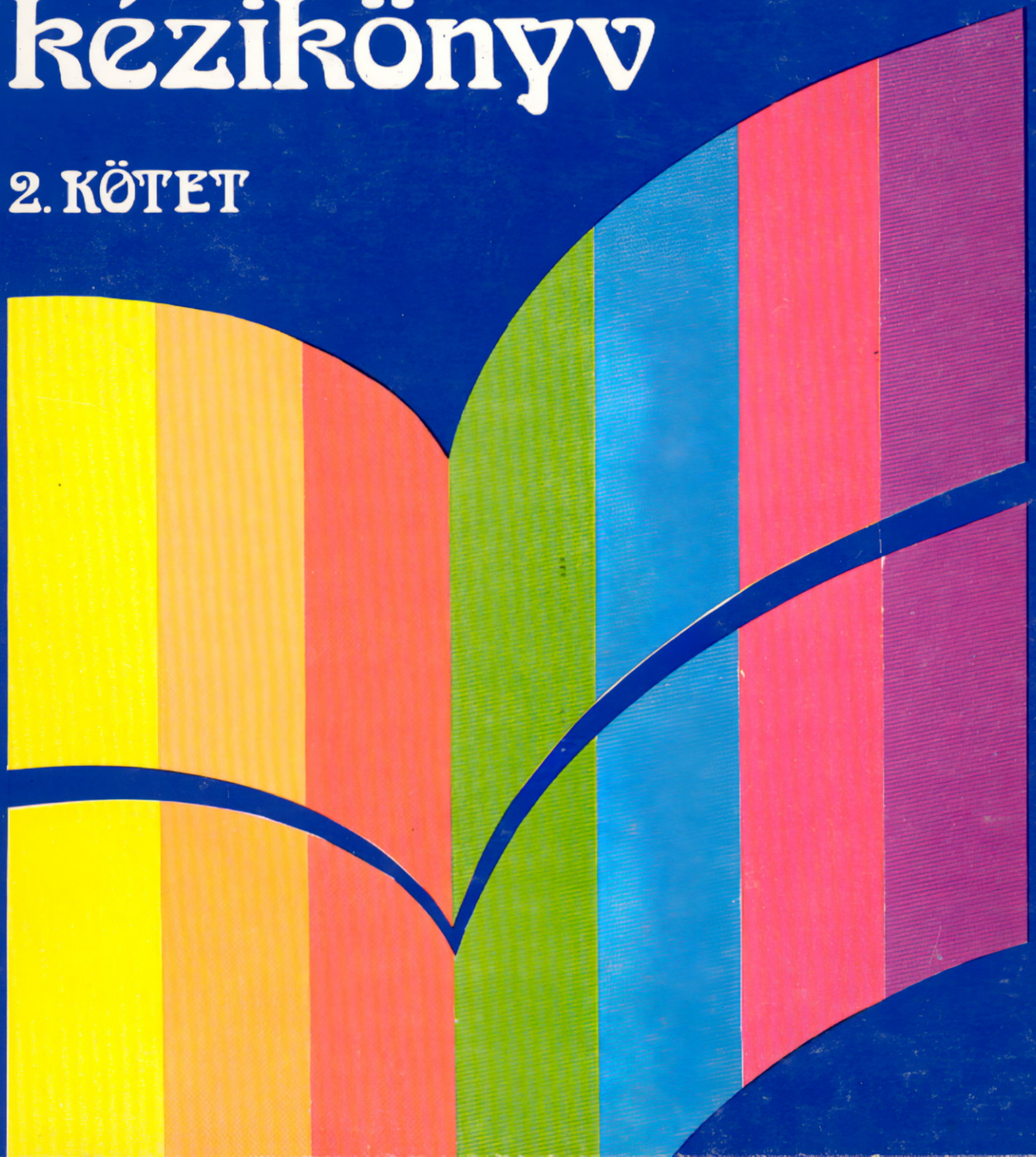
DR. ÚRY LÁSZLÓ

Commodore 64
C 128/64 üzemmód
felhasználói kézikönyv

2. KÖTET



DR. ÚRY LÁSZLÓ

Commodore 64
C 128/64 üzemmód
felhasználói kézikönyv

2. KÖTET

ALKALMAZÁSTECHNIKAI TANÁCSADÓ SZOLGÁLAT
BUDAPEST, 1989

Lektorálta: Donát János

Vadnai Szabolcs

Kiadó: LSI Alkalmazástechnikai Szolgálat
Felelős kiadó: Dr. Kovács Magda
Témafelelős: Székely László
Technikai szerkesztő: Kiss László
ISBN I. kötet: 963 592 952 8
ISBN II. kötet: 963 592 953 6
ISBN összkötet: 963 592 954 4
Eng. szám: 17606
BÁSTYA-Text-Print
Felelős vezető: Balanyi István

Tartalomjegyzék

I. kötet

II. kötet

7. Grafikus lehetőségek

7.1 Bevezetés

A C-64 mikroszámítógép elsődleges outputként a televíziós képernyőt (VDU, video display unit) használja. Ez azt jelenti, hogy az interpreter üzenetei, a PRINT utasítás segítségével kiíratott mennyiségek, az INPUT-prompt mind az elsődleges outputon jelennek meg. A CMD utasítás segítségével megváltoztathatjuk az elsődleges outputot. A számítógép bekapcsolásakor az elsődleges output azonban a VDU. A képernyő egy homogén, egyszínű keretből és az igazi információt hordozó, 320x200 pontból álló pontrácsból áll. A pontrács pontjai – bizonyos korlátozással – a C-64 által használt 16-féle színnel lehetnek kiszínezve. Ha csak kétféle színt használunk, akkor az egyes pontokról azt mondjuk, hogy be-, illetve kikapcsolt állapotban vannak. A 320x200-as pontrács a karakter üzemmódok esetén 8x8-as részekre esik szét. Így összesen 25 sor és 40 oszlop keletkezik. Ezeket a 8x8-as részeket – mivel éppen egy karakter tárolására alkalamasak – karakterhelyeknek hívjuk. A C-64 '**sprite**'-jai, fantomjai 24x21 pontból álló képrészek, amelyek a 320x200-as pontrács tartalmától **függetlenül**, a képernyő tetszőleges részén megjeleníthetők.

A Commodore 64 grafikai lehetőségeit a 6567 Video Interface Chip (VIC-II chip) biztosítja. A különböző üzemmódok a megszakító rendszer segítségével keverhetők. Lehetséges például a képernyő felső részét nagy felbontású üzemmódban az alsó részt karakteres üzemmódban használni; a sprite-ok pedig mindezekkel együtt kezelhetők.

A VIC-II chip a következő grafikus üzemmódokkal rendelkezik:

A./ **Karakteres üzemmód**
1./ Standard karakter üzemmód
2./ Többszínű karakter üzemmód
3./ Bővített háttérszín üzemmód

B./ **Bit térképes üzemmód**
1./ Standard bit térképes üzemmód
2./ Többszínű bit térképes üzemmód

C./ **Sprite-ok**
1./ Standard sprite-ok
2./ Többszínű sprite-ok

A VIC-II chipnek 47, többségében írható/olvasható regisztere van. Ezek a regiszterek közvetlen összeköttetésben állnak a C-64 memóriájának 53248-53294 című byte-jaival ($D000-$D02E). A regiszterek tartalma határozza meg a video-chip működését. Pontosabban azt kellene mondanunk, ezek a regiszterek határozzák meg, **hogyan** jelenítse meg a chip a memóriában tárolt információt. A C-64 memóriájának bizonyos további részei pedig azt az információt tárolják, **amit** a képernyőn meg kell jeleníteni. A két legfontosabb memóriarész (alapértelmezésben):

a/ a képernyő memória (1024-2023, $0400-$07E7)
b/ a színmemória (55296-56295, $D800-$DBE7)

További memóriarészek határozzák meg a karakterek és a sprite-ok alakját.

Általánosságban még két további észrevételt kell tennünk. Az egyik a video-chip vezérlésére vonatkozik. A beépített BASIC-et használó programokból a video-chip csak a POKE/PEEK utasításpár segítségével vezérelhető, ez – főleg kezdőknek – kényelmetlenné teszi a használatát.

A másik fontos észrevétel, hogy a VIC-II chip által használt bizonyos memória-részek helye rögzített (ilyen például – egyes üzemmódokban – a VIC-II regiszterek vagy a színmemória helye). Az összes többi információnak a 64 K-s memória ugyanazon 16 K-s szeletében kell elhelyezkednie.

A négy lehetséges szeletet az alábbi táblázat mutatja. A szelet megváltoztatásához a CIA#2 interface chip A kapujának első két bitjén a táblázatban is látható értékeket kell outputként kiadni. Az, hogy a kapu valamelyik bitje input-e vagy output, az a kapuhoz tartozó **DDR-regiszter** (**data direction register**) megfelelő bitjeitől függ. Ha valamelyik bit magas, az outputot definiál. Ahhoz, hogy a szelet átállítását megoldjuk, már csak azt kell tudnunk, hol találhatók a CIA#2 regiszterei. Az A kapu 8 bitje a $DD02(56578) cimen helyezkedik el. A megoldás:

```
10 POKE 56578,PEEK(56578) OR 3
20 REM A 0. ES AZ 1. BITEK KIMENETEK
30 POKE 56576,(PEEK(56576) AND 252) OR X
```

(A program futtatása meglehetősen furcsa képernyőképet eredményez!)

Az X értékei a következőt jelentik:

X	Bitek	Szelet	Kezdőcím	VIC-II chip tartomány
0	00	3	49152	$C000-$FFFF
1	01	2	32768	$8000-$BFFF
2	10	1	16384	$4000-$7FFF
3	11	0	0	$0000-$3FFF

Bekapcsoláskor vagy restart után az interpreter a 0. szeletet használja ($0000-$3FFF).

Képernyő memória

A képernyő memória helyzetét az 53272($D018) címen levő $18-as regiszter bizonyos bitjei határozzák meg. Ez a regiszter mondja meg azt is, hogy melyik karakterkészletet használja a video chip. Ezért óvatosnak kell lennünk, hogy csak a regiszter megfelelő bitjeit módosítsuk. A képernyő áthelyezésére a következő utasítást használhatjuk:

```
POKE 53272,(PEEK(53272)AND 15) OR X
```

ahol X értéke a következőt jelenti:

X	Bitek	Relatív helyzet a szelet elejéhez képest decimálisan	hexadecimálisan
0	0000XXXX	0	$0000
16	0001XXXX	1024	$0400 (alapértelmezés)
32	0010XXXX	2048	$0800
48	0011XXXX	3072	$0C00
64	0100XXXX	4096	$1000
80	0101XXXX	5120	$1400
96	0110XXXX	6144	$1800
112	0111XXXX	7168	$1C00
128	1000XXXX	8192	$2000
144	1001XXXX	9216	$2400
160	1010XXXX	10240	$2800
176	1011XXXX	11264	$2C00
192	1100XXXX	12288	$3000
208	1101XXXX	13312	$3400
224	1110XXXX	14336	$3800
240	1111XXXX	15360	$3C00

Színmemória

A karakteres üzemmód szín-memóriája nem mozgatható. Függetlenül a 16K-s szelet megválasztásától mindig az 55296 ($D800) címtől az 56295 ($DBE7) címig tart. A képernyő memória és a szín-memória különböző grafikus üzemmódokban különbözőképpen használható. Egy üzemmódban létrehozott kép gyakran teljesen másként néz ki, amikor másik grafikai üzemmódban kerül kijelzésre. A video-chip csak az alsó négy bitet használja a szín generálására, így 16 különböző színt tudunk használni.

Grafikus üzemmódban a bittérkép szineit a képernyő memória tartalma határozza meg. Ebben az esetben a képernyő memóri csak és kizárólag szín-információt hordoz.

Karakter memória

Bármely grafikus jel egy 8x8-as pontrács segítségével ábrázolható, a pontok mindegyike bekapcsolt (1) vagy kikapcsolt (0) állapotban van. A C-64 a karakterek alakját a karakter generátor ROM-ban tárolja.

Minden egyes karakter alakja 8 byte-on tárolódik. Minden egyes byte a karakternek valamely sorát reprezentálja (azon belül minden bit egy pontot reprezentál). A 0 bit azt jelenti, hogy a pont ki van kapcsolva, az 1-es bit pedig azt, hogy a pont be van kapcsolva.

A karakter memória a ROM-ban az 53248 helyen kezdődik (amikor az I/O ki van kapcsolva). Az első 8 byte az 53248 ($D000) helytől az 53255 helyig ($D007) a @ jel alakját tartalmazza, amelynek a képernyő kódja nulla. A következő 8 byte, az 53256 ($D008) helytől az 53263 ($D00F) helyig az A betű pontmátrixát tartalmazza:

Kép	Bináris	Decimális	Cím
**	00011000	24	53256
****	00111100	60	53257
** **	01100110	102	53258
*****	01111110	126	53259
** **	01100110	102	53260
** **	01100110	102	53261
** **	01100110	102	53262
	00000000	0	53263

Az egyenként 256 karakterből álló karakter készletek a memóriában külön-külön 2K (2048 byte) helyet foglalnak, karakterenként 8 byte-ot. Mivel összesen két karakter készlet van, ezért a karakter generátor ROM 4K helyet foglal el.

Természetesen a VIC-II chipnek lényegtelen, hogy az egyes grafikus jeleknek milyen alakjuk is van. Ha a video chip a képernyő memória valamelyik címén – mondjuk – 132-t talál, akkor megvizsgálja, hogy a karakter generátor ROM-ban melyik byte-ok tartoznak ehhez a jelhez, és annak megfelelően jeleníti meg a képernyőn. Ilyen módon a karaktert generáló ROM cseréjével elérhetjük, hogy tetszőleges jeleink lehessenek. Természetesen egy ROM kicserélése nem a legegyszerűbb programozói fogás. A video-chip azonban a memória bármely részéből kiválaszthatja a karaktereket. Egy programozható regiszter segítségével a karakter memória helyét a RAM-ban is kijelölhetjük.

A karakter memória helyzetét a VIC-II chip 53272($D018) címen levő regiszterének 3 bitje adja meg. A 3., 2., 1. bitek jelzik, hogy a kiválasztott 16K-s szeleten belül melyik 2K-s blokkban van a karakter készlet elhelyezve. A 0. bitet nem veszi figyelembe. Ne feledjük, hogy ez ugyanaz a regiszter, ami meghatározzza, hogy hol van a képernyő memória elhelyezve. Vigyáznunk kell, hogy a képernyő memória bitjeit ne zavarjuk. A karakter memória helyzetének megváltoztatásához a következő BASIC utasítást lehet

használni

```
POKE 53272,(PEEK(53272)AND240) OR X
```

ahol az X értéke a következőket jelenti:

X	Bitek	A karakter memória relatív helyzete decimálisan		hexadecimálisan
0	XXXX000X	0		$0000-$07FF
2	XXXX001X	2048		$0800-$0FFF
4	XXXX010X	4096	* D	$1000-$17FF
6	XXXX011X	6144	*	$1800-$1FFF
8	XXXX100X	8192		$2000-$27FF
10	XXXX101X	10240		$2800-$2FFF
12	XXXX110X	12288		$3000-$37FF
14	XXXX111X	14336		$3800-$3FFF

A fenti táblázat *-gal jelölt sorai a 0. és 2. szelet kiválasztása mellett igen furcsán viselkednek. Ebben az esetben a VIC-II chip a karaktereket nem a RAM-ból, hanem a CHARACTER ROM-ból veszi. Ez természetesen azt jelenti, hogy ebben az esetben a RAM-ban, a megfelelő helyen tárolt karakterek **helyett** a ROM-ból veszi a karaktereket. Így a standard karakterkészlet használata nem igényel külön memóriát. A CHARACTER ROM az I/O regiszterek 'mögött' található a $D000-$DFFF (53248-57343) címeken (D a default értéket jelenti, bekapcsoláskor ezzel kezd az interpreter dolgozni).

Mint említettük, a ROM karakterek által elfoglalt címek ugyanazok, mint például a VIC-II chip vezérlő regisztere által elfoglaltaké. Ez azért lehetséges, mert ugyanazokat a helyeket foglalják ugyan el, de nem ugyanabban az időben. Amikor a VIC-II chipnek szüksége van arra, hogy karaktert kérjen, a ROM bekapcsolódik. Máskülönben ezt a területet az I/O vezérlő regiszterek foglalják el, és a ROM karaktereket csak a VIC-II chip érheti el.

Szükség lehet arra, hogy hozzáférjünk a ROM karakterekhez, pl. ha saját karaktereket akarunk használni, és le akarunk másolni néhány ROM karaktert. Ebben az esetben ki kell kapcsolnunk az I/O regisztereket, bekapcsolni a karakter ROM-ot és elvégezni a másolást. Amikor elkészültünk, újra vissza kell kapcsolnunk az I/O regisztereket. A másolási folyamat alatt (amikor az I/O ki van kapcsolva) a program nem szakítható meg (legalábbis a billentyűzetről). Ez azért van, mert az I/O regiszterekre szükség van a megszakítások kezeléséhez. Ha ezt elfelejtjük és végrehajtunk egy megszakítást, akkor furcsa dolgok fognak történni. A billentyűzet sem olvasható a másolási folyamat alatt. A billentyűzet és a többi megszakítás letiltására a következő POKE utasítást kell használni:

```
10 V=56334:POKE V, PEEK(V) AND 254
```

A ROM karakterek most a C-64 memóriájában az 53248-57343 ($D000$DFFF) címeken helyezkednek el.

Miután befejeztük a karakterek másolását a karakter ROM-ból, és készen állunk arra, hogy folytassuk a programot, akkor a megszakításokat újra engedélyezni kell:

```
10 V=56334 : POKE V, PEEK(V) OR 1
```

A következő POKE utasítás kikapcsolja az I/O regisztereket, és bekapcsolja a ROM karaktert:

```
POKE 1,PEEK(1) AND 251
```

Az I/O szabályszerű működését a következő POKE utasítással érhetjük el:

```
POKE 1, PEEK(1) OR 4
```

A karakterkészlet helyzete és tartalma a ROM-ban a következő:

Blokk	Cím		VIC-II kép	Tartalom
	dec.	hex.		
0	53248	D000-D1FF	1000-11FF	nagybetűk
	53760	D200-D3FF	1200-13FF	grafikus karakterek
	54272	D400-D5FF	1400-15FF	inverz nagybetűk
	54784	D600-D7FF	1600-17FF	inverz grafikus karakterek
1	55296	D800-D9FF	1800-19FF	kisbetűk
	55808	DA00-DBFF	1A00-1BFF	nagybetűk és grafikus jelek
	56320	DC00-DDFF	1C00-1DFF	inverz kisbetűk
	56832	DE00-DFFF	1E00-1FFF	inverz nagybetűk és grafikus karakterek

7.2 Karakteres üzemmódok

Ezt a fajta üzemmódot alacsony felbontású üzemmódnak is szokás nevezni. Ebben az esetben a képernyő egy karakternyi (másként 8*8 pontnyi) helyét a karakter-memóriában meghatározott 256 karakter valamelyike töltheti be. A képernyő 25 sorra és 40 oszlopra tagozódik, és a szerkesztő karakterek segítségével könnyen beállíthatjuk azt a pozíciót, amelyikbe írni szeretnénk. A karakter üzemmódok csak annyiban térnek el egymástól, hogy a karakterek színe és alakja másként jön létre a képernyő- , a szín- és a karakter-memóriából.

Standard karakter üzemmód

Ebben az üzemmódban a video-chip a karakter-memóriából olvassa ki a karakter alakját és a színmemóriának megfelelő színnel írja azt ki. A képernyő szerkesztő és a BASIC interpreter a bekapcsolás után ebben az üzemmódban dolgozik. Ebben az esetben a 'karakter színe' meghatározza a 8*8-as pontrács bekapcsolt, illetve be nem kapcsolt pontjainak színét is. Ahhoz, hogy a programból tetszőleges színű karaktert írhassunk ki, két mód is lehetséges.

Példák:

```
(i)  PRINT "<HOME><CTRL-CYN>A"
(ii) POKE 1024,1:POKE 13*4096+8*256,3
```

A (ii) példa a képernyő első karakterhelyére egy sárga A betűt ír ki. A megoldás egyszerű, a képernyő-, illetve a szín-memória megfelelő helyére a megfelelő értékeket beírtuk. A másik lehetséges megoldást az (i) példa mutatja; a PRINT utasításban elhelyeztünk egy szín-vezérlő karaktert is.

Többszínű karakter üzemmód

A standard karakter üzemmódban a karakterek színét a szín-memória teljes egészében meghatározza. A többszínű karakter üzemmódban a karaktereket alkotó (ki- vagy bekapcsolt állapotban levő) pontok négy, tetszőlegesen definiált színnel jelenhetnek meg a képernyőn. A négy szín közül három mindegyik karakterhelyen ugyanaz, a negyedik szín karakterenként változhat, és megegyezik a színmemóriában tárolt színnel. Ebben az esetben a karakterek alakját megadó 8 byte mást jelent, mint a standard karakter üzemmód esetében. Többszínű karakter üzemmódban a 8 byte egy 4x8-as pontrácsot definiál. A pontrács minden pontjába a 0-3 egész számokat képzelhetjük, ezek felelnek meg a már előbb is említett színeknek. A 8 byte bitjei párosával összekapcsolódnak és úgy határozzák meg a pontrács pontjainak 'színét'. A képernyőn ez természetesen 8*8-as alakban jelenik meg, vízszintesen a 4x8-as pontrács minden pontja 'megduplázódik'.Többszínű karakter üzemmódban tehát a vízszintes felbontás a felére csökken.
Például tekintsük az "A" betű képét, és ahogy a karaktermemóriában tárolásra kerül:

Kép	Bináris	PEEK
**	00011000	24
****	00111100	60
** **	01100110	102
******	01111110	126
** **	01100110	102
** **	01100110	102
** **	01100110	102
	00000000	0

Többszínű üzemmód esetén:

KÉP	BINÁRIS
..AABB..	00011000
..CCCC..	00111100
AABBAABB	01100110
AACCCCBB	01111110
AABBAABB	01100110
AABBAABB	01100110
AABBAABB	01100110
.....	00000000

Ezen a képen az A-val jelölt helyek háttér#1 színűek lesznek, míg a B-vel jelölt helyek a háttér#2 színt, a C-vel jelölt helyek pedig a karakter színét használják. A "."-tal jelölt helyek háttér#0 színűek lesznek. A bit-párok a következő táblázatnak megfelelően használják a 8 byte-ot:

Bit pár	Szín regiszter	Címe
00	Háttér#0 szín (képernyő szín)	53281($D021)
01	Háttér#1 szín	53282($D022)
10	Háttér#2 szín	53283($D023)
11	A szín-memóriában az alsó három bit által meghatározott szín	

A többszín üzemmód bekapcsolásához a VIC-II vezérlő regiszter 53270 helyen levő ($D016) 4-es bitjét 1-re kell állítani a következő POKE utasítással:

```
POKE 53270,PEEK(53270)OR 16
```

A többszín karakter üzemmód kikapcsolásakor az 53270 cím 4. bitjét 0-ra kell állítani a következő POKE utasítással:

```
POKE 53270,PEEK(53270)AND 239
```

A többszín üzemmód kiválasztása után a képernyőn minden egyes karakter helyére a többszín üzemmód külön-külön beállítható vagy kikapcsolható. Ezt a karakterhelyhez tartozó színmemória 3.bitje vezérli. Ha a színmemóriában levő szám kisebb, mint 8 (0-7), akkor a megfelelő karakterhelyen standard karakter jelenik meg, a választott színben (0-7). Ha a színmemóriában elhelyezett szám nagyobb vagy egyenlő 8-cal (8-tól 15-ig), akkor az a karakterhely többszín üzemmódban jelenik meg (a színt az utolsó 3 bit határozza meg).

Példák:

```
100 POKE 53281,1: REM HATTER SZINE
110 POKE 53282,3: REM ELOTER SZINE
120 POKE 53283,8: REM HATTER#2 SZINE
130 POKE 53270,PEEK(53270) OR 16
140 REM TOBBSZINU UZEMMOD BE
150 C=13*4096+8*256 : REM C=SZIN MEMORIA ELEJE
160 PRINT CHR$(147);"LSI ATSZ"
170 FOR I=1 TO 9
180 POKE C+I,8: REM TOBBSZINU UZEMMODBAN SZIN ALLITAS
190 NEXT
```

Ebben a példánkban a képernyő alapszíne fehér, a karakter színe fekete, az egyik szín regiszter ciánkék (zöldes kék), a másik narancs.

A 100-120 sorokban beállítottuk a háttér színeit. Ezek a színek nem tartoznak külön hozzá a karakterekhez, ezért megváltoztatásukkal az egész képernyő színstrukturáját megváltoztathatjuk.

Következő példánk a többszínű üzemmód használatát mutatja be. A 100. sorban beállítjuk a többszínű üzemmódot. Először a 130. sor ír ki karaktereket, majd a 160-190 ciklus véletlenszerűen változtatja a karakterek színét.

```
100 POKE 53270,PEEK(53270) OR 16: REM TOBBSZINU UZEMMOD BE
105 POKE 53281,12: PRINT "<CTRL-BLU>";: REM HATTERSZIN
110 PRINT CHR$(147);CHR$(18);
120 PRINT"<C=-5>";
130 FOR L=1 TO 22:PRINT CHR$(85);:NEXT
135 FOR T=1 TO 500:NEXT:PRINT
140 PRINT"<CTRL-CYN>";
145 FOR L=1 TO 500:NEXT
150 PRINT"<CTRL-BLK>NYOMJON LE EGY BILLENTYUT !"
160 GET A$: IF A$="" THEN 160
170 X=INT(RND(1)*16)
180 POKE 53282,X
190 GOTO 160
```

A <CTRL> és a szín billentyűk használatával a karakterek bármilyen színűre változtathatók, beleértve a több-szín karaktereket is. Például gépeljük be az alábbi parancsot:

```
POKE 53270,PEEK(53270) OR 16: PRINT "<CTRL-RED>"
```

A READY vagy bármi más, amit begépelünk, többszín üzemmódban jelenik meg.

A következő példa többszínű programozható karaktereket használ:

```
 10 REM ****************************
 20 REM * TOBBSZINU PROGRAMOZHATO KARAKTEREK *
 30 REM ****************************
 40 POKE 56334,PEEK(56334) AND 254
 50 REM KARAKTERKESZLET BEKAPCSOLASA
 60 POKE 1,PEEK(1) AND 251
 70 :
 80 REM A ROM-BOL ATMASOL 63 KARAKTERT
 90 FOR I=0 TO 63
100 REM 8-BYTEONKENT MASOL EGY KARAKTERT
110 FOR J=0 TO 7
120 REM EGY BYTE MASOLASA
130 POKE 12288+I 8+J,PEEK(53248+I 8+J)
140 NEXT J,I
150 :
160 REM AZ I/O VISSZAKAPCSOLASA ES
170 REM MEGSZAKITASOK ENGEDELYEZES
180 POKE 1,PEEK(1) OR 4
190 POKE 56334,PEEK(56334) OR 1
200 REM A KARAKTERMUTATO ALLITASA
210 POKE 53272,(PEEK(53272)AND 240)+12
220 :
230 POKE 53270,PEEK(53270) OR 16
240 POKE 53281,0 : REM HATTERSZIN#0 FEKETE
250 POKE 53282,2 : REM HATTERSZIN#1 PIROS
260 POKE 53283,7 : REM HATTERSZIN#2 SARGA
270 :
280 REM AZ UTOLSO 4 KARAKTER MODOSITASA
290 FOR CHAR=60 TO 63
300 FOR BYTE=0 TO 7
310 READ NUMBER : REM BEOLVAS 1 BYTE-T
320 POKE 12288+(8*CHAR)+BYTE,NUMBER
330 NEXT BYTE,CHAR
340 :
350 REM AZ UJ KARAKTER KIIRASA
360 PRINT"<CLR>"TAB(255)CHR$(60)CHR$(61)TAB(55)CHR$(62)CHR$(63)
370 GET A$:IF A$="" THEN 370
380 POKE 53277,21:POKE 53270,PEEK(53270)AND 239:
390 REM A $ KARAKTER UJ ALAKJA
410 DATA 129,37,21,29,93,85,85,85:
420 DATA 66,72,84,116,117,85,85,85
430 DATA 87,87,85,21,8,8,40,0
440 DATA 213,213,85,84,32,32,40,0
450 END
```

Bővített háttérszín üzemmód

A bővített háttérszín üzemmód lehetőséget nyújt arra, hogy minden egyes karakter **háttér színét** külön is beállítsuk. Például ebben az üzemmódban megjeleníthetünk egy kék karaktert sárga háttérrel egy fehér képernyőn.

