani a képek címe-
it. Erre való az ERECT, ami újra használhatóvá teszi a nyilvántar-
tást. Ilyenkor is ügyeljünk a lezáró 0 byte-ra! Ha ez nincs, az
ERECT tönkreteszi az egész memóriát!

DEF n;r,c

Definiálja az (n) számú képelemet, (r,c) méretben. Ha már volt
egy (n)-számú képelem, akkor törli a memóriából azt, majd újra-
definiálja az új méretben. A méretet meghatározó paraméterek
közül az első (r) adja az x-irányú, a második (c) pedig az y-ir-
ányú kiterjedést.

ERASE n

Törli a nyilvántartásból az (n) számú képelemet.

CLP n,s

Törli az (n) számú képelem tartalmát, és a színmemóriát (s) szín-
kóddal tölti fel. A színkód értelmezése a szokásos.

INVERTP n

Invertálja (n) számú képelem grafikus tartalmát, a színmemóriát
érintetlenül hagyva. A következő utasításoknál már eldönthetjük,
hogy a színekre is végre akarjuk-e hajtani a műveletet. Ennek
el döntésére való a következő két utasítás.

ATON

Bekapcsolja a színkezelést az SPRINT, MAGNIFY, PUT, GET,
FROM, INTO, MIXX és MRY utasításoknál.

ATOFF

Az előbb leírt utasításoknál kikapcsolja a színkezelést.

GET n;x,y;m

Beolvassa (n) képelembe a képernyő (x,y) koordinátájú, (n) kép-
elem méretű ablakát. Az (m) paraméter a képelem tartalma, és az
újonnan beolvasott adat közötti logikai kapcsolatot határozza
meg. Használatra azonos a SPRINT utasításban leírtakkal:
m = 0 - Felülírás (OVER)
1 - Vagy kapcsolat
2 - És kapcsolat
3 - Kizáró-vagy kapcsolat (OVER ^)

Ha ATON üzemmódban vagyunk, akkor a színek mindig
felülíródnak. ATOFF üzemmód esetén a színek nem változnak.

PUT n;x,y;m

Kirajtolja a képernyőre (n) képelemet (x,y) koordinátára, (m)
technikával.

FROM s;x,y TO p;m

Ez az utasítás megfelel a GET-nek csak két képelemre vonat-
koztatva. A (p) számú képelembe másolja (s) képelem egy (p)
méretű ablakát (m) monitorizált technikával. Ennek az ablaknak
a bal-felső koordinátája (x,y) az (s) képelem bal-felső sarkához
viszonyítva. Színkezelés a GET-ben leírtak szerint.

INTO p TO s;x,y;m

Ez az utasítás a PUT megfelelője két képelemre. (p) képelemet
beírja (s) képelembe, annak (x,y) koordinátájú pontjától (m)
technikával. (x) és (y) az előzőek szerint. Ebből a csoportból már
csak két utasítás van hátra. Ezek valószínűleg meg egy képelem
törlését a két koordináta tengelyre.

MIRX n

Tükrözi (n) képet x irányba (y tengelyre). Az ATON-tól függően
a színekkel együtt.

MIRY n

Tükrözi (n) képet y irányba (x tengelyre). Színkezelés szokás
szerint.

Ebből a fejezetből még egy utasítás hiányzik, amelyet a SIZE
ismertetésekor már említettünk.