A bővített háttérszín üzemmódnak négy regiszter áll rendelkezésére. Mindegyik regiszter a 16 szín bármelyikére állítható. Bővített háttérszín üzemmódban a szín-memóriát az előtér szín tárolására használjuk. A szín-memória használata tehát ugyanolyan, mint a standard karakter üzemmódban.

A bővített háttérszín üzemmód határt szab a különböző megjeleníthető karakterek számának. Ha a bővített háttérszín üzemmódot használjuk, akkor csak a karakter ROM-ban levő első 64 karakter (vagy a programozható karakter készlet első 64 karaktere) használható. Ez azért van, mert a karakter kód első két bitje a háttérszínt definiálja.

Az 'A' betű kódja 1. Ha a bővített háttérszín üzemmódot használjuk, és a POKE utasítással 1-et viszünk a képernyő memóriába, az "A" betű fog megjelenni. Ha POKE utasítással 65-öt viszünk a képernyőre, akkor azt várnánk, hogy a 129-es karakter kóddal rendelkező karakter jelenik meg, ami az inverz A. Nem ez történik. A bővített háttérszín üzemmódban ehelyett ugyanazt a nem inverz 'A' betűt kapjuk mint az előbb, de más háttérszínnel. A következő táblázat megadja az összefüggést:

Karakter kód intervallum	**bit 7**	**bit 6**	**Háttérszín regiszter**	
			szám	**cím**
0-63	0	0	0	53281($D021)
64-127	0	1	1	53282($D022)
128-191	1	0	2	53283($D023)
192-255	1	1	3	53284($D024)

A bővített háttérszín üzemmód használatához a VIC-II 53265 című regiszterének 6. bitjét 1-re kell állítani. Ezt a következő POKE utasítással érhetjük el:

```
POKE 53265, PEEK(53265) OR 64
```

A bővített háttérszín üzemmód ugyanennek a bitnek 0-ra állításával kapcsolható ki. Ezt a következő utasítás végzi el:

```
POKE 53265, PEEK(53265) AND 191
```

Programozható karakterek

A karakter memória helyének kiválasztásakor már szó volt arról, hogy a karakter memória a RAM-ban is lehet. Ebben az esetben a video-chip a RAM-ban tárolt információt tekinti a karakterek alakjának és annak megfelelően írja ki a képernyőre az információt. A karakter memória helyének megváltoztatása előtt természetesen az új helyre a karakterek általunk kívánt alakját kell betölteni. Ezt a legegyszerűbben úgy érhetjük el, hogy a karakter ROM-ot átmásoljuk a RAM-ba, és a POKE utasítás segítségével néhány karakter alakját módosítjuk. Ezt végzi el a következő példaprogram:

```
 5 PRINT CHR$(142) :REM KIS BETU/NAGY BETU
10 POKE 52,48:POKE 56,49 :CLR: REM MEMORIA HELYFOGLALAS
20 POKE 56334,PEEK(56334) AND 254
30 POKE 1, PEEK(1) AND 251
40 FOR I=0 TO 511: POKE I+12288,PEEK(I+53248):NEXT
50 POKE 1,PEEK(1) OR 4
60 POKE 56334,PEEK(56334) OR 1
70 END
```

A példában néhány igen fontos programozói fogás van, amely nélkül a RAM-ban nem használhatnánk a karaktereinket. Ha például a karakter készletet a 12288 ($3000) címtől kezdve helyezzük el, akkor gondoskodni kell arról, hogy a BASIC interpreter 'tudja', hogy a BASIC munkaterület csak eddig tart. Erről gondoskodik a 10-es sorszámú utasítás (részleteket a 3. fejezetben!).

Említettük már, hogy az I/O regiszterek és a karakter ROM ugyanazon a helyen van. Ha tehát a karakter ROM a memóriában van, I/O műveleteket nem végezhetünk. A 20-as sorszámú utasítás ezért letiltja a maszkolható megszakításokat.

A fenti program lefuttatása után hajtsuk végre az alábbi utasítást:

```
POKE 53272,(PEEK(53272) AND 240)+ 12
```

Ugye semmi sem történik? Szinte semmi. A Commodore 64 most a karakter információt a RAM-ból kapja a ROM helyett. De mivel pontosan átmásoltuk a karaktereket a ROM-ból, semmi különbség nem látható... egyelőre.

Most könnyen megváltoztathatjuk a karaktereket. Hajtsuk végre a

```
FOR I = 12288 TO 12288+7:POKE I, 255 - PEEK(I) : NEXT
```

utasítás-sort. Ezzel egy inverz @ jelet hozunk létre! A továbbiakban, ha a @ billentyűt megnyomjuk, egy inverz @ jelet kapunk a képernyőn. (Miért?) A program újbóli futtatása az inverz karakterek inverzét képzi, azaz visszakapjuk az eredeti karaktert.

A képernyő kódok táblázatában megtalálható, hogy a karakterek hol vannak a RAM-ban. Ne feledjük, hogy minden karakter nyolc byte-nyi helyet foglal el. Íme néhány példa:

Karakter	Képernyő kód	Kezdő cím a RAM-ban
@	0	12288
A	1	12296
!	33	12552
>	62	12784

A byte-ok mintája közvetlenül megfelel a karakter képének. Ha elrendezzük a 8 byte-ot úgy, hogy egyiket a másik fölé helyezzük és 8 bináris számjegyként kiírjuk mindegyik byte-ot, akkor az egy 8*8-as mátrixot hoz létre, aml pontosan úgy néz ki, mint a karakterek. Ha egy bit 1, akkor azon a helyen bekapcsolt pont lesz. Ha egy bit nulla, akkor azon a helyen nem lesz bekapcsolt pont.

	76543210	BINARISAN	DECIMALISAN
0	****	00111100	60
1	* *	01000010	66
2	* * * *	10100101	165
3	* *	10000001	129
4	* * * *	10100101	165
5	* ** *	10011001	153
6	* *	01000010	66
7	****	00111100	60

Ha karakter-generáló segédprogramot nem használunk, akkor saját karaktereket legegyszerűbben a fenti négyzetháló kitöltésével kaphatunk. A példán egy mosolygó arc szerepel. A 'bekapcsolt' (itt *-gal jelölt) pontok alapján elkészíthetjük az egyes byte-okat, először bináris, majd decimális alakban. A karakterkészlet módosítását például a következő programmal végezhetjük el:

```
10 FOR I = 12448 TO 12455 : READ A: POKE I,A:NEXT
20 DATA 60,66,165,129,165,153,66,60
```

Ha az előző (karaktereket a ROM-ból RAM-ba másoló) program futtatása, illetve a karakter-memória átkapcsolása **után** a fenti programot is lefuttajuk, a PRINT "<CLR>TTTT" utasítás a képernyő tetejére négy ilyen új 'karaktert' helyez.

A következő példa összefoglalja az eddig mondottakat, átmásol a ROM-ból 64 karaktert, néhányat módosít, és lehetővé teszi az új karakterek kipróbálását.

Saját karakterek készítésénél a karakterek használata előtt mindenképp meg kell nézni, hogyan is 'fest' a karakter. Célszerű két pont vastagságú vonalakból felépíteni, mert ezek már jól látszanak a képernyőn.

```
 10 REM **********************
 20 REM *                    *
 30 REM * PROGRAMOZHATO KARAKTEREK  *
 40 REM *                    *
 50 REM **********************
 60 :
 70 POKE 56334,PEEK(56334) AND 254
 80 :
 90 REM KARAKTERKESZLET BEKAPCSOLASA
100 POKE 1,PEEK(1) AND 251
110 :
120 REM A ROM-BOL ATMASOL 63 KARAKTERT
130 FOR I=0 TO 63
140 REM 8-BYTONKENT MASOL EGY KARAKTERT
150 FOR J=0 TO 7
160 REM EGY BYTE MASOLASA
170 POKE 12288+I*8+J,PEEK(53248+I*8+J)
180 NEXT J,I
190 :
200 REM AZ I/O VISSZAKAPCSOLASA ES
210 REM MEGSZAKITASOK ENGEDELYEZES
220 POKE 1,PEEK(1) OR 4
230 POKE 56334,PEEK(56334) OR 1
240 REM A KARAKTERMUTATO ALLITASA
250 POKE 53272,(PEEK(53272)AND 240)+12
260 :
270 REM AZ UTOLSO 4 KARAKTER MODOSITASA
280 FOR CHAR=60 TO 63
290 FOR BYTE=0 TO 7
300 READ NUMBER : REM BEOLVAS 1 BYTE-T
310 POKE 12288+(8*CHAR)+BYTE,NUMBER
320 NEXT BYTE,CHAR
330 :
340 POKE 53280,15:POKE 53281,15
350 A$=CHR$(60)+CHR$(61)
360 B$=CHR$(62)+CHR$(63)
370 PRINT"<CLR><11 CRSR LE>";
380 PRINT SPC(15);"<CTRL-1>"A$"<CTRL-8>  <CTRL-7>"A$"<CTRL-8>  <CTRL-
5>"A$"<CTRL-8>"390 PRINT SPC(15);"<CTRL-1>"B$"<CTRL-8>  <CTRL-7>"B$"<CTRL-8>
<CTRL-5>"B$"<CTRL-8>"400 GET A$:IF A$="" THEN 400
410 :
420 POKE 53272,21: REM AZ EREDETI KARAKTEREK
430 DATA 4,6,7,5,7,7,3,3
440 DATA 32,96,224,160,224,224,192,192
459 DATA 7,7,7,31,31,95,143,127
460 DATA 224,224,224,248,248,248,240,224
470 END
```

Gépi kódú rutinok felhasználása

Külön szólunk arról, hogyan lehet a karakter üzemmódot gépi kódú rutinokkal támogatni. Karakter üzemmód esetén ugyan ez nem annyira lényeges, mint például bittérképes üzemmódban vagy 'sprite'-ok esetén. Az utóbbi két esetben az elfogadható működési sebesség csak gépi kódú alprogramok segítségével érhető el. Azonban karakter üzemmódban is vannak olyan fogások, amelyek mozgalmasabbá, színesebbé teszik a képernyőn megjelenő szöveget. A példák megértését megkönnyíti, ha tudjuk, hogy a képernyő 8*8-as pontrácsát vezérlő byte-oknak egyes bitjei mit is jelentenek.

Standard karakter üzemmód esetén:

7.bit	0 = normál 1 = inverz
6.bit	0 = normál 1 = siftelt (emelt)
0.-5.bit	tetszőleges karakter kódja.

A képernyőn egy karakter (vagy jel) inverzére egyszerűen átváltoztatható a 7. bit magasra (1) állításával. Gépi kódban ez az EOR #$80 utasítással érhető el. Ha a karakterkészletből a 'kis betűk/nagy betűk' készletet választottuk ki, akkor a 6. bit magasra (1) állításával elérhetjük, hogy a kis betűk nagyra változzanak. Ha a karakter ROM-ból a 'nagy betűk/grafikus jelek' készletet választottuk ki, ez a nagy betűket grafikus jelekre cseréli. Gépi kódban ez az EOR #$40 utasítással hajtható végre.

Példák:

A PRINT utasítás kétféleképpen is helyettesíthető gépi kódú rutinnal. Az első esetben egyszerűen a képernyő-memóriába helyezzük a megfelelő kódokat. Ezzel együtt a szín-memória értékét is be kell állitanunk, hogy a karakterek láthatóak is legyenek:

```
033C           10     *=$033C
033C A2 00     20     LDX #$00
033E 8A        30 LP  TXA
033F 9D 00 04  40     STA $0400,X ; E=VILAGOSKEK
0342 A9 0E     50     LDA #$0E
0344 9D 00 D8  60     STA $D800,X
0347 E8        70     INX
0348 D0 F4     80     BNE LP
034A 60        90     RTS
034B          100     .END
```

Ha a fenti programot betöltöttük a $033C (828) címtől kezdődően, akkor a SYS 828 utasítás segítségével végrehajthatjuk. A program a képernyő tetején mind a 256 karaktert kiírja.

A ?"<HOME> HOGY VAGY" utasítás hatását a következő gépi kódú rutin segítségével érhetjük el:

```
033C            100         *=$033C
033C A2 09      110         LDX #$09
033E BD 4D 03   120 LP      LDA SZOVEG,X
0341 9D 00 04   130         STA $0400,X
0344 A9 0E      132         LDA #$E
0346 9D 00 D8   134         STA $D800,X
0349 CA         140         DEX
034A D0 F2      150         BNE LP
034C 60         155         RTS
034D 00 08 0F   160 SZOVEG  .BYTE $0,$8,$F,$7,$19,$20,$16,$1,$7,$19
0357            170         .END
```

Egy másik lehetőség, hogy a CHROUT KERNAL rutint használjuk. Az LDA #szám/ JSR CHROUT hatása ekvivalens a PRINT CHR$(szám) BASIC utasításéval. Ennek megfelelően a CHROUT a képernyő-szerkesztő karaktereit is kezeli. A fenti program most így alakul:

```
033C             5          *=$033C
FFD2            10 CHROUT   =$FFD2
033C A9 93      20          LDA #$93 ; <HOME>
033E 20 D2 FF   30          JSR CHROUT
0341 A2 00      40          LDX #$00
0343 BD 4F 03   50 LP       LDA SZOVEG,X
0346 20 D2 FF   60          JSR CHROUT
0349 E8         70          INX
034A E0 0B      80          CPX #$0B
034C D0 F5      90          BNE LP
034E 60         100         RTS
034F 20 48 4F   110 SZOVEG  .TEXT "HOGY VAGY"
0359            120         .END
```

Gyakran van szükség a képernyőn megjelenő szöveg villogtatására. Színes display esetén ez a színek állításával is megoldható, fekete-fehér képernyőn hasonló hatás csak a karakterek inverzének képzésével érhető el. Következő két példánk a képernyő első sorának 'villogtatását' végzi el:

```
033C            10      *=$033C
033C A2 28      20      LDX #$28
033E BD FF 03   30 LP   LDA $03FF,X
0341 49 80      40      EOR #$80
0343 9D FF 03   50      STA $03FF,X
0346 CA         60      DEX
0347 D0 F5      70      BNE LP
0349 60         80      RTS
034A            90      .END
```

```
033C             10       *=$033C
033C A0 03       15       LDY #$03
033E A2 28       20 LP1   LDX #$28
0340 B9 4D 03    30 LP2   LDA SZIN,Y
0343 9D FF D7    40       STA $D7FF,X
0346 CA          50       DEX
0347 D0 F7       60       BNE LP2
0349 88          70       DEY
```

```
034A D0 F2      80      BNE LP1
034C 60         90      RTS
034D 04 05 0E 100 SZIN .BYTE 4,5,14
0350          110      .END
```

Az első program esetében a FOR I=1 TO 6:SYS 828 : NEXT I parancs hatására a képernyő első sora háromszor villan. A második program villanási 'technikája' ettől eltér. A SYS 828 parancs hatására az első sor szövege gyors egymásutánban zöld, sárga, piros, végül újból világoskék lesz.

A fenti rutinokat a hardver megszakítási rendszerrel összekapcsolva elérhetjük, hogy pl. a POKE 828, 1 utasítás hatására a képernyő első sora elkezdjen villogni, – miközben a program fut tovább , – s a POKE 828, 0 hatására abbahagyja. Egy ilyen programot az alábbiakban mutatunk be.

A 'villogtató' program a SYS utasítás segítségével aktivizálható. Az aktivizáló rész a hardver megszakító rutin címét, ami a $314-$315 címeken található, elmenti, helyére pedig az UJCIM alatt található 'villogtató' rutin címét helyezi. Hogy az áttöltés közben ne történjék semmi baj, a SEI utasítással letiltjuk a maszkolható megszakításokat, majd a végén a CLI utasítással újból engedélyezzük. A rutin lefutása után, valahányszor egy megszakítás generálódik, a vezérlés az UJCIM címre adódik át, amelyik ellenőrzi a $828 című byte tartalmát. Ha az 0, akkor nem történik semmi, ha ettől eltérő számot talál, akkor az első sor feleannyit villan. A félannyi villanásra azért van szükség, hogy a POKE 828,0 után a karakterek eredeti, ne pedig inverz alakjukban maradjanak a képernyőn.

```
100         ;
105         ; VILLOGTATO RUTIN
110         ;
115         ; ATIVIZALASA: SYS 831
120         ; HIVASA: POKE 828,2*N
125         ; ( N-SZER VILLAN )
130         ; GYAKORISAG: POKE 830,N
135         ; ( KIS N GYORSABB )
140         ;
145         *=$033C
150 TENY    .BYTE $00 ; VILLANASOK SZAMA
155 DELV    .BYTE $0A
160 DELC    .BYTE $0A ; GYAKORISAG
165         SEI
170         ;
175         LDA $0314;HARDVER MEGSZAKITO
180         STA VISSZA+1; REGI CIMEK
185         LDA #$58; ELMENTESE
190         STA $0314
195         LDA $0315
200         STA VISSZA+2
205         LDA #$03
210         STA $0315
215         CLI
220         RTS
225         ;
230 UJCIM   LDA TENY ; HA TENY=0,
```

```
235         BEQ VISSZA ; AKKOR NINCS VILLANAS
240         DEC DELV ; KITARTASI IDO CSOKKENTESE
245         LDA DELV ; HA NEM 0,
250         BNE VISSZA ; AKKOR NINCS VILLANAS
255         DEC TENY ; ISMETLESI TENYEZO
260         ;          CSOKKENTESE
265         LDA DELC
270         STA DELV ; GYAKORISAG UJRATOLTESE
275         JSR VILLOG
280 VISSZA  JMP $FFFF
 85         ;
290 VILLOG  LDX #$28 ; VILLANAS
295 LP2     LDA $033FF,X
300         EOR #$80
305         STA $03FF,X
310         DEX
315         BNE LP2
320         RTS
325         .END
```

7.3 Bittérképes üzemmód

A video-chip másik alapvető üzemmódja az, amikor a 320*200-as képernyő minden pontja a külön-külön ki-, vagy bekapcsolható a memória megfelelő bitjének alacsonyra (0) vagy magasra (1) állításával. A bittérképes üzemmódot éppen ezért **nagyfelbontású üzemmódnak** is szokás hívni, hiszen karakter üzemmódban csak a rögzített (vagy magunk által programozott) grafikus jeleket használhatjuk. A bittérképes üzemmód hátránya, hogy közel 8 Kbyte a helyfoglalása a memóriában (320*200)8).

Általában a nagyfelbontású képernyőt sok rövid, egyszerű rutin ismételt meghívásával állítjuk elő. Sajnos meglehetősen lassú, ha ezeket a rutinokat BASIC-ben írtuk. A rövid, gyakran használt rutinokat gépi kódban célszerű megírni. A megoldás vagy az, hogy programjainkat teljesen gépi kódban írjuk, vagy ilyen rutinokat hívunk a SYS parancs felhasználásával.

A másik – egyszerűbb – megoldás, ha a C-64 BASIC valamelyik, grafikát is kezelő kiterjesztését használjuk. Ilyen a SUPERGRAFIK nevű program, aminek a használatát a fejezet végén ismertetjük.

A Commodore 64-en a bittérképezésnek két lehetséges típusát implementálták. Ezek a következők:

1. Standard (nagy felbontású) bittérképes üzemmód (320*200 pontos felbontás)
2. Többszínű bittérképes üzemmód (160*200 pontos felbontás).

Mindegyik nagyon hasonlít a hasonló nevű karakter üzemmódra: a standard-nek nagyobb ugyan a felbontása, de kevesebb színnel használhatjuk. Másrészről a többszínű bittérképes üzemmód a vízszintes felbontást felére csökkenti, de növeli a felhasználható színek számát.

Standard, nagy felbontású bittérképes üzemmód

A standard bittérképes üzemmód vízszintesen 320, függőlegesen 200 pontos felbontást ad, minden egyes 8*8-as részben két szín közötti választás lehetőségével. A bittérképes üzemmód a VIC-II chip 53265 ($D011) címen található regiszterének segítségével kapcsolható be. Ha a szóban forgó regiszter 5. bitjét magasra állítjuk, a videochip bittérképes üzemmódban kezd el dolgozni. Ezt legegyszerűbben a következő BASIC utasítással érhetjük el:

```
POKE 53265, PEEK(53265) OR 32
```

Az üzemmód kikapcsolása ugyanennek a bitnek alacsonyra (0) állításával érhető el:

```
POKE 53265, PEEK(53265) AND 223
```

Bit térképes üzemmód esetén a szín-memória és a képernyő memória szerepe megváltozik. A **képernyő-memória** a bittérkép 8*8-as részeinek (az egyes karakter helyeknek) a színét határozza meg. Pontosabban, a **felső négy bit** a 8*8-as karakterhely **bekapcsolt**, az **alsó négy bit** a **kikapcsolt** pontjai színét jelenti.

A POKE 1024, 18 utasítás hatására (18=$12) a képernyő bal felső sarkában levő 8*8-as pontrács elemei – attól függően, melyik van ki-, vagy melyik van bekapcsolva – csak fehér (1) és piros (2) színűek lehetnek.

Bittérképes üzemmód esetén a képernyő-memória tehát a színeket határozza meg. A bittérkép, amelyik valójában megadja a képernyő tartalmát, a memória 8192 egymás utáni byte-jából áll. A bittérkép első byte-jának címét ugyanúgy kell megadni, mint a karakter-memória címét.

A fentiek alapján olyan bittérképes üzemmódra, amelyiknek bittérképe a 8192-es címen kezdődik, a következőképpen térhetünk át:

```
10 BASE=2*4096:POKE 53272,PEEK(53272) OR 8
20 REM A BIT TERKEP 8196-ON KEZDODIK
30 POKE 53265,PEEK(53265) OR 32
40 REM BITTERKEPES UZEMMODRA VALO ATTERES
```

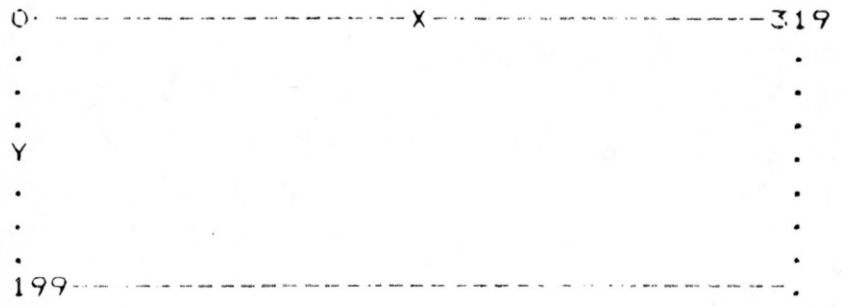
A program futtatása után furcsa képernyőt kapunk, elfelejtettük ugyanis a képernyőt 'törölni'. Ezt például a következő BASIC programmal végezhetjük el:

```
50 FOR I=BASE TO BASE+7999:POKE I,0:NEXT
60 REM A BIT TERKEP TORLESE
70 FOR I=1024 TO 2023:POKE I,3:NEXT
80 REM A BITTERKEP SZINENEK BEALLITASA
```

Ez gépi kódú rutinnal csak egy pillanat!

A bittérkép használata

A pontok ki-, illetve bekapcsolásához tudnunk kell, hogy hogyan lehet megtalálni a megfelelő bitet a memóriában. Ennek kiszámolásához a 320x200-as pontrácsot a következő koordináta rendszerben helyezzük el:



A pont függőleges és vízszintes helyzetének meghatározásához az X és Y betűket fogjuk használni. Az X=0 és Y=0 pont tehát a képernyő bal felső sarkában van. Minél nagyobb az X érték, a pont annál inkább jobbra, és minél nagyobb az Y értéke, annál inkább lejjebb van.

Az egyes pontoknak megfelelő bitek sajnos nem sorfolytonosan vannak a memóriában elhelyezve, hanem karakterhelyenként. Ezen belül pedig sorfolytonosan (úgy, ahogy az egyes karaktereket definiálhatjuk). Így a bittérkép 2. byte-ja a képernyő 2. sora első 8 pontjának állapotát határozza meg. A pontok koordinátái: (0,1)...,(7,1).)

Az X, Y koordinátájú pont ki- vagy bekapcsolásához pontosan tudni kell, hogy a bittérkép hányadik byte-jának melyik bitje felel meg az (X,Y) pontnak. Ehhez először meghatározzuk, melyik karakterhelyen áll a pont:

```
OSZLOP = INT(X/8) : SOR = INT(Y/8)
```

Következő lépés, hogy meghatározzuk, hogy az adott karakterhely hányadik sorában áll:

```
SRKAR = Y - INT(Y/8) * 8
      = Y AND 7
```

Az oszlopszám:

```
OSZKAR = X - INT(X/8) * 8
       = X AND 7
```

Ez alapján a byte és a bit már adódik:

```
BIT = 7 - OSZKAR = 7 - X AND 7.
BYTE = 320*SOR + 8*OSZLOP + SRKAR + BASE
```

Az (X,Y) pont bekapcsolása a következő utasítással történhet:

```
POKE BYTE,PEEK(BYTE) OR 2↑BIT
```

A következőkben egy egyszerű példát mutatunk be a bittérképes üzemmód használatára. Az alábbi program a képernyőre egy szinuszgörbét rajzol:

```
10 BASE=2*4096:POKE 53272,PEEK(53272) OR 8
20 REM A BIT TERKEP 8196-ON KEZDODIK
30 POKE 53265,PEEK(53265) OR 32
40 REM BIT TERKEPES UZEMMODRA VALO ATTERES
50 FOR I=BASE TO BASE+7999:POKE I,0:NEXT
60 REM A BIT TERKEP TORLESE
70 FOR I=1024 TO 2023:POKE I,3:NEXT
80 REM A BIT TERKEP SZINENEK BEALLITASA
100 FOR I=0 TO 8*<pi> STEP 0.05
110 X=160+SIN(I)*I/8/<pi>*159
130 Y=100+COS(I)*I/8/<pi>*99
160 CH=INT(X/8): RO=INT(Y/8)
170 LN= Y AND 7
180 BY=BASE+RO*320+8*CH+LN
190 BI=7-(X AND 7)
200 POKE BY,PEEK(BY) OR (2↑BI)
210 NEXT I
220 GOTO 220
```

Többszínű bittérképes üzemmód

Ugyanúgy, mint a többszínű üzemmódú karakterek, a többszínű bittérképes üzemmód is lehetővé teszi négy különböző szín használatát a bittérkép minden egyes 8x8-as részén. Mint a többszínű karakteres üzemmódban, itt is felére csökken a vízszintes felbontás (320 pontról 160 pontra).

A többszínű bittérképes üzemmód egyrészt a bittérképes üzemmód, másrészt a többszínű üzemmód beállításával érhető el:

```
POKE 53265,PEEK(53265) OR 32
POKE 53270,PEEK(53270) OR 16
```

A kikapcsolás ezzel analóg módon történik:

```
POKE 53265,PEEK(53265) AND 223
POKE 53270,PEEK(53270) AND 239
```

Ha egy programon belül gyakran váltunk át egyik grafikus üzemmódból a másikba, akkor célszerű a fenti eljárásokat alprogramként megírni és GOSUB-bal hívni!

Ugyanúgy, mint a többszínű karakteres üzemmód esetén, itt is a bittérkép két egymás utáni bitje határozza meg a neki megfelelő – két egymás melletti – pont színét:

Bitek	Szín
00	Háttér szín #0
01	A képernyő memória felső négy bitje
10	A képernyő memória alsó négy bitje
11	A szín-memória alsó négy bitje

A többszínű bittérképes üzemmód a memória 8K-s részét használja fel a bittérképre. A többszínű bittérképes üzemmódhoz a következő színeket választhatjuk: (1) háttér szín #0, (2) a szín memória és (3) a normál képernyő alsó, illetve felső félbyte-ja által meghatározott szín.

7.4 A képenyő görgetése

A fenti cím a scroll (up, down stb.) kifejezéssel azonos tartalmat fejez ki: a képernyőt valamilyen irányban soronként (oszloponként) eltoljuk. Soron itt nem egy karaktersor, hanem egyetlen rasztersor értendő. Ezzel az eljárással tetszőleges információt egyenletesen vihetünk fel a képernyőre. A képernyő görgetését a video-chip azon tulajdonsága teszi lehetővé, hogy a képernyő tartalma 0-7 raszter sorral (0-7 raszter oszloppal) eltolva jeleníthető meg.

A képernyő görgetéséhez a következő lépésekre van szükség:

(i) A görgetés irányától függően a képernyő valamelyik sorát vagy oszlopát le kell takarni. Ez alól fognak ugyanis az új sorok a képernyőre 'rágörögni'.
A 38 oszlopos üzemmódhoz a video-chip 53270 ($D016) címen levő regiszterének 3. bitjét használjuk. Ha ez a bit magas, 40 oszlopos üzemmódban dolgozik a video-chip, ha alacsony, akkor 38 oszloposban. A bit ki-, illetve bekapcsolása a szokásos módon végezhető el.
A 24 soros üzemmódhoz az 53265-ös címen levő ($D011) regiszter 3. bitjét kell alacsonyra állítani.

(ii) Az eltolás mértékét a fenti regiszterek 2.-0. bitjei határozzák meg, amelyek egy 0 és 7 közötti számot adnak. A képernyő tartalma X, illetve Y irányban ennyivel eltolódik. Ha a képernyőt az (X,Y) vektorral szeretnénk eltolni, akkor a következő utasításpárt kell kiadnunk:

```
POKE 53270,(PEEK(53270) AND 248)+X
POKE 53265,(PEEK(53265) AND 248)+Y.
```

(iii) X és/vagy Y értékét a 0-7 intervallumban változtatva egy karakterhelynyi részen tudjuk görgetni a képernyőt.

(iv) A (iii) végrehajtása után a képernyő-memóriát egy sorral vagy oszloppal a megfelelő irányban el kell tolnunk és ezzel egyidőben az X és Y megfelelő értékét újra be kell állítani. Ezt kellő gyorsasággal csak gépi kódú rutinnal tudjuk elvégezni. Egyetlen kivétel a BASIC-ben, ha az utolsó sor is betelt és újabb PRINT utasítást adunk ki. Ennek hatására az egész képernyő egy (karakter) sorral feljebb tolódik.

(v) A (iii) és (iv) alatti eljárást kell ismételni.

Példa (a képernyő görgetése felfelé):

```
 10 POKE 53265,PEEK(53265) AND 247
 20 REM 24 SOROS UZEMMOD
 30 PRINTCHR$(147)
 40 REM A KEPERNYO TORLESE
 50 FOR X=1 TO 24 : PRINT CHR$(17);:NEXT
 60 REM A KURZOR UTOLSO SORBA VALO ALLITASA
 70 POKE 53265,(PEEK(53265) AND 248)+7:PRINT
 80 PRINT" HOGY VAGY?"
 90 FOR R=6 TO 0 STEP -1
110 POKE 53265,(PEEK(53265) AND 248)+R
120 FOR X=1 TO 50:NEXT:NEXT
130 GOTO 80
```

7.5 Egyéb grafikus lehetőségek

A képernyő kikapcsolása

A video-chip 53265 ($D011) című regiszterének 4. bitje vezérli a video-chip működését. Amikor ez a bit magas (1), a video-chip normálisan működik. Ha ezt a bitet 0-ra állítjuk, a VIC-II chip felfüggeszti működését, s a képernyő a keret színére változik (és üres lesz). Semmilyen információ nem vész el, csupán nem lesz kijelezve. Ezt a hatást tapasztaljuk például a kazettát használó utasítások végrehajtásánál. A képernyő 'elsötétítését', illetve visszakapcsolását a következő utasítással érhetjük el:

```
POKE 53265,PEEK(53265)AND 239:REM SOTET
POKE 53265,PEEK(53265)OR 16:REM NORMAL
```

A képernyő elsötétítése valamelyest felgyorsítja a processzor működését, és így a BASIC programok végrehajtását. Hosszabb számítások idejére célszerű elsötétíteni a képernyőt.

Raszter regiszter

A raszter regiszter a VIC-II chipben az 53266 ($D012) címen található. A raszter regiszter kettős célú regiszter. Amikor ezt a regisztert olvassuk, akkor az az aktuális raszter helyzet legalacsonyabb 8 bitjét adja. A legnagyobb helyi értékű bit az 53265 ($D011) cím 7. bitje. A raszter regiszter felhasználható időzített változtatások végrehajtására a képernyőn, és így megszabadulhatunk a képernyő nem kívánt villódzásától. A változtatásokat a képernyőn akkor kell végrehajtani, amikor a raszter a nem látható display területen van, azaz, amikor a raszter számláló nem 51 és 251 közé esik. Amikor az aktuális raszter érték ugyanaz lesz, mint a raszter regiszterbe írt szám, akkor a VIC-II chip program megszakító regiszterének megfelelő bitje 1-re állítódik. Ha a raszter szerinti megszakítás engedélyezve van, akkor a programmegszakítás is generálódik.

Program megszakító állapot regiszter

A program megszakító állapot regiszter bármilyen megszakító eszköz állapotát jelzi. A program megszakító állapot regiszter az 53273 ($D019) címen található.

	Bit	Leírás
IRST	0	Amikor az aktuális raszter érték = a tárolt raszter számmal
IMDC	1	Sprite-háttér ütközés esetén (csak az első)
IMMC	2	Sprite-sprite ütközés esetén (csak az első)
ILP	3	Fényceruza negatív átmenete
IRQ	7	Tetszőleges megszakítás esetén

A program megszakító állapot regiszter nem írható és olvasás (PEEK, LD) után nem törlődik. Ez lehetővé teszi a megszakítások ellenőrzését. A megszakításokat a kiolvasott érték visszaírásával lehet törölni.

Ha a fenti leírásnak megfelelő esemény bekövetkezik, a megfelelő bit magasra (1) állítódik, és ha a megszakítást kérő eszközre engedélyeztük a megszakítást, akkor a video-chip egy maszkolható megszakítást generál.

A program megszakítását engedélyező bitek az 53274 ($D01A) regiszterben találhatók. Ennek a byte-nak az első négy bitje (0. - 3. bitek !) rendre az előző táblázatban leírt eseményeknek felel meg. Ha valamelyik bitet magasra állítottuk, az a megszakítás engedélyezését jelenti. Ha a POKE 53274,3 utasítást (vagy gépi kódú megfelelőjét) hajtottuk végre, akkor azt követően valahányszor egy sprite egy másik sprite-tal vagy a háttérrel ütközik, egy maszkolható megszakítást generál a video-chip. A program megszakító állapot regiszter tartalmának ellenőrzésével eldönthetjük, hogy melyik eset is következett be.

Ez a hatékony programmegszakítási rendszer lehetővé teszi egy képernyőn a különféle üzemmódok keverését. Például a képernyő felső fele lehet bittérképes, másik fele lehet karakteres üzemmódban. A hatás egyszerűen úgy érhető el, hogy a raszter szerinti megszakításokat megfelelően kezeljük. A raszter számlálót a képernyő tetejére állítjuk, s amikor a megszakítás bekövetkezik, a raszter regisztert a képernyő közepére állítjuk, illetve bit térképes üzemmódba térünk át. Ha a képernyő közepén következik be a megszakítás, akkor a raszter számlálót visszaállítjuk a képernyő tetejére, és áttérünk karakteres üzemmódra. A játékprogramok gyakran használják ezt a lehetőséget, amikor a képernyő alját az eredmény kijelzésére karakteres üzemmódban használják. Sajnos a BASIC nem elég gyors ahhoz, hogy a megszakításokat kezelni tudja. Ha a video-chip megszakító rendszerét akarjuk használni, akkor gépi kódú rutinokat kell írnunk.

Javasolt színkombinációk

A TV készülékek szín-felbontó képessége korlátozott. Bizonyos színek egymás mellé helyezése érzékelhető színhatást eredményez, megint más színek egymás mellé helyezése torz színeket ad. A következő táblázat összefoglalja, milyen színhatásokat érdemes használni, illetve elkerülni.

üres kör = kiváló
tele kör = jó
x = gyenge

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	x	○	x	○	○	●	x	○	○	x	○	○	○	○	○	○
1	○	x	○	x	○	○	○	x	●	○	●	○	○	x	○	○
2	x	○	x	x	●	x	x	○	○	x	○	x	x	x	x	●
3	○	x	x	x	x	●	○	x	x	x	x	●	x	x	●	x
4	○	●	x	x	x	x	x	x	x	x	x	x	x	x	x	●
5	○	●	x	●	x	x	x	x	x	x	x	●	x	○	x	●
6	●	○	x	○	x	x	x	x	x	x	x	x	x	●	○	○
7	○	x	○	x	x	x	●	○	●	○	●	○	○	x	x	x
8	●	○	○	x	x	x	x	○	x	○	x	x	x	x	x	●
9	x	○	x	x	x	x	x	○	○	x	○	x	x	x	x	○
10	●	●	○	x	x	x	x	○	x	○	x	x	x	x	x	●
11	○	○	x	●	x	x	x	○	x	x	x	x	○	○	○	○
12	○	○	●	x	x	x	●	x	x	●	x	○	x	x	x	○
13	○	x	x	x	x	○	●	x	x	x	x	○	x	x	x	x
14	○	○	x	○	x	x	○	x	x	x	x	●	x	x	x	●
15	○	○	○	x	●	●	○	x	x	●	●	○	○	x	●	x

7.6 Sprite-ok

Mint a bevezetésben is említettük, a sprite-ok 24*21-pontos képelemek, amelyek a képernyő bármelyik részén megjeleníthetők **függetlenül** attól, hogy a képernyőt milyen üzemmódban és mire használjuk. Hasonlóan a karakterekhez, a 24*21 pont bármelyike be-, illetve kikapcsolt állapotban lehet. A video-chip a sprite alakját megadó információt a memória megadott helyén keresi és ez alapján állítja elő a képernyőn.

A VIC-II chip egyidejűleg 8 sprite-ot tud kezelni. További sprite-ok csak a video-chip megszakítási rendszerének felhasználásával helyezhetők a képernyőre. A sprite-ok legfontosabb tulajdonságai a következők:

1./ Mérete: 24 vízszintes és 21 függőleges pont.
2./ Minden egyes sprite-nak külön színe lehet.
3./ Sprite többszínű üzemmód (hasonló a többszínű karakter üzemmódhoz).
4./ Nagyítás (2-szeres) vízszintes, függőleges vagy mindkét irányban.
5./ Sprite és háttér takarásának megválasztása.
6./ Sprite-ok egymás közti takarásának megválasztása.
7./ Megszakítás generálása sprite-ok ütközésekor.
8./ Megszakítás generálása sprite és háttér ütközésekor.

A sprite fenti képességei teszik egyszerűvé a 'kaland' típusú játékok programozását. Mivel a sprite-okat a hardver támogatja, még az is lehetséges, hogy jó minőségű játékprogramot írjunk BASIC-ben.

A VIC-II chip közvetlenül 8 sprite-ot tud kezelni. Ezek 0-tól 7-ig vannak számozva. Minden egyes sprite-nak külön regiszterei vannak, amelyek a színét, a helyzetét, az üzemmódját határozzák meg. A képernyő memória első 1000 byte-ja utáni 17-24-ik byte-ok a sprite-ok alakját definiáló memóriarészt határozza meg.

Sprite-ok definiálása

A sprite-okat ugyanúgy definiáljuk, mint a programozható karaktereket. Egyetlen különbség, hogy a sprite-ok mérete nagyobb, ezért több byte-ra van szükség. Egy sprite 24*21 = 504 pontot tartalmaz. Így 504/8 = 63 byte-ra van szükség egy sprite alakjának definiálásához. A be- (1), illetve kikapcsolt (0) pontoknak megfelelő biteket sorfolytonosan kell a memóriában elhelyezni. Az egymás utáni byte-ok bitjeit a video-chip a következőképpen használja fel a sprite kialakításához:

BYTE 0	BYTE 1	BYTE 2
BYTE 3	BYTE 4	BYTE 5
BYTE 6	BYTE 7	BYTE 8
...	...	...
...	...	...
...	...	...
BYTE 60	BYTE 61	BYTE 62

Minden egyes sprite esetén külön megadható, hogy a memória melyik 63 byte-ja a sprite definiciója. Az egyszerűség kedvéért ezt a gép 'felkerekíti' 64-re. A sprite definíciója a videochiphez rendelt 16K-s memóriaszelet bármelyik 64-gyel osztható címén kezdődhet. A sprite-ok megadásánál használjuk az úgynevezett **s-lapokat**. Ez azt jelenti, hogy a memória 16K-s szeletét 64 byte-onként felbontjuk. A 0. s-lap a szelet 0-63 című byte-jaiból, a 10. s-lap a 640-703 című byte-okból áll. A sprite alakja helyének definíciójához azt kell megadni, hogy a lehetséges 256 (0-255-ig számozott) s-lap közül melyiken található.

A fenti 8 mutatót (sprite-onként egy) az 1000 byte-os képernyő-memória utáni 17-24-ik byte-ok tartalmazzák. Ezek a 2040-2047 ($07F8-$07FF) című byte-ok (ha nem állítottuk át a képernyő memória helyét). A 2040-es címen levő byte a 0., a 2041-es címen levő byte az 1. stb. sprite-hoz tartozik. Ha tehát a 2040-es címen 14-et találunk, az azt jelenti, hogy a 0. sprite definíciója a 14. s-lapon található (első byte-jának címe: 14*64=896). A sprite definició első byte-jának címe általában a következő képlettel számítható ki:

B=SZELET*16384+SPRITE MUTATÓ*64.

Természetesen az összes s-lap nem használható sprite definícióra. Vannak olyan s-lapok, amelyek ROM-ba esnek, mások byte-jait a program vagy az operációs rendszer használja.

Sprite-ok bekapcsolása

Az 53269 címen levő ($D015) VIC-II vezérlő regiszter sprite kapcsolási regiszter néven ismert. Mindegyik sprite-nak van egy bitje ebben a regiszterben, ami azt vezérli, hogy a sprite be van-e kapcsolva vagy sem. Ha a megfelelő bit 1, akkor a spriteot bekapcsoltuk; ha 0, akkor nem. Például az első sprite bekapcsolásához az szükséges, hogy az 1-es bitet 1-re állítsuk. Ezt a következő POKE utasítás végzi:

```
POKE 53269,PEEK(53269)OR 2
```

Egy általánosabb utasítás a következő:

```
POKE 53269,PEEK(53269)OR 2↑SN
```

ahol az SN a sprite szám (0-tól 7-ig).

Az SN sorszámú sprite-ot a következő utasítással kapcsolhatjuk ki:

```
POKE 53269,PEEK(53269) AND (255-2↑SN)
```

A sprite kikapcsolt állapotban semmiféle hatást nem gyakorol a képernyőre. Bekapcsolása pillanatában a helyzetét meghatározó regiszteknek megfelelően megjelenik a képrnyőn. Ez nem jelenti azt, hogy látható lesz (lásd a sprite-ok helyzetéről szóló részt)!

Sprite-ok színe

A sprite színe a VIC-II chip által generált 16 szín bármelyike lehet. Mindegyik sprite-nak megvan a saját szín regisztere. Ezen regiszterek memória helyei:

Cím	Leírás
53287 $D027	SPRITE#0 SZIN REGISZTER
53288 $D028	SPRITE#1 SZIN REGISZTER
53289 $D029	SPRITE#2 SZIN REGISZTER
53290 $D02A	SPRITE#3 SZIN REGISZTER
53291 $D02B	SPRITE#4 SZIN REGISZTER
53292 $D02C	SPRITE#5 SZIN REGISZTER
53293 $D02D	SPRITE#6 SZIN REGISZTER
53294 $D02E	SPRITE#7 SZIN REGISZTER

A sprite-ban levő mindegyik bekapcsolt (1) pont a sprite szín regiszterben levő színben jelenik meg. A sprite fennmaradó része **áttetsző** lesz, és azt látjuk, ami a sprite mögött van.

Többszínű üzemmód

A többszínű üzemmód lehetővé teszi, hogy négy különböző színű pont legyen mindegyik sprite-ban. Mint a többi többszínű üzemmód esetén, a vízszintes felbontás felére csökken. Más szavakkal, ha sprite többszínű üzemmódban dolgozik (ugyanúgy, mint a többszínű karakter üzemmód esetén), akkor 24 pont helyett 12 pontpár jelenik meg. Mindegyik pontpár színét a két pontnak a memóriában megfelelő bit-pár határozza meg a következő táblázat szerint:

BIT-pár	Leírás
00	ÁTTETSZŐ (ami a képernyőn van)
01	SPRITE TÖBB SZÍN REGISZTER#0 53285, $D025
10	SPRITE SZÍN REGISZTER
11	SPRITE TÖBB SZÍN REGISZTER#1 53286, $D026

Mindegyik sprite-ról külön-külön eldönthetjük,hogy standard vagy többszínű üzemódban használjuk-e. A video-chip 53276 ($D01C) címén levő regiszterének bitjeit használhatjuk a többszínű üzemmód beállítására. Ha az SN-ik bit magas, akkor az SN-ik sprite többszínű üzemmódban jelenik meg, ha alacsony, akkor pedig standard üzemmódban. Az SN. sprite többszínű üzemmódját tehát a következő utasítással kapcsolhatjuk be, illetve ki:

```
POKE 53276,PEEK(53276)OR 2↑SN:REM BE
POKE 53276,PEEK(53276)AND(255-2^SN):REM KI
```

Sprite-ok nagyítása

A VIC-II chip rendelkezik azzal a képeséggel, hogy egy sprite-ot vízszintes, illetve függőleges irányban, vagy egyidejűleg mindkét irányban kétszeresére nagyítsa. Nagyításkor mindegyik pont a sprite-ban kétszer olyan széles vagy kétszer olyan magas lesz, mint volt. A felbontás valójában nem nő, épp csak a sprite lesz nagyobb.

A sprite vízszintes irányú nagyításához a VIC-II chip 53277 ($D01D) című regiszterének megfelelő bitjét be kell kapcsolni (1-re kell állítani). A következő POKE utasítás X irányban nagyítja az SN. sprite-ot:

```
POKE 53277,PEEK(53277)OR 2↑SN
```

Egy sprite X irányban normál méretben való megjelenítéséhez a megfelelő POKE utasítás:

```
POKE 53277,PEEK(53277) AND (255-2↑SN)
```

A sprite függőleges irányú nagyításához a VIC-II 53271 ($D017) című regiszterének megfelelő bitjét be kell kapcsolni (1-re kell állítani). A következő POKE utasítás Y irányban nagyítja az SN. sprite-ot:

```
POKE 53271,PEEK(53271) OR (2↑SN)
```

A függőleges irányú nagyítás hatását a következő utasítás segítségével semlegesítjük:

```
POKE 53271,PEEK(53271) AND (255-2↑SN)
```

Sprite-ok elhelyezése

Legutoljára hagytuk annak a kérdésnek a tisztázását, hogyan helyezhetjük el a sprite-ot a képrnyőn. Ehhez röviden szólni kell arról a 'térről', amelyben a sprite-ok mozognak. A videochip a sprite-okat egy 512*256-os pontrácsban helyezi el. Ennek a pontrácsnak természetesen csak egy rögzített 320*200-as része látható. Ha a sprite azon kívül van, akkor nem, vagy csak részben látható. A video-chip a sprite darabjait modulo 512, illetve modulo 256 helyezi el, ami azt jelenti, hogy ha például a 24*21-es pontrács bal felső pontjának koordinátái (500,200), akkor a sprite egyes részei a 0-11-es oszlopokban jelennek meg (500 + 23 = 11 modulo 512!). Ugyanez a jelenség tapasztalható függőleges irányban is. A 'sprite-tér' nem látható része azt a célt szolgálja, hogy a sprite-ok egyenletesen rámozgathatók, rágörgethetők legyenek a képernyőre.

A fentiek alapján egy sprite helyzetének meghatározásához (ez mindig a sprite bal felső pontjának koordinátáját jelenti!) a következő értékeket kell megadni:

1./ a sprite X koordinátája ($0 < X < 512$)
2./ a sprite Y koordinátája ($0 < Y < 256$).

A sprite X koordinátáját nem lehet egy byte-on megadni. A sprite legnagyobb helyiértékű bitjét (MSB) egy külön regiszter tárolja.

A következő táblázat felsorolja az összes sprite helyzet regiszter címét. A megfelelő programrész használhatja ezeket POKE utasításokon keresztül:

Cím	Leírás
53248 $D000	SPRITE 0 X HELYZET REGISZTER
53249 $D001	SPRITE 0 Y HELYZET REGISZTER
53250 $D002	SPRITE 1 X HELYZET REGISZTER
53251 $D003	SPRITE 1 Y HELYZET REGISZTER
53252 $D004	SPRITE 2 X HELYZET REGISZTER
53253 $D005	SPRITE 2 Y HELYZET REGISZTER
53254 $D006	SPRITE 3 X HELYZET REGISZTER
53255 $D007	SPRITE 3 Y HELYZET REGISZTER
53256 $D008	SPRITE 4 X HELYZET REGISZTER
53257 $D009	SPRITE 4 Y HELYZET REGISZTER
53258 $D00A	SPRITE 5 X HELYZET REGISZTER
53259 $D00B	SPRITE 5 Y HELYZET REGISZTER
53260 $D00C	SPRITE 6 X HELYZET REGISZTER
53261 $D00D	SPRITE 6 Y HELYZET REGISZTER
53262 $D00E	SPRITE 7 X HELYZET REGISZTER
53263 $D00F	SPRITE 7 Y HELYZET REGISZTER
53264 $D010	SPRITE X MSB REGISZTER

A 0. sprite X irányú helyzetét a $D000 byte és a $D010 byte 0. bitje együtt határozza meg. Ha a bit magas ($=1$), akkor a $D000 címen levő byte-hoz még hozzá kell adnunk 256-ot, hogy az X koordinátát megkapjuk. Az SN. sprite X, illetve Y irányú koordinátáit a következőképpen számíthatjuk ki:

```
 5 VIC=53248
10 X=PEEK(VIC+2*SN)+256*(PEEK(VIC+16)AND2↑SN)/2↑SN
20 Y=PEEK(VIC+2*SN+1)
```

Az SN. sprite helyzetét (X,Y)-ra a következő program állitja be:

```
1060 HX=INT(X/256):LX=X AND 255
1070 POKE VIC+2*SN,LX:POKE VIC+2*SN+1,Y
1080 POKE VIC+16,PEEK(VIC+16) OR HX*2↑SN
```

A legkisebb koordinátapár, amikor a teljes 24*21 pontú képelem a látható részben van: (24,50). A legnagyobb koordinátapár, amikor a normál méretű sprite teljes egészében látszik: (320,229).

Sprite-ok nagyítása esetén a nagyítás a sprite bal felső pontjából történik. Így a normál méretű, illetve X és/vagy Y irányba nagyított sprite-ok helyzetét ugyanúgy kell beállítani.

Sprite-ok további tulajdonságai

A video-chip az egyes sprite-ok megjelenítésekor a kisebb sorszámút a nagyobb előtt jeleníti meg. Ez azt jelenti, hogy ha két sprite nem áttetsző részei egybeesnek, akkor a képernyőn a kisebb sorszámú sprite látszik csak. A 24*21-es képelem azon részein, ahol a kisebb sorszámú sprite áttetsző, de a nagyobb sorszámú nem, természetesen a nagyobb sorszámú sprite látszik.

Lehetőségünk van eldönteni, hogy a sprite és a háttér nem áttetsző részeinek egybeesésekor mit lássunk. Erre az 53275 ($D01B) című regiszter szolgál. Ha az SN. bit alacsony (ez a kezdőérték), akkor az SN. sprite látszik a háttér előtt. Ha a bitet magasra (=1) állítjuk, akkor a háttér látszik, a sprite pedig nem.

A video-chip segítségével lehetőségünk van megszakítás generálására abban az esetben, ha két sprite, vagy valamely sprite és a háttér nem áttetsző része egybeesik (ezt hívjuk **ütközés**nek). Sprite-ok közt akkor is történhet ütközés, ha nincsenek a látható részben! Az 53278 ($D01E) címen levő regiszter bitjei magasra állítódnak,ha a megfelelő sprite valamely másik spritetal ütközik. Ugyanez a feladata az 53279 ($D01F) címen levő regiszternek; a megfelelő bit magasra állítódik, ha a megfelelő sprite a háttérrel ütközik.

Mindkét regiszter olyan, hogy olvasás után (PEEK vagy LD) törlődik, értékét a feldolgozásig tehát célszerű tárolni.

Megjegyezzük végül,hogy többszínü üzemmódban a 01 bitkombinációnak megfelelő pontok az ütközés szempontjából áttetszőnek számítanak még akkor is (!), ha a képernyőn az ilyen színű pontok látszanak.

Sprite-ok használata

Sprite-ok használata gépi kódból a legegyszerűbb. BASIC-ben a megfelelő regiszterek értékeinek beállítása a POKE, PEEK utasításpárral történik, ami meglehetősen nehézkes. Bizonyos BASIC bővítések lehetővé teszik a sprite-ok kényelmes definiálását és használatát. Ezekről a BASIC kiterjesztésekről szóló ismertetőben lesz szó. Még ebben az esetben sem biztos azonban, hogy BASIC-ben sikerül a kellő sebességet elérnünk. A sprite-ok definiálására külön segédprogramok is léteznek.

A sprite-okat, ha BASIC-ből használjuk, általában a következő műveletekkel állítjuk elő, illetve használjuk:

a/ a sprite alakját definiáló 63 byte beállítása;
b/ a sprite-mutató értékének beállítása;
c/ standard vagy többszínű üzemmód kiválasztása;
d/ a sprite-hoz tartozó színek, takarási prioritások meghatározása;
e/ a sprite kezdő helyzetének beállítása;
f/ a sprite bekapcsolása;
g/ a fenti értékek folyamatos változtatása, a megszakítások kezelése.

Példák:

Példáinkban elsősorban BASIC programrészeket adunk, amelyek bemutatják, hogy az előbb felsorolt lépéseket hogyan valósíthatjuk meg. Gépi kódú alprogramok elsősorban a sprite-ok mozgatásához szükségesek, a kívánt gyorsaság BASIC-ből ritkán érhető el.