MAGNIFY n;x,y;m

Az utasítás paraméterei azonosak a PUT utasításával, működé-
sükben a különbség annyi, hogy ez a SIZE-ban beállított méret-
ben nagyítja a képelemet. Különlegessége az (m) paraméter,
melynek értéke a szokásosnál 128-cal nagyobb is lehet, ilyen-
kor a SPRINT-ben megszokott rasterpontos kiíratását kapjuk.
Képelemek magnetofonra mentésével az utolsó fejezet foglaljo-
zik. Irjuk be a következőket:

```
10 CLS : CIRCLE 8,168,7 : FILL
8,168
20 ATON : DEF 1,2,16 : GET 1,0,
0;0
30 FOR I=0 TO 11
40 COLOR I*2,0,1,32;24
50 COLOR I*2+1,0,1,32;40
60 NEXT I
70 FOR I=0 TO 15
80 PUT 1:I*16,I;0
90 NEXT I
```

A 10-es és 20-as sorban definiált képelemet sorban kiírjuk
képpontoként lefelé haladva. Itt jól látszik a színek elcsúszása
és a PUT színkezelési technikája, vagyis, a színeket a képelem
között koordinátájához közelebb eső karakter koordinátára írja.
Most írjuk ehhez hozzá a következőket:

```
30 CLS
35 ATOFF
70 LET X=100: LET Y=100
80 LET C=2: LET R=2
90 PUT 1:X,Y;3
100 LET XX=X+C: LET YY=Y+R
110 IF XX>ABS C OR YY>240-ABS C
THEN LET C=C
120 IF YY<ABS R OR YY>176-ABS R
THEN LET R=R
130 PUT 1:X,Y;3: PUT 1:XX,YY;3
140 LET X=XX: LET Y=YY
150 GO TO 100
```

Ha elindítjuk a programot láthatjuk, hogy az előbbi golyó pattog
a képernyőn, de nagyon lassan. A most következő utasítascso-
port tagjai ezt a feladatot sokkal elegánsabban, és sokkal gyors-
sabban oldják meg. Előbb azonban próbáljuk ki a MAGNIFY uta-
sítást is.

```
40 FOR I=1 TO 16
50 SIZE 1,1;0
60 MAGNIFY 1:128-I*4,96-I*4;0
65 NEX I
```

SPRITE kezelő utasítások:

A GB fő szolgáltatása a 16 db függetlenül vezérelhető, és ani-
málható SPRITE. Használatuk nem csak játékoknál indokolt.
Egyéb szemléltető jellegű feladatok is hiánytalanul megoldhatók
vele. Ezeket a sprite-okat is 1-től 16-ig számozzuk. Bonyolult-
abb programszervezéssel ennél több is mozgatható, csak a se-
besseg jelenti az igazi határt. Erről majd a függelékben olvas-
tatunk többet. A sprite-ok a képernyővel kizáró-vagy kapcsola-
tba lépnek.

PLACE n;x,y

Az (n) számú sprite koordinátáit adja meg.

DIRECT n;c,r

Az (n) sprite elmozdulásának vektorát adja meg x és y sorrend-
ben. Értéke minusz 127-től plusz 127-ig terjedhet.

SCALE n;k1,k2 ... k8

Az (n) sprite animációs fázisainak (max. 8 db) képelem számait
adjuk meg vele. Nyolcnál kevesebb fázis esetén nyugodtan ír-
junk annyi elemet, amennyire szükségünk van.

_SPRON n(#)

Az (n) sprite-ot kapcsolja be. Ha (n) helyére csak egy kettőske-reszletet (#) írunk, úgy mind a tizenhat sprite –ra vonatkozik az utasítás.

_SPROFF n(#)

Az (n) sprite-ot kapcsolja ki. (#) mint az előbb.

_MODE n; c, w

A sprite-okat többféle logika szerint tudjuk vezérelni. Ezt a mozgáti logikát határozza meg a (c) paraméter. A (w) pedig egy ablak száma (0 ... 16), értelmezését lásd később. A (c) egy nyolcbites szám, amelynek az alsó öt bite van értelmezve. Amelyik bit egyesbe van állítva, az a funkció bekapcsolódik.

0. bit (h.é. 1)

A (w) paraméter által meghatározott ablakon belül kell, hogy tartózkodjon a sprite. Ha ez a bit 1 –es-re van állítva, akkor az ablak szélét elérve, a sprite visszatér. Ha ez 0, akkor az ablakból kilépve a sprite kikapcsolódik.