(i) A sprite-ok alakját meghatározó byte-ok kiszámítása igen fáradságos. Első példánk, amelynek listáját a következő oldalon közöljük, a DATA utasítást használja a sprite **alakjának** megadására. A feldolgozó rész kiszámítja a megfelelő byte-okat és elhelyezi őket a memóriában. Ha szükségünk van ezekre a byte-okra, kilistázhatjuk a képernyőn. A program segítségével tetszőleges alakú sprite-ot definiálhatunk, csupán a DATA utasításokban szereplő szövegrészt kell módosítani.

```
 10 INPUT"<CLR>S-LAP=";S
 20 CIM=S*64
 30 GOSUB 100
 40 PRINT"KERI-E A BYTE-OK ERTEKET?"
 50 GET A$:IF A$="" THEN GOTO 50
 60 IF A$<>"I" THEN END
 70 PRINT"<CLR>":FOR I=0 TO 62
 80 PRINTPEEK(CIM+I),
 90 NEXT:END
100 FOR I=0 TO 20:READ A$
110 FOR J=0 TO 2:T=0
120 FOR K=0 TO 7:X=0
130 IF MID$(A$,K+1+8*J,1)="A" THEN X=1
140 T=T+X*2↑(7-K):NEXTK
150 POKE CIM+3*I+J,T:NEXT J:NEXTI
160 RETURN
```

```
170 REM  012345678901234567890123
180 DATA ........................
190 DATA ........................
200 DATA .......AAAAAAAAAA.......
210 DATA ........AAAAAAAA..AA....
220 DATA ........................
230 DATA ..AAAAAAAAAAAAAAAAAAA...
240 DATA ....AAAAAAAAAAAAAAAAAAAA
250 DATA ....AAAAAAAAAAAAAAAAA...
260 DATA ..AAAAAAAAAAAAAAAAAA....
270 DATA .......AAAAAAAAAA......
280 DATA ........AAAAAAA.........
290 DATA ........AAAAAA..........
300 DATA .......AAAAAA...........
310 DATA ......AAAAA.............
320 DATA ......AAAA..............
330 DATA ......AAA...............
340 DATA ........................
350 DATA ........................
360 DATA ........................
370 DATA ........................
380 DATA ........................
390 REM  012345678901234567890123
```

(ii) Az előző programrész csak standard sprite-ok definiálására alkalmas. A DATA utasításban egy tizedespont jelenti, hogy a pont nincs bekapcsolva, az A jel jelenti a bekapcsolt pontokat. Többszínű sprite-ok esetén három különböző színt is használhatunk. Az ennek megfelelő programot úgy kapjuk, hogy az előző programban a valódi feldolgozást végző, 100. sorral kezdődő alprogramot a következőre cseréljük:

```
100 FOR I=0 TO 20:READ A$
110 FOR J=0 TO 2:T=0
120 FOR K=0 TO 3:X=0
130 G$=MID$(A$,K+1+4*J,1)
135 IF G$<>"." THEN X=ASC(G$)-64
138 IF G$="." THEN X=0
140 T=T+X*2↑(6-2*K):NEXT K
150 POKE CIM+3*I+J,T:NEXT J:NEXT I
160 RETURN
```

A DATA utasítások természetesen most már csak 12 hosszú sztringet tartalmazhatnak. Az egyes karakterek a következő színű pontoknak felelnek meg:

pont	áttetsző
A	SPRITE MULTICOLOR# 0
B	a sprite saját színe
C	SPRITE MULTICOLOR# 1

A következő DATA utasítások egy közismert madarat definiálnak:

```
170 REM  012345678901
180 DATA ......AA....
190 DATA .....ABCA...
200 DATA ....AABBBAAA
210 DATA ...AABBBA...
220 DATA .....ABA....
230 DATA .AA..ABA....
240 DATA ABBA.ABA....
250 DATA .ABA.ABA....
260 DATA ..AA.ABA....
270 DATA ...AABBBA...
280 DATA ..ACBCBCBA..
290 DATA .ACBCBCBCBA.
300 DATA .ABCBCBCBCA.
310 DATA ..ABCBCBAA..
320 DATA ...AAAAA....
330 DATA ....A..A....
340 DATA ...A....A...
350 DATA ..A......A..
360 DATA ...A......A.
370 DATA ..AAA....AAA
380 DATA ..A..A..A..A
390 REM  012345678901
```

(iii) Az előző program csak a sprite alakját definiálja; ahhoz, hogy lássuk is, be kell állítanunk a sprite-mutatókat, a sprite színét, helyzetét és végül a video-chip megfelelő bitjének magasra állításával engedélyezni kell a sprite-ot. Ha az előző programban a 30. sort a **30 GOSUB 100:GOSUB 1000** utasításra cseréljük és az alábbi utasításokkal kiegészítjük, akkor a sprite megjelenik a képernyő közepén:

```
1000 PRINT"<CLR>";
1010 POKE 2040,13   :REM SPRITE#0=13-AS SPRITE-LAP
1020 VIC=53248      :REM VIC REGISZTEREI ITT KEZDODNEK
1030 POKE VIC+21,1  :REM SPRITE#0 ENGEDELYEZESE
1040 POKE VIC+39,1  :REM          SZINE
1050 POKE VIC,150   :REM          X KOORDINATA
1060 POKE VIC+1,100 :REM          Y KOORDINATA
1080 POKE VIC+39,1  :REM          SZINE
1090 RETURN
```

(iv) Ha az 1000 sorszámmal kezdődő programot a következő alprogramra cseréljük,

az a sprite-ot a képernyőn keresztül mozgatja:

```
1000 PRINT"<CLR>"
1010 POKE 2040,13  :REM SPRITE#0=13-AS SPRITE-LAP
1020 VIC=53248     :REM VIC REGISZTEREI ITT KEZDODNEK
1030 POKE VIC+21,1 :REM SPRITE#0 ENGEDELYEZESE
1040 POKE VIC+39,1 :REM          SZINE
1045 POKE VIC+1,100
1050 FOR J=0 TO 347
1060 HX=INT(J/256):LX=J-256*HX
1070 POKE VIC,LX:POKE VIC+16,HX:NEXT
1080 RETURN
```

(v) Ha ugyanezt a sprite-ot nagyítva akarjuk a képernyőn keresztül mozgatni, akkor a sprite-ot a képernyő jobb oldaláról kell elindítanunk. Ennek oka a 'sprite-tér' és a látható képernyő egymáshoz való viszonya: a képernyő bal oldalán, a nem látható részben már nincs elég hely.

```
1000 PRINT"<CLR>"
1010 POKE 2040,13 :REM SPRITE#0=13
1020 VIC=53248    :REM VIC REGISZTEREI ITT KEZDODNEK
1030 POKE VIC+21,1:REM SPRITE#0 ENGEDELYEZESE
1040 POKE VIC+39,1:REM          SZINE
1044 POKE VIC+29,1:REM          X IRANYBAN NAGYITAS
1045 POKE VIC+23,1:REM          Y IRANYBAN NAGYITAS
1050 FOR J=488 TO 511:GOSUB 1060:NEXT
1055 FOR J=0 TO 340:GOSUB 1060:NEXT
1058 RETURN
1060 HX=INT(J/256):LX=J-256*HX
1070 POKE VIC,LX:POKE VIC+16,HX:RETURN
```

(vi)

```
100 REM ************************
101 REM *                      *
102 REM *       VADASZAT       *
104 REM *  AZ EGEN ATUSZO LEGGOMBOT KELL  *
105 REM *   A SPACE BILLENTYUVEL LELONI   *
106 REM *                      *
107 REM ************************
108 :
109 REM ********************
110 REM * SPRITE ADATOK BEALLITASA  *
111 REM ********************
112 V=53248:REM VIC CHIP KEZDOCIME
113 REM SPRITEMUTATOK
114 POKE 2042,14:POKE 2043,13: REM SPRITE 2,3
115 REM SPRITE ALAKJANAK BETOLTESE
116 FOR N=0 TO 62:READ Q:POKE 832+N,Q:NEXT
117 FOR N=0 TO 62:READ Q:POKE 896+N,Q:NEXT
118 REM HATTER SZINE,SPRITE-OK SZINE
119 POKE V+33,0:POKE V+41,7:POKE V+42,3
120 :
121 REM *********
122 REM * IDOMERES  *
```

```
123 REM *********
124 TJ=TI:J=0:PRINT"<CLR>"
125 IF TI>TJ+3600 THEN 176
126 :
127 REM ********************
128 REM * KEZDETI ADATOK BEALLITASA *
129 REM ********************
130 REM A RAKETA BEALLITASA
131 IF Y=229 THEN GOTO 135
132 A=RND(1)*60:POKE V+4,A:POKE V+5,229
133 POKE V+16,4:POKE V+6,0
134 REM X,Y IRANYU NAGYITAS, UTKOZES TORLESE
135 POKE V+23,0:POKE V+29,0:POKE V+30,0
136 :
137 REM A LEGGOMB BEALLITASA
138 U=RND(1)*100+45:Y=229:A$=""
139 REM SPRITE-OK ENGEDELYEZESE
140 POKE V+21,12
141 A$="":REM A RAKETA INDITASAT JELZI
142 :
143 REM ************************
144 REM * A JATEKOT SZABALYOZO CIKLUS  *
145 REM ************************
146 FOR X=0 TO 344 STEP 5
147 :
148 REM A LEGGOMB MOZGATASA
149 RX=INT(X/255):LX=X-RX*255
150 POKE V+16,PEEK(V+16) OR (RX*8)
151 POKE V+6,LX
152 POKE V+7,U
153 :
154 IF A$<>"" THEN 159
155 GET A$
156 IF A$="" THEN GOTO 163
157 :
158 REM A RAKETA MOZGATASA
159 Y=Y+(Y>=6)*6
160 POKE V+5,Y
161 :
162 REM TALALAT ELLENORZESE
163 IF PEEK(V+30)=0 THEN GOTO 169
164 REM VILLOGTATAS
165 POKE V+23,255-PEEK(V+23)
166 POKE V+29,255-PEEK(V+29)
167 J=J+1:PRINT "<HOME><CRSR LE>";J
168 :
169 NEXT X
170 :
171 POKE V+21,0
172 GOTO 125:REM UJ JATEK
173 REM ********************
174 REM * A VEGEREDMENY KIIRASA  *
175 REM ********************
176 PRINT"<CLR>       TALALATOK SZAMA=";J
177 INPUT"       AKARSZ MEG JATSZANI";D$
178 IF D$="I" THEN GOTO 124
179 CLR:PRINT"<CLR>":POKE V+21,0
```

```
180 END
181 REM **************
182 REM * A LEGGOMB ADATAI *
183 REM **************
184 DATA 0,127,0,1,255,192,3,255,224,3,227,224
185 DATA 7,217,240,7,223,240,7,217,240,3,231,224
186 DATA 3,255,224,3,255,224,2,255,160,1,127,64
187 DATA 1,62,64,0,156,128,0,156,128,0,73,0,0,73,0
188 REM **************
189 REM * A RAKETA ADATAI  *
190 REM **************
191 DATA 0,62,0,0,62,0,0,62,0,0,28,0
192 DATA 0, 0,0,0, 0,0,0, 0,0,0, 0,0
193 DATA 0, 0,0,0, 0,0,0, 0,0,0, 0,0
194 DATA 0, 0,0,0, 0,0,0, 0,0,0, 0,0
195 DATA 0,24,0,0,24,0,0,24,0,0,24,0
196 DATA 0,24,0,0,24,0,0,60,0,0,60,0
197 DATA 0,126,0
```

7.7 SUPERGRAFIK

Bevezetés

A C-64 grafikus lehetőségeinek ismertetése közben gyakran utaltunk arra, hogy ezeket legegyszerűbben gépi kódú rutinok segítségével használhatjuk fel. Hasonló kényelmes megoldás speciális BASIC bővítéseket alkalmazni – amelyek majdnem annyi lehetőséget biztosítanak, mint a gépi kódú rutinok –, sokkal egyszerűbb programozási technológiával.

A következőkben egy igen népszerű, a DATA BECKER GMBH által forgalmazott, M. Thielker által kidolgozott programcsomagot ismertetünk. A programcsomag elsősorban a mérnöki tervezés igényeit elégíti ki, karakterkészletek definiálására például nem alkalmas.

A SUPERGRAFIK utasításai kétféle képernyőt tudnak kezelni, az úgynevezett kis felbontású (LO-RES) és az úgynevezett nagyfelbontású (HI-RES) képernyőt. Az első esetben valójában standard karakter üzemmódban dolgozik az interpreter, és a grafikus utasításokban szereplő 'pontok' a képernyő 4*4-es pontrácsai lesznek. Így összesen 80*50 pont létezik. Az alacsony felbontású képernyő a képernyő memóriával azonos. A pontok ki-, illetve bekapcsolása a megfelelő grafikus karakterek beírásával történik.

A nagyfelbontású képernyő a bittérképes üzemmódnak felel meg. Ennek felbontóképessége 320*200, a pontok a képernyő legkisebb önállóan megjeleníthető képelemeire utalnak. A pontok ki-, illetve bekapcsolása a bittérkép megfelelő bitjének alacsonyra, illetve magasra állítását jelenti.

Két nagyfelbontású képernyőt használhatunk. Ezek közül mindig az elsőben hajtódnak végre a HI-RES grafikus utasítások. A két képernyő a !TRANS utasítással felcserélhető.

A SUPERGRAFIK **nem** támogatja a többszínű üzemmódokat.

A program betöltése

Amennyiben a SUPERGRAFIK program kazettán áll rendelkezésünkre, a <SHIFT> és a <RUN/STOP> billentyűk együttes lenyomásával, majd a kazettás egység <PLAY> billentyűjének lenyomásával töltsük be, és futtassuk le a főprogramot. Ennek hatására a SUPERGRAFIK rendszer betöltődik, és a rendszer bejelentkezik a következő üzenettel:

```
CMB 64 HI-RES GRAPHICS SOFTWARE REV.F
(C) BY DATA BECKER GMBH
CREATED 19/01/1983
BY M. THIELKER
GRAPHICS READY.
```

Ha a SUPERGRAFIK-ot gyári lemezen sikerült megszereznünk, akkor a töltő programot a LOAD"*",8 parancs segítségével töltsük be, majd a RUN paranccsal futtassuk le. A rendszer ugyanúgy jelentkezik be, mint a kazettáról való töltés esetén.

SUPERGRAFIK utasítások

Valamennyi SUPERGRAFIK utasítás felkiáltójellel (!) kezdődik. A SUPERGRAFIK program – mint minden BASIC beszúrás – a CHRGET rutint írja felül. A SUPERGRAFIK új CHRGET rutinja figyeli a felkiáltójelet (!). Ha ilyent talál – és nem egy másik utasítás végrehajtása közben (ilyen például a PRINT"!") –, akkor nem a ROM-ba tér vissza, hanem elkezdi saját rutinjai végrehajtását. Az új utasítások parancs módban is használhatók. Problémát csak az okoz, hogy például a nagyfelbontású képernyő esetén a billentyűzetről bevitt utasítások szintaxisát nem tudjuk vizuálisan ellenőrizni.

Elöljáróban röviden szólunk az utasítások leírásában használt jelölésekről:

x1, y1	koordinátapár (kezdőpont)
x2, y2	koordinátapár (végpont)
p	paraméter
lf	logikai fileszám (1-255)
d	egységszám (1-15)
dn	csatornaszám
fn	file név

Az x1,x2, illetve y1,y2 koordináták értékei alacsony felbontású képernyő esetén a $0 <= xi <= 79$ illetve a $0 <= yi <= 49$ intervallumba kell, hogy essenek.

Hasonlóan, nagy felbontású képernyő esetén a koordináták értékeinek a $0 <= xi <= 319$ illetve a $0 <= yi <= 199$ intervallumokba kell esniük.

0<=xi<=319 illetve a 0<=yi<=199 intervallumokba kell esniük.

!GCLEAR

Törli a nagyfelbontású képernyőt. A kisfelbontású képernyő egyszerűen törölhető a ?"<CLR>" utasítással.

!GRAPHICS p

Az utasítás a p paraméter értékétől függően meghatározza, hogy milyen üzemmódban dolgozzon az interpreter. p lehetséges értékei és a megfelelő üzemmódok a következők:

p = 0 a kisfelbonású képernyő kerül kijelzésre, és az ennek megfelelő utasítások hajtódnak végre;

p = 1,2 a nagyfelbontású képernyő kerül kijelzésre, és az ennek megfelelő utasítások hajtódnak végre;

p = 3,4 a nagyfelbontású képernyő kerül kijelzésre, de a kis felbontásnak megfelelően hajtódnak végre az utasítások;

p = 5 a kisfelbontású képernyő kerül kijelzésre, de az összes utasítás a nagyfelbontású képernyőre vonatkozik, és a bittérképen a megfelelő változások megtörténnek, bár nem látszanak.

Tekintsük a következő programot:

```
10 !GRAPHICS 5:!GCLEAR
20 !LINE 319,0 TO 0,199
30 GET A$: IF A$="" THEN 30
40 !GRAPHICS 1:END
```

A RUN parancs végrehajtása után a 30.programsorban az interpreter egy billentyű lenyomására vár. A rajz (egy átlós vonal) már kész, de nem látszik. Nyomjunk le egy tetszőleges billentyűt és rögtön a nagyfelbontású képernyőt látjuk.

!DOT x1,y1

A képernyő (x1,y1) koordinátájú pontját bekapcsolja. Ha a koordináták nem esnek az üzemmódnak megfelelő értékhatárok közé, hibajelzést kapunk. Ennek az egy utasításnak a segítségével – bár kicsit lassan – tetszőleges képet rajzolhatunk. A következő példa egy szinuszgörbét rajzol a képernyőre:

```
10 !GRAPHICS 1:!GCLEAR
20 FOR I=0 TO 319
30 !DOT I,98*SIN(4*I*<pi>/320)+100
40 NEXT I
```

!CDOT x1,y1

A képernyő (x1,y1) koordinátájú pontját kikapcsolja. Ha a koordináták nem esnek az üzemmódnak megfelelő értékhatárok közé, hibajelzést kapunk. Ha a fenti példához hozzáírjuk a következő sorokat, akkor egy olyan programot kapunk, ami rajzolás után letörli a szinuszgörbét:

```
50 FOR I=319 TO 0 STEP-1
60 !CDOT I,98*SIN(4*<pi>*I/320)+100
70 NEXT I
```

!TEST x1,y1,<változó>

Az interpreter ellenőrzi, hogy az (x1, y1) koordinátájú pont be van-e kapcsolva. Ha igen, a <változó> értéke 1 lesz, különben 0. Tipikus használata:

```
15 !TEST 173,92,x:IF x THEN GOTO 232
```

!LINE x1,y1 TO x2,y2

Az (x1,y1) és az (x2,y2) pontok közt egyenes húz. Az egyenest az (x1,y1) ponttól (kezdőpont) kezdi az interpreter megrajzolni; utolsónak kapcsolva be az (x2,y2) pontot. Az alábbi program egy derékszögű koordinátarendszert rajzol:

```
10 !GRAPHICS1:!!GCLEAR
20 !LINE 159,0 TO 159,199
30 !LINE 0,99 TO 319,99
```

!CLINE x1,y1 TO x2,y2

Az (x1,y1) és az (x2,y2) pontokat összekötő egyenes mentén minden pontot kikapcsol.

!DLINE x1,y1 TO x2,y2

Az utasítás hatása majdnem megegyezik a !LINE utasításéval, azzal a különbséggel, hogy az egyenes minden második pontját kapcsolja csak be, 'szaggatott' egyenest rajzol.

!FILL x1,y1 TO x2,y2

Az utasítás a fenti ábrán látható téglalap határán és belsejében levő összes pontot bekapcsolja. A bekapcsolás fentről lefelé, azután pedig balról jobbra halad. Először kerül sor az (x1,y1) pont bekapcsolására, majd az oldalra levő pontéra stb., legvégül az (x2,y2) pontot kapcsolja be az interpreter.

!CLEAR x1,y1 TO x2,y2

Az utasítás a !FILL utasítás ellentéte, a fenti téglalap határán, illetve belsejében levő pontokat kapcsolja ki ugyanabban a sorrendben, ahogy a !FILL bekapcsolta őket. Például az alábbi programrész a képernyő közepére rajzol egy négyzetet, majd törli:

```
10 !GRAPHICS 1:!GCLEAR
20 !FILL 154,94 TO 164,104
30 FOR I=1 TO 500:NEXT I
40 !CLEAR 154,94 TO 164,104
```

!FRAME x1,y1 TO x2,y2

Az utasítás az ábrán látható téglalap határán levő összes pontot bekapcsolja, ugyanabban a sorrendben, ahogy a !FILL utasítás bekapcsolta a pontokat.

!CFRAME x1,y1 TO x2,y2

Hatása ellentétes a !FRAME utasítás hatásával, az ott látható téglalap határán levő pontokat kapcsolja ki, ugyanabban a sorrendben.

!MOVE x1,y1

A grafikus kurzort az (x1,y1) helyre pozicionálja. Az előbb felsorolt utasítások végrehajtása után a grafikus kurzor **mindig** az utoljára be- vagy kikapcsolt pontra mutat. A !MOVE utasítás a következő utasítással együtt használható.

!DRAW x2,y2

A grafikus kurzor jelenlegi helyzetétől az (x2,y2) pontig húz egy egyenest. A következő program a képernyő közepére rajzol egy négyzetet:

```
10 !GRAFICS 1:!GCLEAR
20 !MOVE 154,94:!DRAW 164,94
30 !DRAW 164,104:!DRAW 154,104
40 !DRAW 154,94
```

!DISP p, x1,y1, <sztring>

A fenti utasítás segítségével a <sztring>-ben definiált alakzatot kiírhatjuk a **nagyfelbontású** képernyőre. Első lépésként az interpreter a grafikus kurzort az (x1,y1) pontra állítja, majd elkezdi a <sztring> feldolgozását. A <sztring>-ben levő vezérlő karakterek hatására a grafikus kurzor pontosan úgy mozog, mint a valódi. A kinyomtatható jelek helyett azonban csak egy pont kerül kiírásra. Pontosabban, ha p=0, akkor a megfelelő helyen levő pont kikapcsolódik,ha pedig p=1,akkor bekapcsolódik. A !DISP hatását legegyszerűbb az alábbi program segítségével kipróbálni:

```
10 A$="<R><D><R><D>x<R><D>x<R><D>x
20 B$=A$+A$+A$
30 !GRAPHICS 1:!GCLEAR
40 !DSIP 1, 100,100, B$
```

(A 10. sorban <R> = <CRSR JOBBRA>, <D> = <CRSR LE>)

!CIRCLE x1,y1, x2,y2 ,x3,y3

Az utasítás az (x1,y1) közzéppont körül x2,y2 féltengelyű ellipszist rajzol. Ha x3,y3 is szerepel, akkor az azt a szögtartományt definiálja, amelybe eső részét az ellipszisnek kell rajzolni. A szögeket fokban kell megadni.

```
10 !GCLEAR:!GRAPHICS 1
20 !CIRCLE 40,40,30,20
30 !CIRCLE 40,40,29,19
40 OPEN 4,4:!HARD#4
```

!CCIRCLE x1,y1, x2,y2 ,x3,y3

Az utasítás a !CIRCLE utasítás ellentettje, az (x1,y1) középpont körüli x2,y2 féltengelyű ellipszis mentén minden egyes pontot kikapcsol. Ha x3,y3 is szerepel, akkor az azt a szögtartományt definiálja, amelybe eső részét az ellipszisnek ki kell törölni.

!DCIRCLE x1,y1, x2,y2 ,x3,y3

Az utasítás a !CIRCLE utasítás párja. Az (x1,y1) középpont körüli x2,y2 féltengelyű ellipszis mentén minden második pontot bekapcsol ('szaggatott' ellipszist rajzol). x3, y3 jelentése ugyanaz, mint a !CIRCLE utasításban.

!TEXT <sztring>, x1,y1, p ,<szín>

Az utasítás segítségével a <sztring> tartalmát a **nagyfelbontású** képernyőre írathatjuk ki. x1,y1 a szöveg kezdő pozícióját adja meg, p a karakterkészletet azonosítja. p=0 esetén a 'nagybetűk/ grafikus jelek' karakterkészlet, p=1 esetén a 'kisbetűk/ nagybetűk' karakterkészlet felhasználásával állítja elő a szöveget a nagyfelbontású képernyőn.

A <szín> paraméter a szöveg színét adja meg. Ha elmarad, akkor a szöveg a legutolsó !LCOL utasításban definiált színnel íródik ki.

A következő példa a képernyő közepén bekeretezve kiírja a programcsomag (angol helyesírás szerinti) nevét:

```
10 !GRAPHICS 1:!GCLEAR:!FRAME 0,90 TO 319,102
20 A$="SUPERGRAPHICS/64":!TEXT A$, 98,100,0
30 GET X$: IF X$="" THEN 30
40 !GRAPHICS 1
```

!BCOL <szín>

Az utasítás a képernyők háttérszínét definiálja. Miután a SUPERGRAFIK a többszínű üzemmódot nem támogatja, ezért !BCOL valójában a kikapcsolt pontok színét definiálja. A SUPERGRAFIK a !BCOL kezdőértékét 0-ra (fekete) állítja.

!LCOL <szin>

Az utasítás a bekapcsolt pontok színét definiálja, kezdőértéke 7 (sárga).

A képernyők tárolására, illetve külső tárolóból való visszatöltésére a következő utasítások használhatók:

!GSAV "d:fn", dn

A képernyő tartalmát kimenti a dn egységszámú lemezegység d sorszámú meghajtóján levő fn nevű file-ba.

!RECALL "d:fn", dn

A dn egységszámú lemezegység d sorszámú meghajtóján levő fn nevű file-t betölti az aktuális képernyő-memóriába.

Említettük, hogy két nagyfelbontású képernyő van. Ezek közül az elsőt használja az interpreter. A !TRANS utasítás hatására a két bit-térkép felcserélődik. Az első !TRANS utasítás után általában újból törölni kell a nagyfelbontású képernyőt, hogy az oda újból bekerülő véletlen biteket töröljük.

```
10   !GRAPHICS 1 : !GCLEAR
     ...
     (1.képernyő szerkesztés)
     ...
1000 !TRANS : !GCLEAR
     ...
     (2.képernyő szerkesztés)
     ...
2000 !TRANS : REM AZ ELSO KEPERNYO LATSZIK!
```

!HARD#lf

A képernyő tartalmát kinyomtatja az lf sorszámú logikai file-ba. A 'kinyomtatja' kifejezést az indokolja, hogy a C-64 kimenetén a képernyő tartalma a C-64 kompatibilis nyomtatóknak megfelelő formában jelenik meg. A nyomtatási kép a bekapcsolt pontoknak megfelelő pontokból áll össze. A bekapcsolt pont színe érdektelen. Az

```
OPEN 4,4 : !HARD#4
```

parancs a nyomtatóra kimásolja a képernyő tartalmát.

Sprite-ok kezelésére a következő utasítások állnak rendelkezésre:

!SPRITE sn, <sztring>

A fenti utasítás az sn. sorszámú sprite alakjának megadására szolgál. A <sztring> hossza kötelezően 63 byte. A sztring elemeinek ASCII kódja rendre adják a sprite-ot definiáló byte-okat.

Ha ezek értékét DATA utasitásokban helyeztük el, akkor a megfelelő sztring a következő programrésszel állitható elő:

```
10 B$=""
20 FOR I=0 TO 62
30 READ A:B$=B$+CHR$(A)
40 NEXT I
```

Ezután, mondjuk a 4. sprite definiálásában a B$-t a következőképpen használhatjuk fel:

```
50 !SPRITE 4,B$
```

Célszerű még egy B$ = "" utasítást is kiadni. Ennek hatására a B$ 63 byte-ja felszabadul.

!SMODE sn, pl, co

Ezzel az utasítással definiálhatjuk az sn. sorszámú sprite megjelenési módját.

pl = 0 normál méretben;
pl = 1 X irányban nagyítva;
pl = 2 Y irányban nagyítva
pl = 3 mindkét irányban nagyítva
co a sprite színe (0-15).