1. bit (h.é. 2)

A sprite –ok mozgatakor előfordul, hogy a képelemek te-rülete átfedi egymást, vagyis a két sprite ütközik egymással. Ha ez a bit egyesbe van állítva, úgy a két bekapcsolt sprite ütközésekor a sprite –ok visszapattannak egymásról.

2. bit (h.é. 4)

Két bekapcsolt sprite ütközésekor a sprite kikapcsol.

3. bit (h.é. 8)

A sprite –ok kírásai technikájából adódik, hogy ha a sprite –ot ráírjuk valamilyen – a képernyőn lévő – rajzra, vagy akár egy másik sprite –ra, akkor a sprite képe nem lesz azonos a képelemével. Ezt hívjuk **háttérütközésnek**. Ez a bit a háttérről pattanást határozza meg.

4. bit (h.é. 16)

Ha a sprite ütközik a háttérrel, akkor kikapcsolódik.

_SET n(#)

A sprite –ok kírásai technikája miatt mozgatakor előtt ki kell írni őket a képernyőre. Ez az utasítás az (n) sprite-ot írja ki függetle-nül attól, hogy be van –e kapcsolva. Amennyiben (n) helyett (#) –et adunk meg, úgy az összes bekapcsolt sprite –ra vonatkozik az utasítás. Ha a sprite kilógna a beállított ablakból, úgy ki-kapcsolja azt. Amennyiben nincs megadva a képelem száma, úgy **B Integer out of range** hibüzenetet kapunk.

_MOVE n(#)

Ez az utasítás végzi a sprite mozgatakor a beállított paraméterek szerint. Ha (n) –et adunk meg, akkor mindenképp megpróbálko-zik a sprite mozgatakorával, ha (#) –et adunk meg, akkor csak a bekapcsolt sprite –ok mozgatakorát végzi el. Hiba esetén kikap-csolja a sprite –ot.

_GO n(#); x, y

Az (n) sprite koordinátáit írja át úgy, hogy ha az be van kapcsol-va, akkor át is rajzolja az új helyre. (#) megadása esetén csak a bekapcsolt sprite –okra hajtódik végre a művelet.

_STEP n(#); x, y

Az (n) sprite koordinátáit tolja el (x,y) vektorral. x,y = -127 ... 127. Áthelyezés nincs. Szükség esetén a SET –tel kell megoldai-ni. (#) kezelése szokás szerint.

_PLUS n(#); x, y

Lényegében megegyezik a STEP utasítással, a különbség csak annyi, hogy ez a sprite mozgatakorának vektorát változtatja meg.

_CLRSP n(#)

Törli az (n) sprite adatterületét (Lényegében 0 byte-tal tölti fel). (#) esetén az összeset törli.

_SPRET n(#)

Az (n) sprite vektorait fordítja meg annak bekapcsolt vektortól füg-getlenül. (#) esetén mint előbb.

Még két utasítás ismertetése hátra van. Ezek jelentősége a Spectrum képszerkesztéséből adódik ugyanis az animáció akkor szép, ha nem látjuk a letörést és kírás művelete miatti villódzás, vagy csak alig látható. Ennek biztosítására hat végre a számító-gép minden mozgatakorát művelet előtt egy **HALT** utasítást, mely-nek leírása az előzőekben megtalálható. Ez azonban nagyobb képelem használatakor, vagy sok sprite kezelésekor nagyon le-

lassítja a program futását, ezért ilyenkor ezt inkább mellőzni kell. Ezt a szinkronizálást lehet a

_FAST utasítással ki, illetve a
_SLOW utasítással bekapcsolni.

Bármilyen animáció készítésekor érdemes mind a két üzemmód-t kipróbálni a tökéletes futás érdekében. Mint már a bevezető is utal rá, a GB csak 16 sprite –ot tud egyezre kezelni, de ez átalakítható a blokkmasoló utasítás (SPOKE) és függvénye, a (MEMORY) segítségével. Ha ezekkel folyamatosan cserélgetjük a sprite –memória tartalmát, megoldható a bővös 16 –os határ átépése.