A legegyszerűbb eset: !SMODE 4,0,12

!SMOVE sn, x1, y1

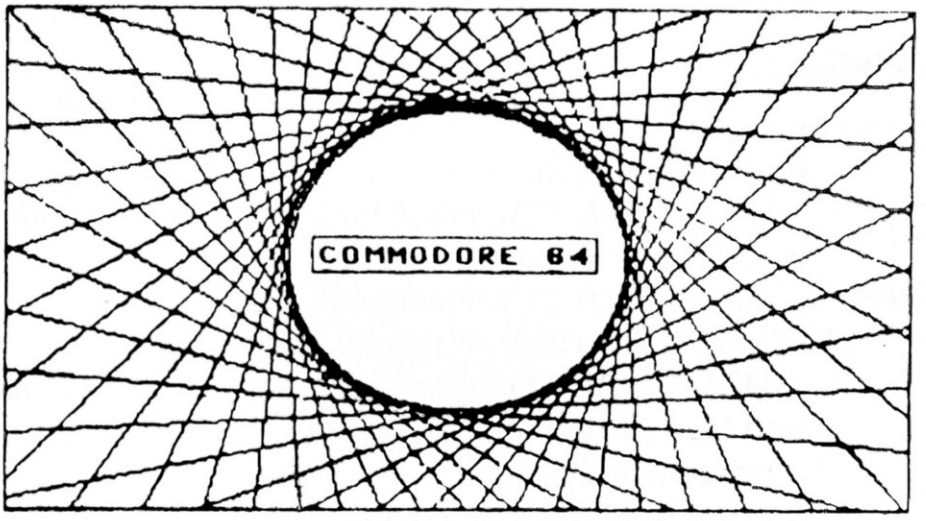
Az utasítás beállítja az sn. sprite helyzetét. Ebben az esetben az (x1,y1) koordinátapár a sprite térre vonatkozik. Így az egyes koordináták a következő feltételeknek kell, hogy eleget tegyenek:

0 < = x1 < = 511;
0 < = y1 < = 255.

Példa SUPERGRAFIK programra:

```
  5 !BCOL 8:!LCOL 7
 10 !GCLEAR: !GRAPHICS 1
 20 !CIRCLE 159,99,60,60
 30 FOR ALFA=0.1 TO 2*<pi>+0.1 STEP <pi>/20
 35 BETA=<pi>/2+ALFA
 40 X0=159+60*COS(ALFA)
 50 Y0=99+60*SIN(ALFA)
 60 B=Y0-TAN(BETA)*X0
 70 GOSUB 490
```

```
100 !LINE X(0),Y(0) TO X(1),Y(1)
110 NEXT
120 !TEXT "COMMODORE 64",110,101,0,0
121 !LCOL 0
122 !FRAME 108,90 TO 208,104
125 !FRAME 0,0 TO 319,199
130 GOTO 2000
490 I=0
500 Y=B
510 IF Y<0 OR Y>199 THEN GOTO 550
520 Y(I)=Y:X(I)=0:I=I+1:IF I=2 THEN RETURN
550 Y=TAN(BETA)*319+B
560 IF Y<0 OR Y>199 THEN GOTO 600
570 Y(I)=Y:X(I)=319:I=I+1:IF I=2 THEN RETURN
600 X=-B/TAN(BETA)
610 IF X<0 OR X>319 THEN GOTO 650
620 Y(I)=0:X(I)=X:I=I+1:IF I=2 THEN RETURN
650 X=(199-B)/TAN(BETA)
660 IF X<0 OR X>319 THEN RETURN
670 Y(I)=199:X(I)=X:I=I+1:RETURN
2000 GET A$:IF A$<>"" THEN END
2010 GOTO 2000
```



7.8 SIMONS' BASIC: grafikus utasításkészlet

Bevezetés

A SIMONS' BASIC grafikus utasításai lehetővé teszik a nagyfelbontású, tehát bit térképes képernyő használatát mind standard, mind többszínű üzemmód esetén. A standard üzemmód esetén a képernyő 320*200-as pontrácsnak felel meg, s az egyes utasításokban ennek megfelelő paramétereket kell használni. Többszínű üzemmód esetén a SIMONS' BASIC interpreter a képernyőt 160*200-as pontrácsnak tekinti, s az ennek megfelelő paraméterértékeket kell használni. A kis- és nagyfelbontású képernyőhöz különböző típusú grafikus utasítások tartoznak.

A utasításkészlet nem tartalmaz külön törlő utasításokat. Az egyes utasításokban szereplő <plot-typ> paraméter ('rajzolás módja') értéke határozza meg, milyen színű pontot rajzoljon. Ennek értékei a következők lehetnek:

Standard üzemmód esetén:

<plot-typ>	jelentés
0	törlés
1	rajzolás
2	invertálás (ahol volt pont, azt törli,ahol nem, oda tesz)

Többszínű üzemmódban:

<plot-typ>	jelentés
0	törlés (Háttérszín # 0)
1	a MULTI/LOW COL utasítások első szín (a képernyő memória byte felső 4 bitjének megfelelő szín)
2	a MULTI/LOW COL utasítások második színe (a képernyő memória byte alsó 4 bitjének megfelelő szín)
3	a MULTI/LOW COL utasítások harmadik színe (a színmemóriában levő szín)
4	a pont invertálása 0 színű pont inverze 3 színű 1 színű pont inverze 2 színű 2 színű pont inverze 1 színű 3 színű pont inverze 0 színű

COLOUR ck, ch

Az utasítás a háttér, illetve a keret színét állítja be. ch a háttér, ck a keret színének kódja (0-15).

HIRES cp, ch

Az utasítás törli a nagyfelbontású képernyőt. A ch paraméter a háttér színét állítja be. A cp paraméter utasítja az interpretert, hogy a grafikus utasításokban a rajzoláskor cp színű pontokat használjon. Ez a szín felel meg a plot-typ = 1 értéknek.

REC x,y, a,b, < plot-typ >

Az (x,y), (x + a,y), (x,y + b), (x + a,y + b) pontok által meghatározott téglalap határpontjait rajzolja meg. Például a

```
10 COLOUR 1,1:HIRES 0,1
20 REC 0,0,40,20,1
30 GOTO 30
```

programrész egy teljesen fehér képernyőre egy fekete téglalapot (fehér belső résszel) rajzol.

MULTI c1, c2, c3

Az utasítás a többszínű-üzemmód nem háttér színeit definiálja. Általában a HIRES utasítással együtt használjuk:

```
HIRES 0,1:MULTI 0,2,6
```

NRM

A kijelzés karakteres üzemmóddal folytatódik. A program futásának bármely okból való megszakítása egyben ennek az utasításnak a végrehajtását is jelenti.

LOW COL c1, c2, c3

További három alternatív színt definiál. Többszín üzemmódban ezt követően ezeket használja az interpreter

HI COL

Az interpreter az utasítás végrehajtását követően újból a MULTIban definiált színeket használja.

PLOT x,y, < plot-typ >

A nagyfelbontású képernyő (x,y) koordinátájú pontja a < plot-typ > értékének megfelelően beállítódik.

TEST (x,y)

TEST aritmetikai függvény, melynek értéke 0, ha az (x,y) koordinátájú pont háttérszínű, különben 1. Az utasítás teszteli, ellenőrzi, berajzoltunk-e egy pontot vagy sem.

LINE x,y,x1,y1, < plot-typ >

Az utasítás az (x,y) és (x1,y1) pontokat összekötő egyenest rajzolja meg, először az (x,y) pontot és legvégül az (x1,y1) pontot. A rajzolás – mint minden grafikus utasításban – < plot-typ > értékének megfelelően történik.

CIRCLE x,y, xr, yr, < plot-typ >

Az utasítás egy (x,y) középpontú és xr,yr tengelyű ellipszist rajzol, a < plot-typ > értékének megfelelően. Ha xr = yr, akkor egy kört kapunk.

ARC x,y, sa, ea, i, xr, yr, < plot-typ >

Az utasítás egy (x,y) középpontú és xr,yr tengelyű ellipszis ívét rajzolja meg. Az ívek végpontjaihoz tartozó átmérők az x tengellyel sa, illetve ea szöget zárnak be. A rajzolás a < plot-typ > értékének megfelelően történik.

Az i paraméter azt határozza meg, hogy az ellipszis mely pontjait rajzolja meg az interpreter. Az ellipszis így megrajzolt pontjait azután egyenes szakaszokkal köti össze. Először az sa szögű ellipszisponthoz tartozó ellipszispont kerül megrajzolásra, majd az sa + i, sa + 2i, stb. szögekhez tartozó és ezeket a pontokat szakaszokkal köti össze az interpreter. Minél kisebb i értéke, annál 'simább' ellipszist kapunk.

i hatását a következő példa jól mutatja:

```
10 HIRES 0,1
20 ARC 100, 51, 0,360, 18,50,50,1
30 ARC 200, 51, 0,360, 60,50,50,1
40 HRDCPY
```

PAINT x,y,<plot-typ>

Az utasítás segítségével egy tetszőleges, konvex, zárt vonallal határolt alakzat belső pontjait szinezhetjük be a <plot-type>-nak megfelelően. Az interpreter az (x,y) pontból indul el, először felülről lefelé, majd jobbról balra. Ha háttérszínű pontot talál, akkor az abban az irányban való mozgását abbahagyja, és megpróbál irányt változtatni. Ha sikerül, akkor továbbhalad, ha nem, akkor véget ér az utasítás végrehajtása. A PAINT utasítás a HIRES/MULTI/LOW COL utasításokban elsőnek definiált színt használja.

A PAINT utasítás hatását az alábbi két program összehasonlításával lehet szemléltetni:

```
10 HIRES 0,1
20 REC 120,60,40,40,1
30 PAINT 130,70,1
```

illetve

```
10 HIRES 0,1
20 REC 120,60,40,40,1:PLOT 140,100,0
30 PAINT 130,70,1
```

BLOCK x,y, x1,y1,<plot-typ>

Az (x,y), (x1,y), (y,x1), (x1,y1) pontok által meghatározott téglalap valamennyi pontjára végrehajtja a <plot-typ>-nak megfelelő műveletet. A BLOCK utasítás egy REC és egy PAINT egymás utáni végrehajtásával helyettesíthető.

DRAW <sztring>, x,y,<plot-typ>

Az utasítás az (x,y) ponttól kezdődően a <plot-typ> értékének, valamint a <sztring>-ben szereplő vezérlőkaraktereknek megfelelő alakzatot rajzol (a grafikus kurzor mozgatása). A sztring legfeljebb 74 vezérlőkaraktert tartalmazhat. Az egyes –normál– méretben egy pontnyi elmozdulást eredményező vezérlőkarakterek a következők:

0	kurzor jobbra
1	kurzor fel
2	kurzor le
3	kurzor balra
5	pont rajzolása + kurzor jobbra
6	pont rajzolása + kurzor fel
7	pont rajzolása + kurzor le
8	pont rajzolása + kurzor balra
9	befejezi a rajzolást

A DRAW utasítás **csak** egy előzetes ROT utasítás végrehajtása **után** okoz látható eredményt.

ROT r,s

Megadja az interpreternek, hogy ezután a DRAW utasítás segítségével rajzolandó alakzatok hány fokkal elforgatva és mekkora nagyítással jelenjenek meg a képernyőn. A forgatás, illetve a nagyítás középpontja egyaránt a DRAW utasításban megadott (x,y) kezdőpont. Az r paraméter lehetséges értékeit a következő táblázat foglalja össze:

r	fok
0	0
1	45
2	90
3	135
4	180
5	225
6	270
7	315

Az s paraméter a nagyítás arányát adja. s = 1 esetben normál méretben kerül a rajz a képernyőre. s = 2 esetében a nagyítás kétszeres stb.

```
10 A$="57575757878787575757575757777777777777575757575757888888888888"
20 A$=A$+"888888888888886565656565656566666666666666"
30 A$=A$+"565656565656868686565656"
40 HIRES 0,1
50 FOR Y=0 TO 7
60 FOR X=1 TO 3
70 ROT Y,X
80 DRAW A$,160,80,1
90 PAUSE 1
100 DRAW A$,160,80,1
110 NEXT:NEXT
115 FOR X=3 TO 1 STEP -1
120 FOR Y=7 TO 0 STEP -1
130 ROT Y,X
140 DRAW A$,160,80,1
150 PAUSE 1:DRAW A$,160,80,0:NEXT:NEXT
160 GOTO 50
```

CSET n

Megadja, hogy melyik karakterkészlettel dolgozzon az interpreter. (A <C=> és <SHIFT> billentyűk együttes lenyomása **nem** változtatja meg a karakterkészletet, csak a fenti utasítás.)

CSET 0 a 'nagybetűk/grafikus jelek' karakterkészletet, a CSET 1 a 'kisbetűk/ nagybetűk' karakterkészletet használja. A CSET 2 utasítás után a MULTI visszaállítja a grafikus képernyőt.

```
10 HIRES 0,1:MULTI 0,4,6
15 FOR I=1 TO 10
20 X=INT(95*RND(1))+2:Y=INT(95*RND(1))+2
25 Z=INT(95*RND(1))+2:W=INT(95*RND(1))+2
27 P=INT(3*RND(1))+1
30 REC X,Y,Z,W,1
35 PAINT X+1,Y+1,P
40 NEXT I
45 PAUSE2:CSET1
50 PRINT"<CLR>AAAAAAAAAAA"
55 PRINT"<CRS LE>AAAAAAAAAAA"
60 PAUSE 1
65 CSET2:MULTI 0,4,6:PAUSE2:CSET2
70 GOTO 15.
```

CHAR x,y,<kód>,<plot-typ>, n

A nagyfelbontású képernyő (x,y) koordinátájú pontjában elhelyezi a <kód> értékének megfelelő karaktert, a <plot-typ>-nak megfelelően. Az n a karakter nagyítását definiálja; ha n=1, akkor a karakter egy 8*8-as részt foglal el, ha például n=3, akkor a karakter egy 8*24-es részbe kerül elhelyezésre. n maximális értéke 8 lehet.

TEXT x,y,<sztring>,<plot-typ>, n,i

Az utasítás karaktersorozatok nagyfelbontású képernyőn való elhelyezésére szolgál. A <sztring>-ben levő szöveg kerül kiírásra a <plot-typ> értékének megfelelően. n a **függőleges** irányú nagyítást adja, maximális értéke 8. i értéke adja meg, hogy milyen sűrűn írja ki az interpreter a karaktereket. n=1, i=8 értékek esetén a kiírás ugyanaz lesz, mint a PRINT <sztring> utasítás esetén. i=11 esetén a karakterek közé 3*8-as 'szóköz' kerül. i<8 esetén a betűk már egymásra íródnak.

A <sztring> karaktersorozat két vezérlőkaraktert tartalmazhat, amelyek előírják, hogy kis- vagy nagy betűk kerüljenek-e a képernyőre, függetlenül attól, hogy a sztring milyen karaktereket tartalmaz.

<CTRL-A> = nagybetű (inverz A-ként jelenik meg)
<CTRL-B> = kisbetű (inverz B-ként jelenik meg)

A következőkben azokat az utasításokat ismertetjük, amelyek a karakter üzemmódban való programozást segítik elő.

BCKGNDS sc, b1, b2, b3

Utasítja a video-chipet, hogy kiterjesztett háttér-üzemmódban dolgozzon. Az sc,b1, b2,b3 paraméterek értéke 0-7 lehet. Ebben az üzemmódban a karakterkészlet 0-63 kódú elemei érhetők csak el (lásd a fejezet megfelelő részét!). sc a képernyő színét definiálja, b1,b2 és b3 pedig a háttérszíneket. Az utasítás hatását a következő programrész illusztrálja:

```
10 PRINT "<CLR>"
20 BCKGNDS 1,3,5,6
30 PRINT "SIMONS' BASIC":PRINT
40 PRINT "<CTRL-RVS ON>SIMONS' BASIC":PRINT
50 PRINT "<CTRL-RVS ON>SIMONS' BASIC":PRINT
```

(**Fontos**: a dőlt betűs karaktereket a <SHIFT> billentyű lenyomva tartása **közben** vittük be a gépbe!)

FLASH co, <seb>

A képernyő a co színnel felírt szövegrészeket villogtatja. Ha a <seb> paraméter is szerepel, akkor ez határozza meg a villogás sebességét, gyakoriságát. <seb> értékének az 1-255 intervallumba kell esnie. A FLASH utasítás mind a nagyfelbontású, mind a kisfelbontású képernyő esetén használható. Az utasítás használatánál arra kell ügyelnünk, hogy a színek kódszámai és a <CTRLszámjegy> utasításban szereplő számjegy különböznek. A 2. színt a <CTRL-3> billentyűzésével érhetjük el. Példánkban a képernyő közepére háromszor kiírjuk az ELJEN szöveget. Ezek közül a középsőt villogtatjuk:

```
10 PRINT "<CLR>"
20 PRINT AT (12,10)" ELJEN<CTRL 3>ELJEN<CTRL 1>ELJEN"
30 FLASH 2,10
40 PAUSE 5
```

OFF

Megszünteti a FLASH utasítás hatását. A fenti példát a következő három utasítással egészíthetjük ki:

```
50 OFF
60 A$="ELJEN"
70 PRINT AT (12,10)A$+A$+A$
```

A középső ELJEN öt másodpercig villog.

BFLASH <seb>, c1,c2 vagy BFLASH 0

A képernyő keretét villogtatja úgy, hogy a színét folytonosan c1-ről c2-re változtatja. A <seb> paraméter a villogás sebességét határozza meg, értéke az 1-255 intervallumba kell, hogy essen. Minél nagyobb <seb> értéke, annál lassabb a villogás. <seb> =60 másodpercenként egy villanásnak (színváltásnak) felel meg. A BFLASH 0 utasítás megállítja a képernyő villogását.

FCHR r,c,w,d,<kód>

Az alacsony felbontású képernyő (r,c),(r+w,c),(r,c+d),(r+w,c+d) karakterhelyei által meghatározott téglalapját kitölti a <kód>nak megfelelő karakterekkel. Így összesen w*d karakter kerül kiírásra.

FCOL r,c,w,d,co

Az alacsony felbontású képernyő (r,c),(r+w,c),(r,c+d),(r+w,c+d) karakterhelyei által meghatározott téglalap valamennyi karakterének színét co-ra változtatja. A következő programrész a teljes képernyőt sárga A betűvel tölti meg:

```
10 FCHR 0,0,40,25,1
20 FCOL 0,0,40,25,7
```

FILL r,c,w,l,<kód>, co

Az utasítás az (r,c), (r+w,c), (r,c+l), (r+w,c+l) koordinátájú karakterhelyek által meghatározott téglalapot a <kód> értékének megfelelő, co színű karakterekkel tölti fel.

MOVE r,c,w,l, dr, dc

Az r,c,w,l értékek által meghatározott téglalapot átmásolja a (dr,dc) karakterhelytől kezdődően. Mint a fenti utasítások esetén, a téglalap csúcsai (r,c),(r+w,c),(r,c+l) és (r+w,c+l). Ha az eltolt téglalap kívűl esne a képernyőn, BAD MODE hibajelzést kapunk.

INV r,c,w,l

Invertálja az (r,c),(r+w,c),(r,c+l),(r+w,c+l) pontok által meghatározot téglalap minden egyes karakterét.

Következő példánk a fenti három utasítás hatását mutatja be:

```
5 PRINT "<CLR>"
10 FILL 5,5,10,15,1,0
15 MOVE 5,5,10,15,20,10
20 INV 15,10,20,15
25 GOTO 20
```

A képernyő görgetése

<irány>W, sr, sc, ec, er
<irány>B, sr, sc, ec, er

A fenti utasítások a paraméterek által megadott részét tolják el a képernyőnek az <irány> paraméter által megadott irányban. Az <irány> valójában **nem** paraméter, hanem az utasítás része. A lehetséges irányok LEFT, RIGHT, UP és DOWN. Így például LEFTW és UPB görgető utasítások lesznek. Az utolsó négy paraméter által meghatározott téglalapot tolja el az utasítás a megfelelő irányban. Amennyiben az utasítás a W betűvel végződik, az egyik oldalon kilépő karakterek a másik oldalon visszalépnek, míg a B betű azt jelenti, hogy a kilépő karakterek elvesznek és a másik oldalon szóközök lépnek be (W = WRAP ROUND; B = BLANKING). A következő program egy szinuszgörbét mozgat a képernyőn keresztül.

```
10 PRINT"<CLR>"
20 FOR X=0 TO 39
30 Y=INT(11*SIN(X/<pi>))+12
40 PRINT AT(X,Y)"●"
50 NEXT
60 LEFTW 0,0,40,25
70 GOTO 60
```

SCRSV 2,8,2 "<név>,S,W" vagy
SCRSV 1,1,1, "<név>"

A kisfelbontású képernyő tartalmát a <név> nevű file-ba másolja. Ezt követően a képernyő tartalma a SCRLD utasítás segítségével visszatölthető. Az utasítás első formája a lemezre vonatkozik, a 8-as lemezegység 2-es csatornáján keresztül a képernyő tartalma a <név> nevű szekvenciális file-ba íródik. Az S és W betűk azt jelentik, hogy egy szekvenciális file-t nyitottunk meg írásra. Az utasítás második formája a kazettás egységre vonatkozik. A <név> nevű szalagos file-ba íródik ki a képernyő tartalma. A SCRSV utasítás paraméterei megegyeznek az OPEN utasítás paramétereivel.

SCRLD 2,8,2, "<név>" vagy
SCRLD 1,1,0, "<név>"

A <név> nevű lemezes, illetve szalagos file-ból visszatölti a képernyőt. Az utasítás paraméterei megegyeznek az OPEN utasítás paramétereivel.

A következő programrész egy zászlót rajzol a képernyőre, kimenti egy lemezes file-ba, törli a képernyőt, majd visszatölti a kimentett állapotot:

```
10 PRINT "<CLR>"
20 FILL 6,10,20,4,160,2
30 FILL 10,10,20,4,160,1
40 FILL 14,10,20,4,160,5
50 SCRSV 2,8,2,"MAGYAR ZASZLO,S,W"
60 GOTO 60

SCRLD 2,8,2,"MAGYAR ZASZLO"
```

COPY

A nagyfelbontású képernyőt átmásolja a nyomtatóra.

HRDCPY

A kisfelbontású képernyőt átmásolja a nyomtatóra.

A két utasítás végrehajtása közt lényeges különbség van. Az utóbbi esetben a képernyő-memóriában tárolt kódoknak megfelelő karakterek kerülnek nyomtatásra. Ha tehát programozható karaktereket használtunk, akkor a nyomtatón megjelenő szöveg egészen más lehet, mint a képernyőn látható. A COPY a bit-térképet másolja át, így – kivéve a többszín-üzemmódot – a nyomtatón ugyanazt a képet kapjuk, mint a képernyőn.

Végül azokat a SIMONS' BASIC utasításokat ismertetjük, amelyek lehetővé teszik a sprite-ok kényelmes programozását.

DESIGN p, ad

A DESIGN utasítás sprite-ok definiálását teszi lehetővé. Az utasítást követően a @ jellel kezdődő sorok határozzák meg a sprite alakját. A p paraméter adja meg, hogy a sprite standard vagy többszínű üzemmódban kerül felhasználásra. p = 0 standard, p = 1 többszínű üzemmódot jelent. ad tetszőleges aritmetikai kifejezés lehet, amelyik megadja azt a címet, ahová a sprite-ra vonatkozó információt (63 byte-ot) az interpreternek töltenie kell. ad értéke csak 64 többszöröse lehet. Az ad paraméter megadásához tudni kell, hogyan helyezkedik el a program a memóriában, nehogy felülírjuk (lásd a 7. fejezet ide vonatkozó részét!).

Sprite-ok definiálása

@ < karaktersorozat >

Egy sprite definiálásához 21 @-utasításra van szükség. Egy utasítás a sprite egy sorának alakját definiálja. A @ jel után álló karaktersorozat 24 vagy 12 hosszú, attól függően, hogy standard vagy többszínű sprite-ot definiálunk-e. A karaktersorozat a következő jeleket tartalmazhatja:

Standard üzemmód

.	Háttérszín#0
B	A sprite szín-regiszterben definiált szin (a MOB SET utasítás állíthatja)

Többszínű üzemmód

.	Háttérszín#0
B	A CMOB utasítás első színe
C	A sprite szín-regiszterben definiált szín (a MOB SET utasítás állíthatja)
D	A CMOB utasítás második színe

Példákat a következő oldalon adunk.

CMOB c1,c2

A sprite többszín-üzemmód színeit definiálja. c1,c2 értéke 0-15 lehet. c1 színűek lesznek a sprite-oknak a @ utasításban levő B betűknek, c2 színűek a D betűknek megfelelő pontjai.

MOB SET sn,sm,co,pr,m

Az utasítás beállítja egy adott sprite színét, üzemmódját, a háttérhez való prioritását, és hogy melyik sprite-lapot tekintse a sprite definíciójának, végül engedélyezi a sprite megjelenését. Mivel a sprite helyzetét megadó mutatók kezdő értéke 0, ezért a sprite még nem látszik, ehhez megfelelő helyre kell állítani. Az egyes paraméterek jelentése a következő:

sn: a sprite sorszáma (0-7)
sm: a sprite-lap sorszáma, ami a sprite definicióját tartalmazza (0-255, de ezek közül csak néhány használható)
co: a sprite színe (0-15)
pr: prioritás a háttérrel szemben, ha
0, akkor a sprite látszik a háttér, ha
1, akkor a háttér látszik a sprite előtt
m: üzemmód, ha
0, akkor standard, ha
1, akkor többszínű üzemmód.

```
  5 HIRES 8,7
 20 CIRCLE 159,99,60,60,1
 30 FOR ALFA=0.1 TO 2*<pi>+0.1 STEP <pi>/20
 35 BETA=<pi>/2+ALFA
 40 X0=159+60*COS(ALFA)
 50 Y0=99+60*SIN(ALFA)
 60 B=Y0-TAN(BETA) X0
 70 GOSUB 490
100 LINE X(0),Y(0),X(1),Y(1),1
110 NEXT
120 TEXT 111, 94," COMMODORE 64",1,2,8
122 REC 107,90,105,22,1
123 PAUSE 5
125 BLOCK 0,0,319,199,1
130 END
490 I=0
500 Y=B
510 IF Y<0 OR Y>199 THEN GOTO 550
520 Y(I)=Y:X(I)=0:I=I+1:IF I=2 THEN RETURN
550 Y=TAN(BETA)*319+B
560 IF Y<0 OR Y>199 THEN GOTO 600
570 Y(I)=Y:X(I)=319:I=I+1:IF I=2 THEN RETURN
600 X=-B/TAN(BETA)
610 IF X<0 OR X>319 THEN GOTO 650
620 Y(I)=0:X(I)=X:I=I+1:IF I=2 THEN RETURN
650 X=(199-B)/TAN(BETA)
660 IF X<0 OR X>319 THEN RETURN
670 Y(I)=199:X(I)=X:I=I+1:RETURN
```

A program futásának eredménye a 249. oldalon látható.

MMOB sn, x1,y1, x2,y2, n, <seb>

Az utasítás egy sprite-ot mozgat a képernyőn. Valódi mozgás természetesen csak a megfelelő MOB SET utasítás kiadása után látszik. Az egyes paraméterek jelentése a következő:

sn: a sprite sorszáma (0-7)
x1: a sprite kezdő pozíciójának x koordinátája (0-511)
y1: a sprite kezdő pozíciójának y koordinátája (0-255)
x2: a sprite végpozíciójának x koordinátája (0-511)
y2: a sprite végpozíciójának y koordinátája (0-255)
n: a sprite nagyítása
0 = normál méret
1 = x irányú nagyítás
2 = y irányú nagyítás
3 = mindkét irányú nagyítás
<seb>: a mozgás sebessége, 255 a leglassúbb (ennyi közbenső helyen jelenik meg a sprite).

RLOCMOB sn, x2,y2, n, <seb>

Az utasítás a sprite-ot jelenlegi helyéről az (x2,y2) koordinátapár által megadott helyre mozgatja. A paraméterek értéke és jelentése megegyezik az MMOB utasítás paramétereivel.

DETECT n

Az utasítás inicializálja a sprite-sprite, illetve a sprite-háttér ütközések ellenőrzéséhez szükséges regisztereket. Ha n = 1, akkor a sprite-háttér; ha n = 0, akkor a sprite-sprite ütközéseket regisztráló regiszter törlődik. CHECK utasítással együtt lehet használni.