És most nézzük a pattogós labdát:

```
NEW
10 PRINT AT 0,0;"0":_FILL 4,17
2
20 _ATON:_DEF 1;1,8:_GET 1;0,0;
0
25 CLS
30 PLACE 1;100,100:_SCALE 1;1
40 _DIRECT 1;1,1:_MODE 1;1,0
50 _SLOW:_SET
60 _MOVE 1;7: GO 60
```

Érdemes megfigyelni a mozgás folyamatosságát. Most állítsuk meg (BREAK), és írjuk át az 50 –es sort:

```
50 _FAST:_SET 1
```

Elindítva látható, hogy a program sokkal gyorsabban is futhat, de a labda most nagyon villog. Most próbáljuk ki mind a 16 sprite –ot.

```
30 FOR I=1 TO 16
40 PLACE I;(I-1)*16,RND*176:_S
CALE I;1
50 _DIRECT I;RND*13-7,RND*11-6:
_SPRON I
60 _MODE I;1,0
70 _IND I
80 _FAST:_SET #
90 _MOVE #; GO TO 90
```

Ezen a példán látható, hogy már a 16 sprite kezelése is lassú, hát még ha ennél több lenne. A 16 sprite lényege nem is az egy-zerre mozgatakor, hanem az, hogy előre beállíthatók, és a meg-felelő helyen és időben egy pillanat alatt bekapcsolhatók, fel-cserélhetők egy másikak.

Zenei utasítások:

A GB zenei utasításai a Spectrum viszonylatában színvonalasnak mondhatók. Különlegességek a hangerő állítási lehetőség. Ezzel a hangerőállítással együtt jár a hangszin változása is. Hangad-áskor a hangmagassági értékek az impulzusok közt eltérő időt adhatjuk meg. Ezen impulzusok hullámformáját, ill. hosszát állít-hatjuk be max. 8 bites. Ennek a 8 – bites számnak az értelmezé-se a következő:

A nyolc bit 1 –es jelszintet jelenti a hangszóró membránjának meghúzását. A hang kiadása a 4. bitől indul, és a 8. után az 1. –n át a 3. bit kiadásával fejeződik be. A hangerő (a hang tel-leltsége) az egymás után szereplő bitek számától függ. A 225 – –os érték esetén halljuk a legerősebb, a 4 –es érték esetén a leghalkabb hangot. A 0 teljes csendet eredményez az adott csatornán.

_SOUND h;m1,m2 (; t1,t2)

A (h) paraméter jelenti a hosszot, az (m) paraméterek pedig a két hangmagasságot határozzák meg. A hátul zárójelben látható (t1,t2) értékek beírása nem kötelező. Ezek a két hangerőt hatá-rozzák meg. Amennyiben ezt nem adjuk meg, az utolsó beállítás marad érvényben.

_DRUM s,h;m,n

Fel –lefutó hangok, effektek kiadására alkalmas utasítás, amely a SOUND utasítást használja. Működésekor (s) darab (h) hosz-zuságú hangból álló hangsorozatot ad ki, melynek kezdeti ma-gassága (m), és minden ciklusban (n) –nel növekszik. (m) és (n) egybyte-os számok, melyek összeadásakor az eredmény is egy-byte-os szám lesz. Így ha (n) < 128 akkor felelfe, ha (n) > 128 ak-kor lefele futó hangot hallunk.

Függvények

A GB nem csak új utasításokat, de új függvényeket is nyújt a fel-használónak. Ezekkel lehet lekérdezni a sprite –ok adatait, szá-mokat konvertálni stb. Az új függvényeket csak a LET utasításon belül, onmagukban lehet használni.

10 LET A=?XPOS(1)

Mint látható, az új függvények kérdőjellel kezdődnek, és a nyitó-zárójel a szó végét. Ebből következik, hogy a függvény argumentumát mindig zárójelbe kell tenni. (Ha nincs argumentuma, a zárójelpár akkor is kötelező!)

Sprite-ok adatainak lekérdezése

XPOS(n)

YPOS(n)

Az (n) sprite X (XPOS), vagy Y (YPOS) koordinátáját eredményezi.

XVEC(n)

YVEC(n)

Az (n) sprite elmozdulásvektorának X vagy Y vektorát adja (XVEC / YVEC).

SPMOD(n)

Az (n) sprite üzemmódbite-ját adja eredményül. Ez egy nyolcbites szám, melynek minden bite mást jelent.

0. bit: Ha ez 1, akkor a sprite be van kapcsolva.

1. bit: Ez a sprite és az ablak kapcsolatát mondja meg.

0-esetén az ablak határánál a sprite visszapattan.

2. bit (h.é. 4): két bekapcsolt sprite ütközésekor a sprite-ok visszapattannak egymásról.

3. bit (h.é. 8): Két bekapcsolt sprite ütközésekor a sprite kikapcsol.

4. bit (h.é. 16): Ez a bit a háttérrel pattanást határozza meg.

5. bit: Ha a sprite ütközik a háttérrel, akkor kikapcsolódik.

CRSP(n) — (SPMOD 6. bite)

Ha bármelyik sprite ütközés vizsgálat be van kapcsolva, ütközések ez a bit egyesbe vált.

SCRN(n) — (SPMOD 7. bite)

U.a. mint az előbb, csak háttér-ütközés esetén.

Egyéb függvények

ADDR(n)

Ez a függvény az (n) sorszámu képelem kezdőcímét adja eredményül. Vigyázat, ez az 0-byte-os fejléc első byte-ja lesz! Nem létező képelemszám esetén (Subscript wrong) üzenetet kapunk. Ha az (n) értéke 0, akkor a képelemmemória kezdőcíme lesz az eredmény. Vagyis, ha ki akarjuk menteni eddigi képeinket, akkor azt a következőképp tehetjük:

```
LET A=?ADDR(0)
SAVE "Kepek" CODE A, 54500-A
```

DPEEK(cc)

Ez a DPOKE utasítás párja. A (cc) címen található kétbyte-os számot adja eredményül.

ERRCOD()

Ennek a függvénynek nincs paramétere! Ha használjuk az ONERR utasítást, úgy hiba után ezzel tudhatjuk meg a hiba kódját.

MEMORY\$(cc,II)

A SPOKE utasítás párja. Eredménye egy olyan string, melynek hossza (II) db. karakter, és tartalma a memóriának (cc) címen kezdődő területe.

BITS(n)

Eredmény szintén string, melynek hossza 8 byte, és tartalma az (n) kétye-byte-os szám bináris alakja.

DEC(as)

Bináris-decimális konverzió. Az (as) tartalmazza binárisan (8 karakteren) a számot, amely megadható számokkal, és grafikus jelekkel is. Grafikus jelek esetén a:

SPACE - (CHR\$ 32) jelent 0-át,
bármilyen más — (de nem "0") jelent 1-et.

A "_USER" utasítás

Talán ez a GB legértékesebb szolgáltatása. Használatával lehetőség nyílik a BASIC utasításkészlet bővítésére, a felhasználói igényei szerint. Alapállapotban, a "_USER" végrehajtása csak egy _HALT-lel lesz egyenlő (mert erre van beállítva).

A GB utasításfelismerő rendszere az " " aláhúzás karakterre épül. Amennyiben egy sor értelmezésekor ezzel találkozunk, elkezd összehasonlítani a táblázatban lévő utasításszavakat, a végrehajtandó utasítással, és ha megtalálja az egyező szót, az annak az utasításnak megfelelő címen folytatja a végrehajtást. Ahhoz, hogy a "_USER" szó esetén saját rutinunkat hívja meg a rendszer, az 56783-as címen el kell helyezni annak kezdőcímét (pl. DPOKE utasítással).