CHECK sn1,sn2 vagy CHECK 0

A CHECK függvény értéke pontosan akkor nulla, ha az sn1 és az sn2 sprite-ok ütköztek, vagy a másik esetben valamelyik sprite ütközött a háttérrel. Az sn1 és az sn2 sprite-ok ütközése azt jelenti, hogy egyszerre ütköztek egy-egy sprite-tal, de nem biztos, hogy egymással.

MOB OFF sn

Kikapcsolja az sn-ik sprite-ot.

MEM

Az utasítás módosítja a memória felosztását és lehetővé teszi programozható karakterek használatát. A képernyő-memória a $C000 címen kezdődik, a sprite-ok definícióját a $F000 címtől tárolhatjuk. Mód nyílik programozható karakterek előállítására a $E000 címtől kezdődően.

Programozható karakterek

A MEM utasítás végrehajtása után a DESIGN és a @ utasítások segítségével módunk van a karakterkészlet átírására. A DESIGN utasítás paraméterét 2-nek adva meg jelezhetjük az interpreternek, hogy karakterdefiníció következik. A karakter alakját a DESIGN utasítást követő @ utasításban kell definiálnunk. A karakterkészlet a $E000 címen kezdődik. A ch sorszámú karakter újradefiniálására a következő utasítást kell végrehajtani:

```
DESIGN 2, $E000+ch*8
```

A <SHIFT-A> lenyomásával Á betűt kapunk a következő programrész futtatása után:

```
10 DESIGN 2,$E000+65*8
20 @.....BB.
30 @..BB.BB.
40 @.BBBB...
50 @BB..BB..
60 @BBBBBB..
70 @BB..BB..
80 @BB..BB..
90 @BB..BB..e
```

8. Hanggenerálás

8.1 Bevezetés

A legtöbb mikroszámítógép lehetőséget biztosít bizonyos hanghatások elérésére. A legegyszerűbbek néhány egyszerű effektuson kívül (például csipognak) másra nem képesek. A C-64-et egy **egy-chipes ADSR-típusú szintetizátorral** látták el tervezői, amely igen bonyolult hanghatások előállítására képes. Ebben a fejezetben a hanggenerátor használatát és az ehhez szükséges zeneelméleti fogalmakat ismertetjük.

A C-64 'zenei képességeit' a **6581**-es chip (**sound interface device**, **SID**) biztosítja. A három, egymástól független hangot megszólaltató szintetizátor kompatibilis a 65xx mikroprocesszor családdal. A SID széles tartományban és folyamatosan biztosítja a hangmagasság (frekcencia), a hangszín (felharmonikusok) és a hangerő szabályozását. Speciális vezérlő áramköreinek működtetése minimális szoftvert igényel. A SID legfontosabb szolgáltatásai a következők:

(i) Három független 'hang' (oszcillátor) a 0-4 kHz-estartományban.
(ii) Hangonként 4 féle (háromszög, fűrészfog, impulzus, fehér zaj) hullámforma
(iii) Három amplitudó-modulátor (tartomány = 48 dB).
(iv) Három burkoló görbe (ADSR) generátor
  Felfutási idő (attack):2 ms-8 s
  Lecsengési idő (decay):6 ms-24s
  Kitartási szint (sustain):0 - csúcs értékig
  Elengedési idő (release):6 ms -24s
(v) Oszcillátor szinkronizáció és körmoduláció
(vi) Programozható szűrő
  Levágási tartomány:30 Hz -12 kHz
  12 dB/oktáv meredekség
  Alul-, felülátereszto, illetve sávszűrő
  Szűrt kimenet
(vii) Hangerő szabályozás
(viii) 2 analóg-digitális potenciométer

A SID chip **29**, egyenként 8-bites **vezérlő regisztert** tartalmaz, ezek értékeinek beállításával, illetve olvasásával lehet a hanggenerátort programozni. A vezérlő regiszterek a memóriában az 54272-54300 ($D400-$D41C) címeken helyezkednek el. Az utolsó négy (54297-54300) regiszter csak olvasható, a többi csak írható regiszter. BASIC-ből a megfelelő POKE, illetve PEEK utasítások végrehajtásával irányíthatjuk a hanggenerátort.

A SID segítségével egyetlen hangot is csak úgy tudunk megszólaltatni, ha legalább 6-7 regiszter tartalmát beállítjuk. Ennek az az oka, hogy a zenei (de általában bármilyen) hangok struktúrája bonyolult. A hang-érzetet számos tényező befolyásol-ja. Ezek közül a legfontosabb a hangerő, ami azonban egyetlen hang esetében is dinamikusan változhat. Az ADSR generátor éppen ezt a dinamikát szabályozza. A hangérzet másik fontos tényezője a megszólaló hang harmónikus struktúrája. Tiszta szinuszos rezgés ugyanis nem létezik, az alaphang felharmonikusai közül több-kevesebb mindig megszólal. Hogy pontosan melyek, azt a hang hullámformája, illetve az alkalmazott szűrés paraméterei határozzák meg. A különböző hangnemekre jellemző, hogy a megszólaló hangoknak milyen a harmónikus struktúrája. A SID chip segítségével négy féle hullámformát definiálhatunk, melyeknek frekvenciáját tetszés szerint megadhatjuk. További lehetőségeink vannak a magas és/vagy alacsony frekvenciák kiszűrésére, a hullámok szuperponálására.

Az alábbi mintapéldánk bemutatja, hogy a fenti összetevők megfelelő beállításával hogyan érhetjük el, hogy a C-64 egyszerű dallamokat játsszon. A megjegyzéseket a program szövegében helyeztük el.

```
100 REM ***********
102 REM * ZENEPROGRAM  *
104 REM ***********
106 :
108 SID=54272: REM A HANGGENERATOR KEZDOCIME
110 FOR L=0 TO 24:POKE SID+L,0:NEXT
112 :
114 POKE SID+5,9:POKE SID+6,0:POKE SID+24,15
116 :
118 REM ************************
120 REM * AZ ITT KOVETKEZO CIKLUS ZENEL  *
122 REM ************************
124 READ HF,LF,DR
126 IF HF<0 THEN RESTORE:GOTO 124
128 POKE SID+1,HF:POKE SID,LF:REM HANGMAGASSAG
130 POKE SID+4,33:REM MEGSZOLAL
132 FOR I=1 TO DR:NEXT:REM KITARTJA A HANGOT
134 POKE SID+4,32:FOR I=1 TO 50:NEXT:REM ELHALLGAT
136 GOTO 124
138 REM ************************
140 REM * A PROGRAM ADATAI (HF,LF,DR)  *
142 REM ************************
144 DATA 16,195,125,21,31,125
146 DATA 16,195,125,21,31,125
148 DATA 25,30,250,25,30,250
150 DATA 16,195,125,21,31,125
152 DATA 16,195,125,21,31,125
154 DATA 25,30,250,25,30,250,
156 DATA 33,135,125,31,165,125
158 DATA 28,49,125,25,30,125
160 DATA 22,96,250,28,49,250
162 DATA 25,30,125,22,96,125
164 DATA 21,31,125,18,209,125
166 DATA 16,195,250,16,195,250
168 DATA 0,0,250,-1,-1,-1
```

8.2 A SID lehetőségeinek ismertetése

Hangerő szabályozás

A SID chip 24-es regiszterének alsó négy bitje határozza meg a hangerő nagyságát, melynek értéke így a 0-15 intervallumba esik. A hangerőt a következő POKE utasítással állíthatjuk be:

```
POKE 54272+24,X
```

Az alaphang frekvenciája

A hang nem más, mint a levegőben terjedő hullámmozgás. Ennek a hullámmozgásnak az egymást követő hullámcsúcsai közt eltelt időt a hullám periódusidejének hívjuk. Ennek a számnak a reciprok értéke adja meg a másodpercenkénti periódusok számát, vagyis a frekvenciát. A hang magasságát, illetve mélységét a hanghullámok frekvenciája határozza meg. Egy hang általában nem egyetlen színuszos rezgésből áll, hanem ennek a rezgésnek a felharmonikusai – esetleg különböző súlyokkal – is megszólalnak. A SID megfelelő regiszterei tartalmának beállításával lehetőségünk van az alaphang frekvenciájának beállítására, illetve a hullámforma kiválasztásával és a szűrők programozásával beállítani, hogy melyik felharmonikusok szólaljanak meg.

A SID chip mindhárom hang számára két-két regisztert biztosít, amelyek segítségével az alaphang frekvenciáját szabályozni lehet. A regiszter-párok 16-bites egész számokat tárolnak, amelyek a következő képlet segítségével adják meg az oszcillátorok frekvenciáját

C = Fout/0.06097

(1 MHz-es órajel esetén, C a tárolt 16-bites szám, Fout az oszcillátoron megjelenő frekvencia valódi értéke.)

A függelékben nyolc oktáv hangjaira adtuk meg azokat az értékpárokat, melyeket a szóban forgó regiszterekben tárolni kell.

Az egyes hangok (oszcillátorok) esetén a következő regiszterek szolgálnak a frekvencia megadására:

Funkció	Regiszter	Cím
1. hang frekvenciájának beállítása		
alsó byte	0	54272
felső byte	1	54273
2. hang frekvenciájának beállítása		
alsó byte	7	54279
felső byte	8	54280
3. hang frekvenciájának beállítása		
alsó byte	14	54286
felső byte	15	54287

Ha már tudjuk egy adott frekvencia előállításához tartozó C értéket, akkor annak alsó (A) és felső (F) byte-jait a következő utasításokkal számíthatjuk ki:

F = INT(C/256): A = C-256*F

Ha tehát a 2. hangot (oszcillátort) a normál A hang megszólaltatására akarjuk programozni, akkor a következő utasításokat kell végrehajtanunk:

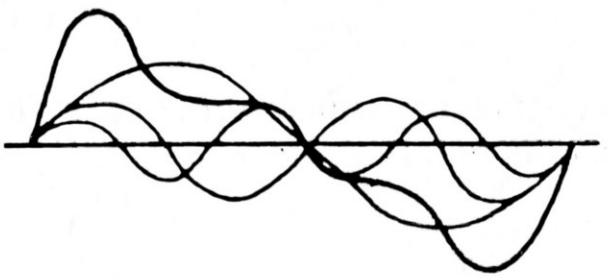
C = 440/0.05966: F = INT(C/256)

POKE 54280,F: POKE 54279,C-256*F

(A hang természetesen még nem szólal meg.)

A hullámforma kiválasztása

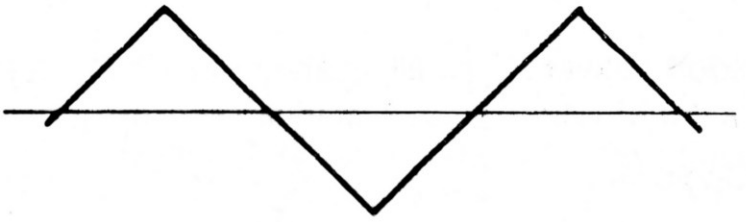
A zenei hangzás minőségét a hangszín, azt pedig a megszólaló hang harmónikus struktúrája határozza meg. Az alapharmonikus határozza meg a hang magasságát, illetve a hangnak megfelelő hullám periódusidejét. A felharmonikus hullámok frekvenciái az alapharmonikusok egész számú többszörösei. A többszörös nagyságának értékétől függően beszélünk második, harmadik stb. felharmonikusról. Az alapharmonikust néha első harmonikusnak is hívják. A következő ábra bemutatja, hogy három felharmonikus esetén milyen eredő hullámformát kapunk:



A legbonyolultabb harmonikus strukturával az akusztikus hangszerek, mint például a gitár vagy a hegedű rendelkeznek.

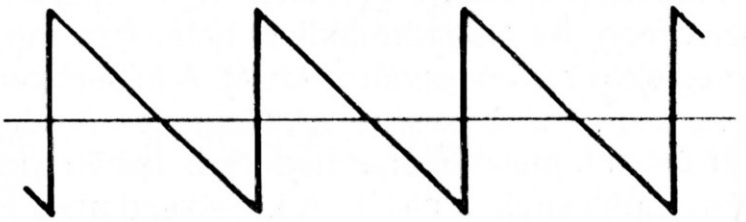
A következőkben összefoglaljuk, hogy milyen a háromszög-, a fűrészfog- és a négyszöghullám, valamint az ún. 'fehér zaj' felharmonikus struktúrája.

a/ A **háromszög-hullám** csak páratlan felharmonikusokat tartalmaz. A jelenlevő harmonikusok amplitúdója fordítottan arányos a harmonikus szám négyzetével. Más szóval a 3. harmonikus amplitúdója 9-szer kisebb, mint az első harmonikus amplitúdója. (Ez nem jelenti azt, hogy 9-szer halkabbnak halljuk, mert a fül hangérzékelése logaritmikus.)



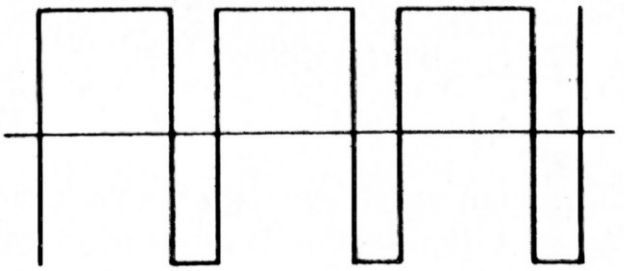
Háromszög-hullám

b/ A **fűrészfog-hullám** az összes felharmonikust tartalmazza. A harmonikusok amplitúdója a felharmonikus sorszámával fordítottan arányos. Például a 2. felharmonikus amplitúdója feleakkora, mint az elsőé.



Fűrészfog-hullám

c/ A **négyszög-hullám** csak páratlan harmonikusokat tartalmaz, amelyek amplitúdója fordítottan arányos a felharmonikus sorszámával.



Négyszög-hullám (nem szimmetrikus)

A **nem szimmetrikus** négyszög-hullám pozitív és negatív részek arányától függően igen sokféle hangérzetet kelthet; a felharmonikusok struktúrájában az egyes harmonikusok súlya más és más.

A hullámforma alkalmas megválasztásával olyan felharmonikus struktúrát alakíthatunk ki, amely megközelíti a kívánt hangzást. Ennek finomítására további eszközök – például a szűrés – áll rendelkezésre. (Az elméletileg végtelen sok felharmonikusból természetesen csak a 0-4 kHz tartományba esők szólalhatnak meg.)

d/ A fenti 'zene' jellegű hullámformákon kívül még rendelkezésünkre áll az úgynevezett **'fehér zaj' hullámforma**:

Fehér zaj

Ezt a különböző zajeffektusok (lövés, tengerzúgás stb.) előállítására használjuk.

Azt, hogy egy hang milyen harmonikus struktúrával szólaljon meg, az egyes hangokhoz tartozó kontroll-regiszter megfelelő bitjeinek magasra állításával adhatjuk meg. A négyszög-impulzus esetén még az impulzus szélességét is külön kell definiálnunk. Az egyes hangokhoz tartozó kontrol-regiszterek a következők:

	Regiszter	Cím
1. hang kontrol-regiszter	4	54276
2. hang kontrol-regiszter	11	54283
3. hang kontrol-regiszter	18	54290

A 7.-4.bitek határozzák meg, hogy melyik hullámformát használjuk a felsorolt négy közül:

Hullámforma	**Bit** (1 = bekapcsolva)
fehér zaj	7
négyszögimpulzus	6
fűrészfog	5
háromszög	4

Amennyiben több bitet is magasra (1) állítottunk, az eredő a kiválasztott hullámformák 'logikai és'-e lesz. Ezzel érdekes hanghatásokat érhetünk el, de az is előfordulhat, hogy a SID chip 'kiakad' és csak újraindítani lehet.

A négyszögimpulzus választása esetén az impulzus pozitív részének a teljes periódushoz való arányát is meg kell adnunk. Erre hangonként további 2 byte szolgál, a felső byte-nak azonban csak az alsó négy (0.-3.) bitje számít. Ha az ezáltal definiált 12 bites egész szám értéke S, akkor a valódi arányt a

$$PWout = \frac{S}{40.95} \text{ (\%-ban!)}$$

összefüggés adja meg. S = 0 esetén tehát konstans DC kimenetet kapunk, S = 2048 esetén pedig egy szabályos négyszögjelet (az arány 50%).

Az egyes oszcillátoroknak megfelelő regiszterek a következők:

	Regiszter	**Cím**
1. hang impulzusszélesség beállítás		
alsó byte	2	54274
felső byte	3	54275
2. hang impulzusszélesség beállítás		
alsó byte	9	54281
felső byte	10	54282
3. hang impulzusszélesség beállítás		
alsó byte	16	54288
felső byte	17	54289

Mint említettük, a felső byte-ok bitjei közül csak az alsó négy számít. Ennek alapján, ha például a harmadik oszcillátoron szabályos négyszögimpulzust akarunk kapni, akkor a következő utasítást kell végrehajtanunk:

```
POKE 54290,PEEK(54290) OR 64
POKE 54288,255:POKE 54289,15
```

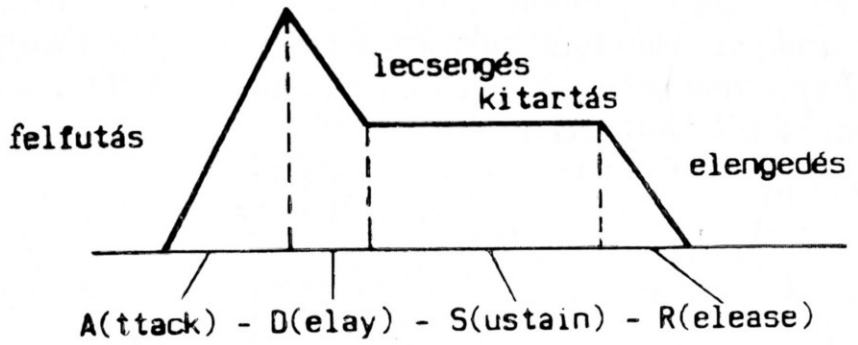
(A hang természetesen nem szólal meg. Az oszcillátor frekvenciáját is külön kell beállítanunk.)

8.3 A burkológörbe (ADSR) generátor

Az ADSR működése

A zenei hang hangereje változik attól kezdve, hogy meghalljuk, egészen addig, mig el nem hal. Először (pl. a zongorabillentyű leütésekor) a hangerő felfut a csúcsértékre. Azt az időt, ami alatt ez lezajlik, nevezzük **felfutási** időnek (attack time). Ezután a hangerő leesik egy középszintre. Azt az időt, ami alatt ez végbemegy, nevezzük **lecsengési** időnek (decay time). Ezen a szinten halljuk általában legtovább a hangot, ezt ezért **kitartási** időnek (sustain time) hívjuk. Végül a hang erről a közbenső szintről (sustain level) nullára esik. Azt az időt, ami alatt ez megtörténik, **elengedési** időnek (release time) hívjuk. Az egyes hangszerekre (és a játszási módra) jellemzők a fenti idők.

A C-64 lehetőséget ad mind a négy időintervallum programozására. A paraméterek angol nyelvű megnevezése alapján a burkológörbe generátort ADSR-generátornak is hívjuk. Az ADSR-generátorok az elektromos zenében optimálisnak bizonyultak a flexibilitás és az egyszerű amplitúdó vezérlés tekintetében.



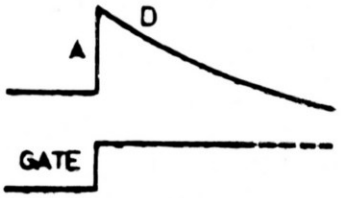
Megfelelően kiválasztott burkológörbe paraméterek lehetővé teszik az ütő- és az elnyújtott hangú hangszerek széles skálájának szimulálását. A hegedű jó példa az elnyújtott hangú hangszerre. A hegedűs a hangerőt a vonó húzásával szabályozza. A hangerő lassan nő a csúcsértékig, majd egy közbenső szintig csökken. A hegedűs ezt a szintet tudja tartani a kívánt ideig, majd a hang lassan elhal. Az ennek megfelelő burkológörbét könnyen előállíthatjuk az ADSR-rel a következő paraméterek beállításával:

felfutás:	500 ms	(10)
lecsengés:	300 ms	(8)
kitartás:		(10)
elengedés:	750 ms	(9)

Megjegyezzük, hogy a hang a közbenső szinten kitartható. A hang addig nem kezd elhalni, amig a GATE (kapuzás) bitet nem töröljük. Kis változtatásokkal a fenti burkológörbe használható réz- és fafuvós, illetve egyéb vonós hangszerek szimulálására is.

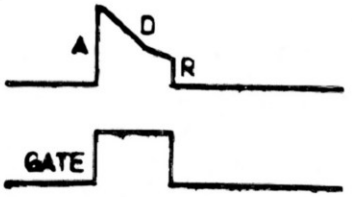
Egy teljesen eltérő burkológörbe alakot kapunk az ütőhangszereknél, mint amilyen például a dob, a cimbalom és a csengő, csakúgy, mint némely billentyűs hangszernél, például a zongoránál és a csembalónál. Az ütőhangszerek burkológörbéje egy szinte pillanatszerű felfutásból és egy azt közvetlenül követő, zérus értékre történő lecsengésből áll. Az ütőhangszerekkel nem lehet kitartani egy közbenső szintet. Például a dob megütésének pillanatában eléri a teljes hangerőt és gyorsan lecseng, függetlenül a megütés módjától. A cimbalom tipikus burkológörbéje látható alább:

felfutás: 2 ms (0)
lecsengés: 750 ms (9)
kitartás: (0)
elengedés: 750 ms (9)



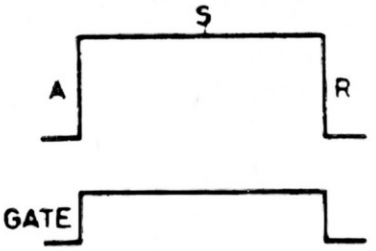
Megjegyzendő, hogy a hang közvetlenül a csúcsérték elérése után elkezd lecsengeni nulla amplitúdóra, függetlenül a GATE bittől. A zongorák és csembalók burkológörbéje kicsit bonyolultabb, de könnyen előállítható ADSR-rel. Ezek a hangszerek rögtön elérik a teljes hangerőt a billentyű leütése után. Ezután az amplitúdó azonnal elkezd lassan lecsengeni, amíg a billentyűt lenyomva tartjuk. Ha a billentyűt elengedjük, mielőtt a hang teljesen elhalt volna, akkor az amplitudó hirtelen leesik zérusra. Ilyen burkológörbét láthatunk a következő ábrán:

felfutás: 2 ms (0)
lecsengés: 750 ms (9)
kitartás: (0)
elengedés: 6 ms (0)



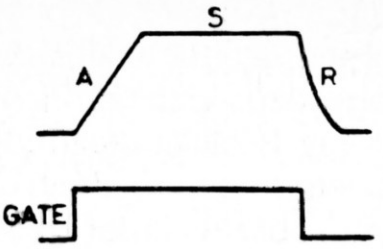
Jegyezzük meg, hogy a hang lassan cseng le, amíg a GATE bitet nem töröljük, majd az amplitúdó gyorsan leesik zérusra. A legegyszerűbb burkológörbéje az orgonának van. Ha megnyomjuk a billentyűt, rögtön eléri a teljes hangerőt és ott is marad. Ha elengedjük a billentyűt, a hang rögtön zérus amplitudójúra csökken. Ilyen burkológörbét ábrázol a következő rajz:

felfutás: 2 ms (0)
lecsengés: 6 ms (0)
kitartás: (15)
elengedés: 6 ms (0)



A SID alkalmazásának legnagyobb előnye abban áll, hogy az akusztikus hangszereket nem szimulálja, hanem majdnem eredeti hangon szólaltatja meg. Az ADSR-rel lehet olyan burkológörbéket is előállítani, amelyeket 'valódi' hangszerrel nem tudunk létrehozni. Jó példa erre a 'visszajátszott' burkológörbe. Ezt a görbét egy lassú felfutással, hosszú kitartással és gyors elengedéssel jellemezhetjük, és olyan hangot ad, mint amikor egy hangszer hangját magnetofon szalagra felvesszük és visszafelé játszuk le. Burkológörbéje a következő:

felfutás: 500 ms (10)
lecsengés: 6 ms (0)
kitartás: (15)
elengedés: 72 ms (3)



Sok különös hang hozható létre, ha egy hangszer burkológörbéjét egy másik hang harmonikus struktúrájával moduláljuk. Ez a többszólamú akusztikus hangszerek hangjára emlékeztet, de észrevehető a különbség. Általában véve a hang szubjektív dolog, és a különböző burkológörbékkel és felharmonikus strukturával végzett kísérletek szükségesek a kívánt hanghatás eléréséhez.

A burkológörbe generátor programozása

Az ADSR ciklus minden egyes elemének idejét külön-külön lehet programozni. Az egyes paraméterek a 0-15 intervallumba esnek (4 bit) és jelentésük a következő:

Érték	Felfutási idő	Lecsengési/elengedési idő
0	2 ms	6 ms
1	8 ms	24 ms
2	16 ms	48 ms
3	24 ms	72 ms
4	38 ms	114 ms
5	56 ms	168 ms
6	68 ms	204 ms
7	80 ms	240 ms
8	100 ms	300 ms
9	250 ms	750 ms
10	500 ms	1,5 s
11	800 ms	2,4 s
12	1 s	3 s
13	3 s	9 s
14	5 s	15 s
15	8 s	24 s

Megjegyezzük, hogy a fenti idők 1 MHz-es Φ2 órajelre lettek kiszámítva. Más frekvencia esetén a fenti értékeket 1/2-vel meg kell szorozni.

Az ADSR generátor vezérlését hangonként (oszcillátoronként) 2 regiszter, valamint a kontroll-regiszter 0.(GATE) bitjének segítségével végezhetjük el. A regisztereket AD és SR regiszteknek hívjuk. A név megfelel a tartalomnak, az AD regiszter felső, illetve alsó 4-4 bitje a felfutási, lecsengési idő paraméterét tartalmazza. Hasonlóan az SR regiszter felső és alsó 4-4 bitje a közbenső szint, illetve az elengedési idő paraméterét tartalmazza.

A GATE bit az AD, illetve az SR ciklusok kezdetét szabályozza. Ha a GATE bitet magasra állítjuk (ahogy ezt előző ábráink is jelezték), egy AD ciklus indul el; ha nullára állítjuk, akkor egy R ciklus kezdődik. Az AD ciklus lefutása után a hangerő addig a közbenső szinten marad, amíg a GATE bitet nullára nem állítjuk. Így a kitartási idő akármilyen hosszú is lehet. Az SR regiszterben tárolt S érték **nem a kitartási időt, hanem a közbenső szintet** adja meg. S lehetséges értékei (0-15) az amplitúdó nagyságát határozzák meg, a közbenső szint amplitúdója az amplitúdó-csúcs S/16 része. A hegedű példája esetén az AD regiszterbe 10*16+8=168, az SR regiszterbe 10x16+9=169 értékeket kell beírnunk.

Az egyes oszcillátorok ADSR-t vezérlő regiszterei a következők:

		Regiszter	Cím
1. hang	AD-regiszter	5	54277
	SR-regiszter	6	54278
	kontrol-regiszter	4	54276
2. hang	AD-regiszter	12	54284
	SR-regiszter	13	54285
	kontrol-regiszter	11	54283
3. hang	AD-regiszter	19	54291
	SR-regiszter	20	54292
	kontrol-regiszter	18	54290

GATE-bit = kontrol-regiszter 0. bitje.

8.4 A SID chip további lehetőségei

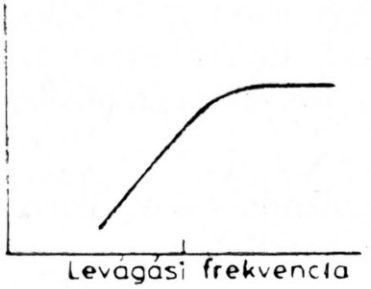
Szűrés

Az egyes hangok felharmonikus struktúrájának változtatására szűrőt is használhatunk. A SID chip háromféle szűrővel van ellátva, ezeket egymástól függetlenül vagy egymással kombinálva is használhatjuk.

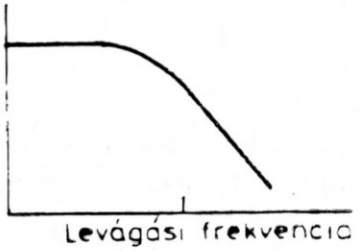
A szűrő lényege, hogy bizonyos frekvencia-tartományba eső hullámokat 'kiszűr', pontosabban amplitúdójukat – a szűrő típusától függő mértékben – csökkenti (esetleg nullára!).

A C-64 SID chipje felül-, alul-, illetve sáváteresztő szűrővel rendelkezik. A szűrők programozhatók, ami azt jelenti, hogy a szűrő működését jellemző levágási frekvencia a szoftverből adható meg.

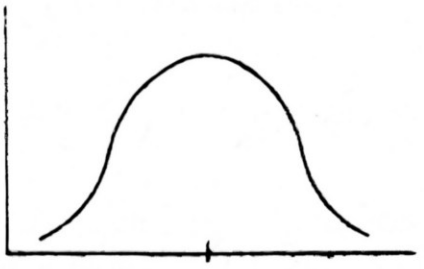
/a/ A felüláteresztő szűrő a levágási frekvencia felett levő hangokat átereszti, míg az alatta levőket levágja. Hogy milyen arányban, azt az alábbi ábra mutatja:



/b/ A SID áramkör aluláteresztő szűrője,mint a neve is mutatja, a levágási frekvencia alatti hangokat átereszti, míg az a fölöttieket levágja.



/c/ Végül az áramkör egy sáváteresztő szűrővel is rendelkezik, amely átenged egy szűk frekvenciasávot a levágási frekvencia környékén, a többi hangot pedig levágja.



/d/ Az alul- és felüláteresztő szűrők sávzáró szűrővé kombinálhatók, amely nem engedi át a letörési frekvencia szűk környezetében, míg a többi hangot átengedi.

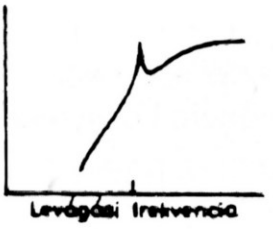
A SID chip 21-24. sorszámú regiszterei (54293-54296) irányítják a szűrőt.

A 21.-22. regiszterek együtt a szűrő levágási frekvenciáját vezérlik. A 21. regiszter 0-2 bitjei, illetve a 22. regiszter mind a 8 bitje egy 11-bites egész számot határoz meg (a 22. regiszter bitjei a magasabbak). Ez az egész szám lineárisan vezérli a szűrő levágási frekvenciáját. A minimális, illetve maximális értéknek közelítőleg 30 Hz és 12 kHz felel meg.e

A 23. regiszter adja meg a szűrő rezonanciáját, illetve határozza meg, mely hangok (oszcillátorok) kimenő jele kerüljön szűrésre.

0.bit: Ha alacsony, az 1. hang szűrés nélkül jelenik meg a kimeneten, ha magas (1) az oszcillátor kimenő jele átmegy a szűrőn, és annak megfelelően módosul.
1.bit: Ugyanez a 2. hangra vonatkozóan.
2.bit: Ugyanez a 3. hangra vonatkozóan.
3.bit: Ugyanez az audio bemeneten (26. láb) megjelenő jelre vonatkozóan.

A 23. regiszter 4.-7. bitjei a szűrő rezonanciáját szabályozzák. A rezonancia olyan hatást okoz, ami kiemeli a levágási frekvencia környezetében levő hangokat, erősebb hangot okozva ezáltal. A rezonanciát a fél-byte-nak megfelelő 0-15 intervallum lineárisan szabályozza: 0 esetén nincs, 15 esetén pedig maximális a rezonancia. Ez utóbbi esetben például a felüláteresztő szűrő az alábbi ábrán látható módon dolgozik:



A 24. regiszter 0.-3. bitjei a hangerőt szabályozzák. A további bitek a szűrés módját határozzák meg.

4.bit: Ha magasra állítjuk, akkor a szűrő aluláteresztő módban kezd el működni és az így megszűrt jel kerül a kimenetre. A levágási frekvencia fölött a csillapítás közelítőleg 12 dB/oktáv. Az aluláteresztő szűrő telt hangokat eredményez.
5.bit: Ha magasra állítjuk, akkor a szűrő sáváteresztő módban kezd el működni. A csillapítás közelítőleg 6 dB/oktáv. A sáváteresztő szűrő esetén vékony, tiszta hangokat kapunk.
6.bit: Ha magasra állítjuk, akkor a szűrő felüláteresztő üzemmódban kezd el dolgozni. A levágási frekvencia alatti tartományban a csillapítás 12 dB/oktáv. A felüláteresztő szűrő bádoghangot ad.
7.bit: Ha magasra állítjuk, a 3. hangot leválasztjuk a közvetlen audio kimenetről. Ha ezzel együtt a 3. hang a szűrőn nem halad keresztül, akkor semmilyen hallható hatást nem okoz, és így modulációs célokra használhatjuk. (Elmaradnak a nem kívánt kimenő jelek.)

Az egyes szűrő üzemmódok egyszerre is kiválaszthatók. Ha azt akarjuk, hogy a szűrőre adott jelek meg is szólaljanak, legalább egy szűrő üzemmódot be kell kapcsolnunk a megfelelő bit magasra állításával.

Szinkronizálás és modulálás

Az egyes hangok kimenő jelét szinkronizálhatjuk, illetve modulálhatjuk egy-egy másik hang (oszcillátor) kimenő jelével. Azt, hogy ezt melyik hanggal tehetjük meg, azt a hang sorszáma dönti el:

Hang	Moduláló/szinkronizáló hang
1	3
2	1
3	2
I	I-1 mod 3

A szinkronizálás, illetve modulálás mindaddig nem következik be, míg a hangok kontrol-regisztereinek megfelelő bitjeit magasra nem állítottuk.

1. **bit**: Ha ezt a bitet magasra állítjuk, a SID chip szinkronizálja az oszcillátor alapfrekvenciáját a szinkronizáló oszcillátor alapfrekvenciájával, és így 'keményszinkron' ('hard sync.') effektust kapunk. A chip a szinkronizáló oszcillátor figyelembevételével változtatja az oszcillátor frekvenciáját; a szinkronizáló oszcillátor frekvenciáján a hangösszetevő komplex harmonikus struktúrájának széles tartományát kapjuk. Azért, hogy létrejöjjön a szinkronizáció, a szinkronizáló oszcillátort zérustól különböző, tetszőleges, de az oszcillátor frekvenciájánál kisebb frekvenciájúra kell beállítanunk. A szinkronizáló hang más paramétereinek nincs hatása a szinkronizációra.

2. **bit**: A bitet magasra állítva az oszcillátor háromszög hullám kimenetének helyét átveszi az oszcillátor és a moduláló oszcillátor 'körmodulált' kombinációja. Nem harmonikus felhang strukturát kapunk, mely csengő- vagy gonghangot, illetve speciális effektusokat eredményez. Azért, hogy a körmoduláció hallható legyen, az oszcillátor háromszögjelét kell kiválasztani és a moduláló oszcillátort tetszőleges, nullától különböző frekvenciájúra kell beállítani. A moduláló hang más paramétereinek nincs hatása a körmodulációra.

3. **bit**: A bitet magasra állítva az oszcillátort töröljük és lezárjuk. Az oszcillátor zaj kimenetét is letiltjuk, míg az impulzus kimenet DC szinten marad. Ezt a bitet általában tesztelési célokra használják, bár felhasználható az oszcillátor és külső jelek szinkronizálására is, lehetővé téve igen komplex jelalakok kialakítását real-time szoftver vezérlés mellett.

Potenciométerek

A 25. regiszter a SID chip POTX (24. láb) bemenetéhez kapcsolt potenciométer mindenkori állását tartalmazza. 0 a minimális, 255 a maximális ellenállás értéknek felel meg. A vezérlés lineáris.

A 26. regiszter a SID chip POTY (23. láb) bemenetéhez kapcsolt potenciométer hasonló adatát tartalmazza.

A két regiszter minden 512-ik 2 órajelre aktualizálódik. A regiszterek csak olvashatók.

Véletlenszám generátor

A 27. regiszterből a mikroprocesszor bármikor kiolvashatja a 3. oszcillátor kimeneti értékének felső 8 bitjét. Az így beolvasott szám közvetlenül a kiválasztott jelalakkal kapcsolatos. Ha a 3. oszcillátor fűrészfog-jelét választjuk kimenetként, akkor a regiszterben a 0 és 255 ($FF) közötti számok jelennek meg egymás után növekvő sorrendben; a növekedési sebességet a 3. oszcillátor frekvenciája határozza meg. Ha a háromszögjelet választjuk ki, akkor a kimeneti regiszter tartalma 0-tól 255-ig nő, majd onnan 0-ig csökken. Ha az impulzus jelalakot választjuk ki, a kimenet 0 és 255 között fog ugrálni. A zaj-kimenetet kiválasztva egy véletlen számsorozatot kapunk, és ekkor ezt a regisztert véletlenszám-generátorként használhatjuk fel. A 3. oszcillátor regiszternek számos időzítési és sorrendi alkalmazása van, de talán a legfőbb funkciója a modulálás. A regiszterből beolvasott számot szoftver segítségével real-time módon hozzáadhatjuk az oszcillátor vagy a szűrő frekvencia vagy impulzus szélesség regiszteréhez. Ily módon különböző dinamikus hatásokat érhetünk el. Szirénaszerű hangot kapunk, ha a 3. oszcillátor fűrészfog-jel kimenetét egy másik oszcillátor frekvencia vezérlő regiszteréhez hozzáadjuk. Érdekes jelenség áll elő, ha a 3. oszcillátor zaj-kimenetét hozzáadjuk a szűrő frekvencia vezérlő regiszteréhez. Vibrátor keletkezik, ha a 3. oszcillátor frekvenciáját 7 Hz körüli értékre állítjuk be és a háromszögjel kimenetét hozzáadjuk (megfelelő skálázással) egy másik oszcillátor frekvencia-vezérlő regiszteréhez. Számtalan effektust hozhatunk létre a 3. oszcillátor frekvenciájának és a háromszögjel skálázásának változtatásával. Általában, ha a 3. oszcillátort modulációra használjuk, akkor a 3. hangösszetevőt nem engedjük az audio kimenetre.

A 28. regiszter a 27. regiszter megfelelője azzal a különbséggel, hogy ilyenkor a mikroprocesszor a 3. hangösszetevőhöz tartozó burkológörbe generátor kimenetét tudja ebből a regiszterből kiolvasni. Ezt a kimenetet hozzáadhatjuk a szűrő frekvenciájához, hogy harmonikus burkológörbét, WAH-WAH és ehhez hasonló jelenségeket kapjunk. 'Fázis' hangokat kapunk, ha ezt a kimenetet az egyik oszcillátor frekvencia-vezérlő regiszteréhez adjuk hozzá. A 3. hangösszetevőhöz tartozó burkológörbe-generátor GATE bitjét magasra kell állítani, hogy a regiszterben megjelenjen a kimenet értéke. A 27. regiszter tartalma viszont mindig változik az oszcillátor kimenetétől függően és természetesen nincs rá hatással a burkológörbe generátor.

8.5 Példák

A fejezet elején bemutattunk egy programot, amelynek DATA utasításait megváltoztatva már tetszőleges dallamot le tudunk játszani. Először azt nézzük meg, hogyan változik a hangszín, ha az ADSR-regiszterek tartalmát, illetve a hullámformát megváltoztatjuk.

(i)
```
 10 SID=54272: REM A HANGGENERATOR KEZDOCIME
 20 FOR L=0 TO 24 : POKE SID+L,0:NEXT
 30 POKE SID+5,9:POKE SID+6,0:POKE SID+24,15
 40 READ HF,LF,DR
 50 IF HF<0 THEN RESTORE:GOTO 40
 60 POKE SID+1,HF:POKE SID,LF:REM HANGMAGASSAG
 70 POKE SID+4,17:REM MEGSZOLAL
 80 FOR I=1 TO DR:NEXT:REM KITARTJA A HANGOT
 90 POKE SID+4,16:FOR I=1 TO 50:NEXT:REM ELHALLGAT
100 GOTO 40
110 DATA 16,195,125,21,31,125
120 DATA 16,195,125,21,31,125
130 DATA 25,30,250,25,30,250
140 DATA 16,195,125,21,31,125
150 DATA 16,195,125,21,31,125
160 DATA 25,30,250,25,30,250
170 DATA 33,135,125,31,165,125
180 DATA 28,49,125,25,30,125
190 DATA 22,96,250,28,49,250
200 DATA 25,30,125,22,96,125
210 DATA 21,31,125,18,209,125
220 DATA 16,195,250,16,195,250
230 DATA 0,0,250,-1,-1,-1
```

Programunk csak a 70., illetve a 100. programsorban tér el az eredetitől, mégis a hangszín egészen más, kevésbé pengő, tompább. Ennek az az oka, hogy a fűrészfog jelet háromszög jelre cseréltük.

Hegedűszerű hang előállítása céljából írjuk át az eredeti program 30. sorát:

```
30 POKE SID+5,88:POKE SID+6,89:REM A=5,D=8,R=9, S=5
```

Változtassuk meg a hullámformát háromszög-jelre, és xilofonszerű hangot kapunk:

```
30 POKE SID+5,9:POKE SID+6,9 REM A=0,D=9,R=9,S=0
70 POKE SID+4,17
100 POKE SID+4,16:FOR T=1TO50:NEXT
```

Változtassuk a hullámformát négyszög-jelre és zongoraszerű hangzást kapunk:

```
25 POKE SID+3,8:POKE SID+2,0
30 POKE SID+5,9:POKE SID+6,0:REM A=0,D=9,S=0,R=0
70 POKE SID+4,65
100 POKE SID+4,64:FOR T=1TO50:NEXI
```

A legizgalmasabb hangzások azonban azok, amelyek nem meglevő hangszereket akarnak utánozni, hanem új 'hangszereket' hoznak létre. Próbáljuk ki a következőt:

```
20 POKE SID+5,144:POKE SID+6,243:REM A=9,D=0,S=15,R=3
```

```
(ii)  10 SID=54272:REM A HANGGENERATOR KEZDOCIME
      20 FOR L=0 TO 24:POKE SID+L,0:NEXT
      25 POKE SID+22,128:POKE SID+21,0:POKE SID+23,1
      30 POKE SID+5,9:POKE SID+6,0:POKE SID+24,79
      40 READ HF,LF,DR
      50 IF HF<0 THEN RESTORE:GOTO 40
      60 POKE SID+1,HF:POKE SID,LF:REM HANGMAGASSAG
      70 POKE SID+4,33:REM MEGSZOLAL
      80 FOR I=1 TO DR:NEXT:REM KITARTJA A HANGOT
      90 POKE SID+4,32:FOR I=1 TO 50:NEXT:REM ELHALLGAT
     100 GOTO 40
     110 DATA 16,195,125,21,31,125
     120 DATA 16,195,125,21,31,125
     130 DATA 25,30,250,25,30,250
     140 DATA 16,195,125,21,31,125
     150 DATA 16,195,125,21,31,125
     160 DATA 25,30,250,25,30,250
     170 DATA 33,135,125,31,165,125
     180 DATA 28,49,125,25,30,125
     190 DATA 22,96,250,28,49,250
     200 DATA 25,30,125,22,96,125
     210 DATA 21,31,125,18,209,125
     220 DATA 16,195,250,16,195,250
     230 DATA 0,0,250,-1,-1,-1
```

Ez a program az elsőtől csak a 25. és 30. sorban különbözik. Ebben beállítjuk a szűrő levágási frekvenciáját és engedélyezzük az első hang szűrését. A 30. sorban a SID + 24. címen levő regiszterben nemcsak a hangerőt állítjuk be, hanem kiválasztjuk a felülateresztő szűrést is. A szűrő az alacsony frekvenciákat kiszűri, ezért a hang fémes jellegű lesz. Érdemes más levágási frekvenciákkal és a többi szűrővel is kísérletezni!

```
(iii) 10 SID=54272
      20 FOR L=0 TO 24:POKE SID+L,0:NEXT
      30 POKE SID+14,5
      40 POKE SID+18,16
      50 POKE SID+3,1
      60 POKE SID+24,143
      70 POKE SID+6,240
      80 POKE SID+4,65
      90 FR=5389
     100 FOR T=1 TO 200
     110 FQ=FR+PEEK(SID+27)*3.5
     120 HF=INT(FQ/256):LF=FQ-HF*256
     130 POKE SID,LF:POKE SID+1,HF
     140 NEXT
     150 POKE SID+24,0
```

A program a SID 27. regiszterének használatát mutatja be. A 3. hangot az 1. hang frekvenciájának a programból (tehát szoftverből) történő modulálására használjuk. Ezért a 24. regiszter megfelelő (7.) bitjét is magasra állítjuk, hogy a 3. hang audio kimenetét kikapcsoljuk. A 27. regiszter olvasása és az első hang frekvenciájának módosítása a 110. és 120. sorokban történik meg. A program szirénaszerű hangot generál.

```
(iv)  10 SID=54272
      20 FOR L=0 TO 24:POKE SID+L,0:NEXT
      30 POKE SID,240:POKE SID+1,33
      40 POKE SID+5,8
      50 POKE SID+22,104
      60 POKE SID+23,1
      70 POKE SID+24,79
      80 FOR N=1 TO 15
      90 POKE SID+4,129
     100 FOR T=1 TO 250:NEXT:POKE SID+4,128
     110 FOR T=1 TO 30:NEXT:NEXT
     120 POKE SID+24,0
```

A zaj-generátort a legkülönfélébb hangeffektusok előállítására használhatjuk. Az első hang zaj-generátorának kimenetét még a felüláteresztő szűrővel is megszűrjük, így közelítőleg tapshatást érünk el.

```
(v)   10 SID=54272
      20 FOR L=0 TO 24:POKE SID+L,0:NEXT
      30 POKE SID+1,100
      40 POKE SID+5,219
      50 POKE SID+15,28
      60 POKE SID+24,15
      70 POKE SID+4,19
      80 FOR T=1 TO 5000:NEXT
      90 POKE SID+4,18
     100 FOR T=1 TO 1000:NEXT:POKE SID+24,0
```

Ötödik példánk szúnyogdongás-szerű hangot ad. Ennek elérése érdekében az 50. sorban a 3. hang frekvenciáját is be kellett állítanunk, továbbá a 70. és 90. utasításokban a szinkronizálást engedélyező első bitet is magasra állítottuk.

```
(vi)  10 SID=54272
      20 FOR L=0 TO 24:POKE SID+L,0:NEXT
      30 POKE SID+1,130
      40 POKE SID+5,9
      50 POKE SID+15,30
      60 POKE SID+24,15
      70 FOR L=1 TO 12:POKE SID+4,21
      80 FOR T=1 TO 1000:NEXT:POKE SID+4,20
      90 FOR T=1 TO 1000:NEXT:NEXT
```

Körmodulációra (vagy lebegtetésre) mutat példát ez a program. A 70. sorban a 2. bit magasra állításával az első hang (oszcillátor) kimenetén az 1. és 3. hang körmodulált kimeneti jele jelenik meg. Ilyen módon nem-harmonikus felhang-struktúrát kapunk, amely harang vagy gong hangzások előállítására szolgál. Fenti programunk egy harangjátékot imitál.

8.6 SIMONS' BASIC SID utasításai

Bevezetés

A SIMONS' BASIC néhány egyszerű utasítást tartalmaz, amelyek megkönnyítik a SID chip programozását. Szemben a SIMONS' BASIC grafikus utasításaival, a hanggenerálást segítő utasítások csak többszólamú dallamok lejátszását teszik lehetővé, és nem támogatják a SID többi lehetőségét (pl. szűrés). Röviden ismertetjük ezeket az utasításokat.

VOL n

Az utasítás a SID hangerejét n-re állítja be. Értelemszerűen n értékének a 0-15 intervallumba kell esnie.

WAVE <hang>,<bit sorozat>

Az utasítás a <hang>-hoz tartozó kontroll-regiszter bitjeit állítja be. A bit-sorozat nyolc, 0-ból és/vagy 1-esből álló sorozat, amelyik a kontroll-regiszter bitjeinek felel meg. Ennek megfelelően az utasítással kiválaszthatjuk a hullámformát, beállíthatjuk a GATE bitet és szabályozhatjuk a szinkronizálást. A sorozat első bitje a kontroll-regiszter 7., utolsó bitje pedig a 0. bitjének felel meg.

MUSIC n, <zene-sztring>

Az utasítás definiálja a lejátszandó zenét. A zene-sztring a kotta megfelelően kódolt formáját tartalmazza.

Egy hangjegyet <hangjegy> <oktáv> <tartam> alakban kell megadnunk. A <hangjegy> értéke C, D, E, F, G, A, B lehet. A félhangokat az előtte levő hang sifte-lésével kapjuk. Az <oktáv> értéke 0-8 lehet. A <tartam> a megszólaló hang hosszát adja meg.

Ebből számítja ki az interpreter a kitartási (SUSTAIN) időt. A <tartam> megadására az F1-F8 billentyűk szolgálnak:

Billentyű	Jelentés
F1	tizenhatod
F3	nyolcad
F5	negyed
F7	fél
F8	egész
F2	dupla
F4	tripla
F8	négyszeres

Az n érték az egész hang (F8) hosszát határozza meg. Minél nagyobb az n, annál hosszabb lesz egy-egy ütem.

A sztring további speciális jeleket tartalmazhat.

<CLR>H

A fenti karakterek megtalálása után következő hangjegyeket a H hang fogja megszólaltatni (H = 1,2,3).

<CLR>G

A <zene-sztring> utolsó utasítása lehet; az utolsó hangot kikapcsolja.

PLAY n

A MUSIC utasítás segítségével tárolt zene lejátszására szolgál. Az n értékei a következők lehetnek:

0 = nem hallhatóvá teszi az éppen játszott zenét,
1 = lejátsza a MUSIC utasításban megadott zenét, majd folytatja a program futását,
2 = elkezdi játszani a MUSIC utasításban megadott zenét, **miközben** a program tovább fut. Lehetőség van pl. a grafikus utasítások hatását zenével kísérni.

Példák

(i)

```
10 VOL 15
20 WAVE 1,10000000
30 ENVELOPE 1,1,10,3,0
40 MUSIC 5,"<CLR>1C5<F2>"
45 REPEAT
50 PLAY 1
55 X=X+1:UNTIL X=4
```

(ii)

```
10 VOL 15
20 WAVE 1,00100000
30 ENVELOPE 1,8,8,8,2
40 A$="┗1Z▮C5▮E5▮C5▮E5▮G5═G5═"
50 B$="C6▮B5▮A5▮G5▮F5═A5═G5▮E5▮F5▮D5▮C5═┗G"
60 C$=A$+A$+B$
70 MUSIC 18,C$
90 PLAY 2
```

Az (i) példa kopogó, topogó hangot ad. A (ii) példa a Boci-boci tarkát játsza. Az egyes hullámformák és ADSR értékek hatását könnyen kipróbálhatjuk a 20. és 30. sorok cseréjével. Próbáljuk meg a következőt:

```
20 WAVE 1,10000000
30 ENVELOPE 1,2,0,12b
```

9. Gépi kódú programozás

9.1 A 6510 mikroprocesszor hardver lehetőségei

Ez a fejezet a C-64 gépi-kódú szinten történő programozásával foglalkozik. Az előző fejezetekben is adtunk gépi-kódú programrészeket, amelyek ennek a fejezetnek az ismerete nélkül nem érthetők. Ezzel szemben a 9. fejezet önmagában is használható annak, aki **csak** a 6510 mikroprocesszor működésére kíváncsi.

A memória használata

A C-64 alapjául szolgáló processzor a 8 bites mikroprocesszorok közé tartozik. Ez azt jelenti, hogy a processzor belső regiszterei 8 bitesek (kivéve az utasításszámlálót). A processzor egyidőben 64 Kbyte nagyságú operatív tárat tud kezelni. Ezek megcímzéséhez két byte-nyi információra van szükség. A címek (hexadecimális alakban) $0000-tól $FFFF-ig terjednek. A címeket szokás 256-osával csoportosítani,ezeket a memóriarészeket hívjuk lapoknak. Például a 0. lap a $0000-$00FF címeket tartalmazza, a $12. lap a $1200-$12FF címeket. A processzor számára a memória különböző részei egyenrangúak. Kivételt a 0. és 1. lap, valamint a 255.lap utolsó 6 byte-ja képez. Részletesen ezekről a továbbiakban szólunk.

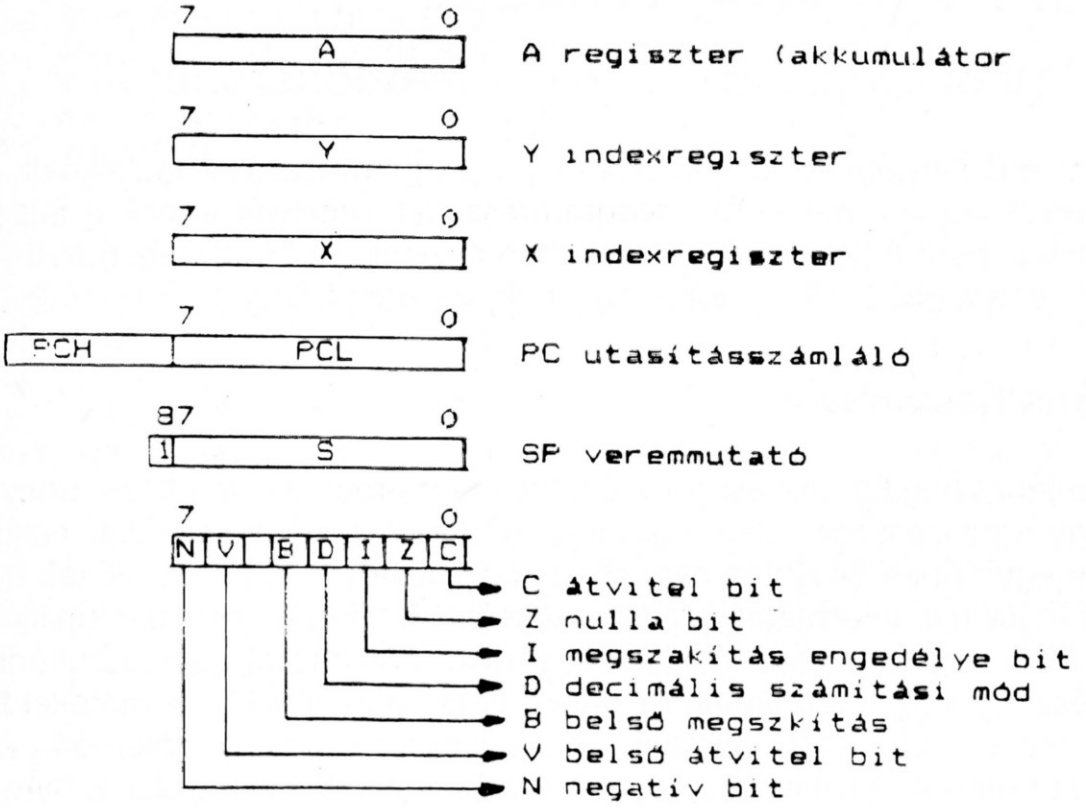
Mnemonikok

Az egyes utasításokat a hatásukra utaló betűkkel jelöljük, ez lényegesen könnyebben jegyezhető meg, mint hogy melyik byte jelenti azt az utasítást. Például az alábbi töltő utasításokat így kellene tárolnunk:

Mnemonik	Tárolás
LDA #$12	$A9 $12
LDA $12	$A5 $12
LDA $12,X	$B5 $12

Természetesen – további segédeszközök nélkül – a C-64 nem 'érti' meg ezeket a mnemonikokat. BASIC környezetben ezért a megfelelő byte-okat a POKE utasítással kell a memória megfelelő részében elhelyeznünk. A 9.5 paragrafusban ismertetjük azokat a lehetőségeket, amelyek lehetővé teszik például a mnemonikok használatát. A példáinkban természetesen mindig a mnemonikokat használjuk.