Ezzel még nem vagyunk készen, mert meg kell határozni az utasítás szintaxisát is, illetve azt, hogy melyik értéket melyik regiszterbe tölts. Ebben segítségünkre lesznek a ROM-ban található rutinok. Megjegyezzük, hogy ezek a rutinok visszatérőskor nem hoznak értéket, hanem azt a kalkulátor stack-ben helyezik el (kivéve, ha csak szintaktikai ellenőrzés folyik). Fontos még, hogy ezek átállítják az értelmezési rendszervázlatot, amely visszatérőskor az első nem értelmezett karakteren áll, és ezt a karaktert az A regiszterbe is betöltik.

CALL #1C7A : két egymástól vesszővel elválasztott numerikus : adatot vár

CALL #1C82 : egy numerikus változót vár

CALL #1C8C : egy stringet olvas

RST 32 : az értelmezési mutatót lépteti tovább.

Amennyiben az aktuális helyen nem olyan típusú az operandus, vagy értelmezhetetlen, hibázenetet kapunk.

Ha két operandust el akarunk választani, akkor az elhatárolójel vizsgálatáról is gondoskodni kell, majd tovább kell léptetni a mutatókat, és meg kell hívni az újabb ellenőrző rutint.

Példaként nézzük az _SPOKE utasítást:

```
10 SPOKE RST 32
20 CALL #1C82 : numerikus operandus
30 CP " :
40 JP NZ, #1C8A : "C Nonsense in Basic"
50 : üzenet
60 RST 32
70 CALL #1C8C
CALL 55487 : ld. alább
```

A szintaxis leírása után el kell helyezni egy CALL 55487 utasítást, a végrehajtás, és a szintaktikai művelet szeptévalasztására. A rutin csak végrehajtás (RUN) esetén fog tovább futni.

Ezek után következhet az értékek valódi beolvasása.

CALL #2DA2 : A 'BC' regiszterbe tölti a kerekített számot

CALL #2D05 : Az akkumulátorba tölti a kerekített számot

CALL #2BF1 : Ezt többek között stringek adatainak lehívására

használjuk.

DE - címe

BC - hossza

Ezek alapján nézzük az SPOKE rutinjának folytatását:

```
80 CALL #2BF1 : string paraméterei
90 PUSH DE : ezeket eltároljuk, mert megváltoztatná a
: következő rutin
100 CALL #2DA2 : a kezdőcím
110 LD E,C
120 LD D,B
130 POP BC
140 POP HL
150 LDIR : tényleges másolás
160 RET : visszatérés a BASIC-értelmezőbe.
```

Ez a rutin összesen 31 byte-ot foglal el. Látható, hogy sokkal takarékosabb, mint a POKE és a RANDOMIZE USR... forma.

Néhány tanács:

- A CALL 55487 utasítás előtt ne hagyjunk semmit a veremben
- A végső RET, illetve a hibarutinok hívása előtt kapcsoljuk be a megszakítást (EI)
- Az IV regisztert ne változtassuk meg.

És végül:

Ha kevés lenne ez az egy utasítás, kiterjeszthető a következőkkel:

```
DPOKE 56783, START
10 USER ORG START
20 LD DE, TABLA : az új táblázat címe
30 LD 56892 : elugrik a kereső rutinra.
```

Az új táblázatban az utasításszavak végét egy szököző jelenti (pl. DEFN "LINE"), ezután kell elhelyezni az utasításrutin címet (itt DEFN "DPOC"), majd következhetnek az újabb szavak illyen módon kódolva. Az utolsó szó kezdőcíme után elhelyezett 255-ös kód (DEFB 255) jelenti a táblázat végét!

Graphics Basic

SOUND utasítás hangmagasság táblázata