Programozási model



Mint a fenti ábrán is látható, a 6510 processzornak hét, egyenként 8 bites belső regisztere van, amelyek közül a PCH és PCL **egyetlen** 16 bites regiszter-párt alkot. Ezt szokásos **utasításszámlálónak** hívni. Normál körülmények között a processzor az utasításszámláló által meghatározott címről olvassa a következő utasítást, majd értelmezi (dekódolja), és végrehajtja.

Az A regisztert **akkumulátornak** hívjuk, a legtöbb műveletet az akkumulátorban levő 8 bites számmal tudja a processzor elvégezni. Az X és Y regisztereket **index-regiszteknek** is szokás hívni, mert a különféle címzési módoknál indexként használhatók.

Az SR regisztert **állapotregiszternek** hívjuk, mivel egyes bitjei az utoljára végrehajtott műveletről, eredményről adnak bizonyos információt. A 6510 utasításainak leírásánál ismertetjük, hogy azok milyen hatással vannak az egyes jelzőbitekre, itt csak általánosságban foglaljuk össze szerepüket.

Az **N jelzőbit** (az SR 7. bitje) az utasítás eredményének 7. bitjét tárolja és gyakorlatilag azzal mindig azonos. Az N elnevezés a 'negatív' szóból származik. Amikor kettes komplemens számokat használunk, a 7. bit az előjelet jelenti. Az N bit a BMI és BPL vezérlésátadó utasításokkal együtt használható.

A **V jelzőbit** (az SR 6. bitje) a 6510 legkevesebbet használt jelzőbitje, aminek oka a vezérlésátadó utasítások szerkezete; ritkán használható ez a jelzőbit. Részletes leírását a BVC utasítás alatt adjuk.

A **B jelzőbit** (az SR 3. bitje) magasra állítódik minden egyes BRK utasítás után. Szerepe, hogy a megszakítások és a BRK utasítások végrehajtása megkülönböztethető legyen. Másként ezt nem tehetjük meg, lévén, hogy ugyanazt a hardver vektort használják.

A **D jelzőbit** (az SR 4. bitje) magasra állítása a chip aritmetikai utasításainak hatását módosítja. Normál körülmények között (D = 0) például az összeadás 8 bites számok hexadecimális számokként való összeadását jelenti. A D jelzőbit magasra (D = 1) állítása lehetővé teszi a BCD (binary-coded decimal) számokkal való műveletvégzést. Ez azt jelenti, hogy a chip a 8 bit alsó, illetve felső 4-4 bitjét egy-egy decimális számjegynek tekinti és annak megfelelően használja azokat.

Az **I jelzőbit** (az SR 2. bitje) magasra állítása letiltja a maszkolható megszakításokat. Ha a jelzőbit alacsony, a processzor IRQ vonalán megjelenő minden lefutó él egy megszakítást generál.

A **Z jelzőbit** (az SR 1. bitje) magasra állítódik minden olyan esetben, amikor az eredmény valamennyi bitje 0. Minden más esetben a jelzőbit alacsony lesz.

A **C jelzőbit** (az SR 0. bitje) elsősorban az összeadásnál és a kivonásnál használjuk, ahol az átvitel szerepét játsza. A C bitet az eltolási műveleteknél is használjuk.

Verem

Mint említettük, a processzor az 1. lapot speciális módon használja. Több utasítás is van, amelyek erre a lapra írnak vagy erről a lapról olvasnak. Az 1. lapot a processzor veremként használja. A **verembe tölthetünk**, illetve a **veremből vehetünk ki** byte-okat. A verembe töltés azt jelenti, hogy az 1. lapnak az SP által definiált címére a processzor beírja a szóban forgó byte-ot, **majd** SP-t eggyel csökkenti. A veremből való kivevés azt jelenti, hogy az 1. lapnak az SP által definiált címéről kiolvassa az ott levő byte-ot, majd az SP értékét eggyel növeli. Ennek megfelelően SP-t **verem mutatónak** hívjuk. Értéke 00-FF lehet. Ennek megfelelően a verem címe $0100-$01FF. Az SP értéke közvetlenül is beállítható; a legtöbb veremművelet azonban a fent vázolt módon változtatja a verem mutató értékét.

Hardver vektorok

A 6510 mikroprocesszor, a legtöbb mikroprocesszorhoz hasonlóan, néhány speciális címet használ a címezhető memória végén. Az NMI, a RESET, az IRQ vonalak aktivizálásakor a programszámlálóba töltődnek az ($FFFA), ($FFFC) illetve ($FFFE) címek. Az IRQ vonal esetében ez csak akkor következik be, ha az I jelzőbit alacsony volt. A töltést megelőzően a visszatérési cím, illetve az SR a verembe kerül. A C-64 az IRQ vonalat a billentyűzet ellenőrzésére és a képernyő kezelésére (pl. a kurzor villogtatása) használja. Az NMI vonalat csak az RS 232-es csatornát kezelő rutinok használják.

I/O kapu

A $0001 cím egy kétirányú adatkapu, amelynek adatirány (DDR, data-direction) regisztere a $0000 címen található. A C-64 ezt a kaput arra használja, hogy lehetőséget biztosítson 64K-nál nagyobb memória felhasználására.

Címzési módok

A 6510 processzor 13 különböző címzési módot használ. Az egyes címzési módok eltérő eljárásokat jelentenek, amelyek segítségével a processzor a művelet operandusát előállítja. A 6510-es processzor utasításai maximum 3 byte hosszúak. Az első byte mindig meghatározza az utasítást és a címzési módot is. A további byte-ok a cím előállításához kellenek.

1 byte-os utasítások nem utalnak semmilyen címre vagy adatra, és a processzoron belül fejtik csak ki hatásukat. Ebben az esetben címzésről valójában nem is lehet beszélni. (Az angol terminológia az '**implied addressing**' elnevezést használja.) A jelzőbiteket állító,illetve bizonyos vermet kezelő utasítások tartoznak ebbe a csoportba.

2 byte-os utasítások az utasítást leíró byte-on kívül egyetlen byte-ból állnak. Ha ezt a byte-ot a processzor adatként használja, akkor **közvetlen címzésről** beszélünk. Az assembler-listákon ezeket az értékeket általában a # jel vezeti be, például LDA #$00.

A többi 2 byte-os utasítás a második byte-ot címként kezeli. összesen 6 különböző címzési mód van.

Az ugró utasítások esetén a második byte a vezérlésátadás helyének az éppen végrehajtott utasítás utáni byte-tól való távolságát adja és ezért ezt a címzést **relatív címzésnek** hívjuk.

Az összes többi címzési mód a 0. laphoz kapcsolódik és lehetővé teszik a cím felső, nulla byte-jának elhagyását. Így az utasítás rövidebb lesz és gyorsabban is hajtódik végre. Az egyes címzési módok a következők:

(i) 0. lapos címzés

A második byte a 0. lapon levő címet jelenti. Például LDA $55 a $0055 cím tartalmát tölti az akkumulátorba.

(ii) 0. lapos címzés az X indexregiszterrel

A cím az X indexregiszter tartalmának a második byte-hoz való hozzáadásával áll elő. Például LDA $55, X végrehajtásakor a cím, amelynek tartalma A-ba töltődik, $55 + X. Ha X tartalma éppen $12, akkor az $67.

(iii) **0. lapos címzés az Y indexregiszterrel**

Ez a címzés a fenti címzéshez hasonló azzal a különbséggel, hogy az X helyett az Y indexregiszter tartalma adódik a második byte-hoz.

(iv) **Indexelt indirekt címzés**

Ebben az esetben indexelésre csak az X regiszter használható. A 0. lapos X indexregiszterrel való címzés két egymást követő byte-ra mutat, amelyek együtt a valódi címet tartalmazzák.Tekintsük például a LDA $00,X utasítást. Legyen X tartalma $12 és ($12) = $33, ($13) = $AB A 0. lapos X indexregisztert használó címzés a $00 + X = $12 címet adja. A $12 címen található 2 byte-os mutató értéke $AB33 (forditott sorrendben kell tárolni!)
Az utasítás végül a $AB33 címen levő byte-ot tölti az akkumulátorba. A zárójelek az utasítás részei (ha assemblert használunk).

(v) **Indirekt indexelt címzés**

Ebben az esetben indexelésre csak az Y regiszter használható. A címzés hatását legegyszerűbben az LDA ($12), Y utasítás hatásának leírásával érzékeltethetjük. A $12 címen egy 2 byte-os mutató található (az előző példában ez $AB33 volt.) Ehhez a mutatóhoz adódik hozzá az Y regiszter értéke (ha például Y = $12 volt, akkor a valódi cím $AB45 lesz), és az így kapott címet használja a processzor.

3 byte-os utasítások az utasítást leíró byte-on kívül 2 byte-ot tartalmaznak, amelyek egy 16 bites címet alkotnak. Ez szolgál a címzés alapjául. Először kell az alsó, azután a felső byte-ot tárolni. Az egyes címzési módok részben a 0. lapos címzések megfelelői:

(i) **Abszolút címzés**

A 2 byte egy 16 bites címet határoz meg. Például LDA $1234 a $1234 cím tartalmát tölti az akkumulátorba.

(ii) **Abszolút címzés az X indexregiszterrel**

Az utasítás utolsó két byte-jából előállított cím értékéhez hozzá kell még adnunk az X regiszter értékét, hogy a címet megkapjuk. Például a LDA $31AB, X utasítás, ha X = $01, a $31AC cím értékét tölti az akkumulátorba.

(iii) **Abszolút címzés az Y indexregiszterrel**

Az előző címzési mód értelemszerű megfelelője.

(iv) **Abszolút indirekt címzés**

Ezt a címzési módot egyedül a JMP utasítással használhatjuk. JMP ($FF03) hatására az $FF03-on levő mutató az utasításszámlálóba töltődik, s onnan folytatódik a program futása.
Az egyes utasítások után mindig feltüntetjük, milyen címzési módokkal használhatók.

9.2 A 6510 processzor utasításkészlete

Ebben a paragrafusban – a mnemonikok ABC sorrendjében – felsoroljuk a 6510 utasításkészletét. A 6510-es processzor szoftver kompatibilis a 6502-es processzorral. Az utasítások végrehajtásánál megadjuk a ciklusok számát is. Az utasítások leírásában * azt jelenti, hogy amennyiben a címzés módosítása laphatárt lép át, további ciklusra van szükség. A + azt jelenti, hogy az ugrás végrehajtása esetén még egy plusz ciklus kell.

ADC

Az akkumulátorhoz hozzáadja az átvitel bitet és az operandust; A,C: = A + M + C

Utasítás		Címzés	Hossz	Ciklusok száma
$61	97	ADC (0. lap, X)	2	6
$65	101	ADC 0. lap	2	3
$69	105	ADC # érték	2	2
$6D	109	ADC abszolút	3	4
$71	113	ADC (0. lap), Y	2	5 *
$75	117	ADC 0. lap, X	2	4
$79	121	ADC abszolút, Y	3	4 *
$7D	125	ADC abszolút, X	3	4 *

Állapotregiszter: NV-BDIZC
xx xx

Végrehajtás: Az akkumulátor jelenlegi tartalmához hozzáadja a címzési mód által meghatározott byte-ot, továbbá az átvitel bitet. Amennyiben az eredmény nem tárolható egyetlen byte-on, a C jelzőbit 1 lesz. A belső túlcsordulást jelző V bit magas lesz, ha a 6. bitről van átvitel a 7. bitre.

Ha a végeredmény nulla (az akkumulátor minden bitje 0), a Z bit magas lesz, különben 0. Ha az A regiszter 7. bitje 1, az N bit ugyancsak 1 lesz, jelezve a 'negatív' eredményt.

Példák: Az átvitel bit megfelelő beállítása lehetővé teszi több byte-os számok összeadását. Az akkumulátor tartalmának eggyel való növelése, illetve csökkentése:

```
CLC
ADC #$01      ; A növelése
CLC
ADC #$FF      ; A csökkentése
```

Decimális aritmetika esetén (amikor D = 1) egy-egy byte alsó és felső négy bitje 0-9 közti számjegyeket tartalmaz, és a chip automatikusan ennek megfelelően végzi el az összeadást. Az alábbi példa 321-et (decimálisan) ad a CIM, CIM + 1 helyeken tárolt decimális számhoz. Így 828, 829 a 0-9999 intervallumba eső BCD számot tartalmaz.

```
10 SED ;(CIM)=(CIM)+123
20 CLC
30 LDA CIM+1
40 ADC #$23
50 STA CIM+1
60 LDA CIM
70 ADC #$01
80 STA CIM
90 CLD
100 RTS
110 .END
```

Decimális aritmetika esetén a Z bit nem ad helyes eredményt. A nulla végeredményt a TAX/BEQ vagy a CMP #$00 utasításokkal lehet lekérdezni.

AND

Az akkumulátor és a memória tartalmának 'logikai és'-e; A: = A AND M.

Utasítás		Címzés	Hossz	Ciklusok száma
$21	33	AND (0. lap, X)	2	6
$25	37	AND 0. lap	2	3
$29	41	AND # érték	2	2
$2D	45	AND abszolút	3	4
$31	49	AND (0. lap), Y	2	5
$35	53	AND 0. lap, X	2	4
$39	57	AND abszolút, Y	3	4 *
$3D	61	AND abszolút, X	3	4 *

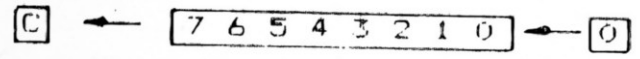
Állapotregiszter: NV-BDIZC
x x

Végrehajtás: Az akkumulátor és a címzési mód által meghatározott byte bitjeire végrehajtja az AND műveletet. Az eredmény az akkumulátorba kerül. Ha A valamennyi bitje 0, akkor a Z bit magas lesz, különben alacsony marad. Az N bit magas lesz, ha az akkumulátor 7. bitje magas.

Példa: AND #$FF a jelzőbiteket ugyanúgy állítja be, mintha az LDA utasítást hajtottuk volna végre. AND #$7F mindig alacsonyra állítja a 7. bitet.

ASL

Az akkumulátor vagy a memória tartalmát egy bittel balra tolja:



Utasítás		Címzés	Hossz	Ciklusok száma
$06	6	ASL 0. lap	2	5
$0A	10	ASL A	1	2
$0E	14	ASL abszolút	3	6
$16	22	ASL 0. lap, X	2	6
$1E	30	ASL abszolút, X	3	7

Állapotregiszter: NV-BDIZC
x xx

Végrehajtás: Az utasítás a memória vagy akkumulátor tartalmát egy bittel balra tolja. A 0. bit helyére nulla kerül, a 7. bit pedig a C bitbe másolódik. Az eredmény az akkumulátorba kerül. A Z és N bitek az eredménynek megfelelően állítódnak.

Példák: Az utasítás segítségével szorozhatunk egy számot. Például LDA $20 / ASL A / ADC $20 a $20 tartalmát 3-mal szorozza meg, feltéve, hogy ennek értéke legfeljebb 127. Ekkor az ASL A után a C bit automatikusan 0 lesz. Négy egymást követő ASL az alsó négy bitet a felső négy bitbe tolja át.

BCC

Vezérlésátadás, ha C = 0; (PC: = PC + < módosítás > ; feltéve, hogy C = 0)

Utasítás		Címzés	Hossz	Ciklusok száma
$90	144	BCC < módosítás >	2	2 * +

Állapotregiszter: nem változik.

Végrehajtás: Ha a C = 0 feltétel teljesül, az utasítás második byte-ja hozzáadódik az utasításszámláló tartalmához, amelyik már a következő utasítás elejére mutat. Ha C = 1, az utasításszámláló nem változik.

Példák: Első példánk egy 2 byte-os összeadást mutat,ahol túlcsordulás esetén egy másik ágon (OVERFL) folytatódik a program:

```
10 CLC      ;(CIM)=(CIM)+(LO,HI)
20 LDA CIM
30 ADC #LO
40 STA CIM
50 LDA CIM+1
60 ADC #HI
70 STA CIM+1
80 BCC CONT
90 JMP OVERFL
100 .END
```

A C bit állításával egyszerű programlevágásokat hozhatunk létre. Erre példa a következő programrész:

```
       BCC TOVABB
       JMP CIM1
TOVABB JMP CIM2
```

BCS

Vezérlésátadás, ha C = 1; (PC: = PC + < módosítás >; feltéve, hogy C = 1)

Utasítás	Címzési mód	Hossz	Ciklusok száma
$B0 176	BCS < módosítás >	2	2 * +

Állapotregiszter: nem változik.

Végrehajtás: Megegyezik a BCC utasítással, azzal a kivétellel, hogy a vezérlésátadásra a C = 1 esetben kerül sor.

BEQ

Vezérlésátadás, ha Z = 1; (PC = PC + < módosítás >; feltéve, hogy Z = 1)

Utasítás	Címzési mód	Hossz	Ciklusok száma
$F0 240	BEQ < módosítás >	2	2 â +

Állapotregiszter: nem változik.

Végrehajtás: Ha Z = 1, az utasítás második byte-ja (kettes komplemens alakban tárolt számként) hozzáadódik az utasításszámlálóhoz, amelyik közvetlenül az utasítás utánra mutat. Ennek hatására előre vagy hátra történik az ugrás. Ha Z = 0, a vezérlésátadás nem történik meg.

Példák: A Z jelzőbit nem állítható közvetlenül, de a legtöbb művelet eredményeként állítódhat, ezért a BEQ használata feltételes vezérlésátadás esetén elég gyakori. Általában összehasonlítás után használjuk. A

```
LDA INPUT / CMP #$2C / BEQ VESSZO
```

utasítássorozat esetén a vezérlésátadás akkor történik meg, ha INPUT egy vessző ASCII kódját tartalmazza

A BEQ utasítást gyakran használjuk ciklusszervezéssel is:

```
     LDX  #$FF
LOOP TXA
     STA  $0400, X
     DEX
     BEQ EXIT
     ...   ciklusmag
     JMP LOOP
EXIT ...   folytatás
```

BIT

A memória bizonyos bitjeit ellenőrzi. Z = 1 lesz, ha A AND M nulla, N = M7; V = M6. M a címzési módnak megfelelő operandus tartalmát jelenti.

Utasítás	Címzési mód	Hossz	Ciklusok száma
$24 36	BIT 0. lap	2	3
$2C 44	BIT abszolút	3	4

Állapotregiszter:

```
NV-BDIZC
M7 |     x
   M6
```

Végrehajtás: A BIT utasítás mindössze három jelzőbit értékét változtatja meg, a regiszterek és a memória nem változik. Ha (A AND M) nem tartalmaz egyetlen magas bitet sem, akkor Z magas lesz. A 6. és 7. bit értékét a V, illetve az N bitbe másolja a processzor.

Példák: A BIT utasítást a BMI/BPL vagy a BVC/BVS vezérlésátadó utasításokkal együtt gyakran használjuk a CIA chip vezérlését végző rutinokban, lévén hogy a PB7 és PB6 biteknek speciális szerepe van.

BMI

Vezérlésátadás, ha N = 1 (PC: = PC + < módosítás >; feltéve, hogy N = 1)

Utasítás	Címzési mód	Hossz	Ciklusok száma
$30 48	BMI < módosítás >	2	2 * +

Állapotregiszter: nem változik.

Végrehajtás: Ha az N bit értéke 1, akkor a második byte (kettes komplemens számként) hozzáadódik az utasításszámláló tartalmához. Ennek megfelelően előre vagy hátra egy ugrás történik. Ha N = 1, a vezérlésátadás nem történik meg.

Példák: Az utasítást a memória valamely byte-jának tesztelésére használhatjuk. Például a következő programrész addig vár, míg az RS 232-es csatorna DSR vonala alacsony nem lesz:

```
LOOP BIT $D801
     BMI LOOP
```

BNE

Vezérlésátadás, ha Z = 0. (PC: = PC + < módosítás >; feltéve, hogy Z = 0)

Utasítás	Címzési mód	Hossz	Ciklusok száma
$D0 208	BNE < módosítás >	2	2 * +

Állapotregiszterek: nem változnak.

Végrehajtás: A BNE pontosan úgy működik, mint a BEQ, csak az ugrás a Z = 0 esetben történik meg. Ha Z = 1, a vezérlésátadásra nem kerül sor.

Példák: A BNE utasítást lehet talán a legegyszerűbben ciklusok szervezésére felhasználni; például

```
     LDX #$00
LOOP TXA
     STA $03FF,X
     DEX
     BNE LOOP
```

a képernyő tetejére nyomtat 255 karaktert. A BNE-t, hasonlóan a BEQ-hoz, gyakran használjuk a CMP utasítás után.

BPL

Vezérlésátadás, ha N = 0 (PC: = PC + < módosítás >; feltéve, hogy N = 0)

Utasítás	Címzési mód	Hossz	Ciklusok száma
$10 16	BPL < módosítás >	2	2 * +

Állapotregiszter: nem változik.

Végrehajtás: Hasonlóan működik, mint a BMI, azzal a kivétellel, hogy a vezérlésátadásra abban az esetben kerül sor, ha N = 0, azaz az eredmény pozitív vagy nulla volt.

BRK

Belső megszakítás. S: = PCH, SP: = PCL, S: = SR, SP: = SP-1, PC: = ($FFFE).

Utasítás	Címzési mód	Hossz	Ciklusok száma
$00	BRK	1	7

Állapotregiszter: NV-BDIZC
1 X

Végrehajtás: A BRK utasítás eredménye hasonló, mint egy megszakításé; az utasításszámlálót és az állapotregisztert a processzor a verembe tölti, és az ($FFFE) címre ugrik. Az elmentett utasításszámláló 2-vel nagyobb a BRK utasítást tartalmazó byte címénél. Eltérően a megszakítástól, az állapotregiszter B bitje magasra állítódik.

BVC

Vezérlésátadás, ha V = 0. (PC: = PC + < módosítás >; feltéve, hogy V = 0)

Utasítás	Címzési mód	Hossz	Ciklusok száma
$50 80	BVC < módosítás >	2	2 * +

Állapotregiszter: nem változik.

Végrehajtás: Ha V alacsony, az utasításszámláló tartalmához (ami a 2. byte után mutat) kettes komplemens számként hozzáadódik az utasítás második byte-ja, és az ennek megfelelő utasítással folytatódik a program végrehajtása.

Ha kettes komplemens számokat használunk, két **ellenkező** előjelű szám összeadása

sohasem okoz túlcsordulást. Azt okozhat azonban azonos előjelű számok összeadása. Ha a 100 + 89 összeget akarjuk kiszámítani, az 189-et ad eredményül, ami túlcsordulást és egy negatív számot eredményezhet. Összeadás esetén, ha az összeadandók és az összeg előjel bitjei N1, N2, illetve N, akkor

$V = (\overline{N1 \text{ EOR } N2})$ AND (N EOR N1)

BVS

Vezérlésátadás, ha V = 1. (PC: = PC + < módosítás >; feltéve, hogy V = 1).

Utasítás	Címzési mód	Hossz	Ciklusok száma
$70 112	BVS < módosítás >	2	2 * +

Állapotregiszter: nem változik.

Végrehajtás: Megegyezik a BVC végrehajtásával, azzal a különbséggel, hogy az ugrás a V = 1 esetben történik meg.

CLC

Törli az átvitel bitet, C: = 0.

Utasítás	Címzési mód	Hossz	Ciklusok száma
$18 24	CLC	1	2

Állapotregiszter: NV-BDIZC
0

Végrehajtás: a C bitet 0-ra állítja. Egyéb változás nem történik.

Példák: Az átvitel bit felhasználása automatikus az összeadásnál és a kivonásnál (ADC,SBC). Használhatjuk a műveletektől független jelzőbitként, hiszen közvetlenül alacsonyra, illetve magasra állítható.

CLD

Törli a decimális számítási mód jelzőbitjét; D: = 0

Utasítás	Címzési mód	Hossz	Ciklusok száma
$D8 216	CLD	1	2

Állapotregiszter: NV-BDIZC
0

Végrehajtás: A D bitet 0-ra állítja. Egyéb változás nem történik.

Példák: Valahányszor hexadecimális (bináris) aritmetikát használunk, előtte a D jelzőbitet törölni kell. A C-64 BASIC interpreter **nem** használja a decimális számítási módot, így a D bit – ha nem állítottuk magunk magasra – mindig alacsony.

CLI

A maszkolható megszakításokat letiltó bitet törli; I: = 0

Utasítás	Címzési mód	Hossz	Ciklusok száma
$58 88	CLI	1	2

Állapotregiszter: NV-BDIZC
0

Végrehajtás: Az I bit alacsony lesz. Ettől kezdődően az IRQ vonalon megjelenő valamennyi megszakításkérést fogadja a rendszer. Ez azt jelenti, hogy az utasításszámláló és az állapotregiszter elmentése után a ($FFFE) címmel folytatódik a program. A CLI utasításnak az I bit alacsonyra állításán kívül más hatása nincs.

CLV

Törli a belső túlcsordulást jelző bitet, V: = 0.

Utasítás	Címzési mód	Hossz	Ciklusok száma
$B8 184	CLV	1	2

Állapotregiszter: NV-BDIZC
0

Végrehajtás: A V bit alacsony lesz. Egyéb változás nem történik.

CMP

Összehasonlítja az akkumulátor és az operandus tartalmát. SR az A-M értékének megfelelően állítódik be.

Utasítás		Címzési mód	Hossz	Ciklusok száma
$C1	193	CMP (0. lap, X)	2	6
$C5	197	CMP 0. lap	2	3
$C9	201	CMP # érték	2	2
$CD	205	CMP abszolút	3	4
$D1	209	CMP (0. lap), Y	2	5 *
$D5	213	CMP 0. lap, X	2	4
$D9	217	CMP abszolút, Y	3	4 *
$DD	221	CMP abszolút, X	3	4 *

Állapotregiszter: NV-BDIZC
X XX

Végrehajtás: A CMP utasítás az N, Z és C biteken kívül nem változtatja meg sem a regiszterek sem a memória tartalmát. Az utasítás az akkumulátorból kivonja a címzési módnak megfelelő byte-ot és N,Z,C értéke ennek megfelelően íródik be. Ha az eredmény 7. bitje magas, N is az lesz; ha 0, akkor Z magas lesz; végül ha keletkezik átvitel, C is magas lesz. A különbséget nem tárolja a processzor.

Példák: A CMP utasítást leggyakrabban a Z bit lekérdezésével használjuk. Például LDA $033C / CMP #$20 / BEQ CIM hatására a vezérlés akkor kerül a CIM-re, ha a $033C címen $20 volt. Különben a program folytatódik.

Ha a byte értéke, amit kivontunk az akkumulátorból, nagyobb mint A, akkor a C magas lesz. Ezt is fel lehet használni ellenőrzésre. Például a következő programrész ellenőrzi, hogy az akkumulátorban levő szám milyen intervallumba esik:

```
LDA MUTATO,Y
CMP #$20
BCC M1
CMP #$40
BCC M2
CMP #$60
BCC M3
```

CPX

Összehasonlítja az X regiszter és az operandus tartalmát. SR az X-M értékének megfelelően állítódik be.

Utasítás		Címzési mód	Hossz	Ciklusok száma
$E0	224	CPX #érték	2	2
$E4	228	CPX 0. lap	2	3
$EC	236	CPX abszolut	3	4

Állapotregiszter: NV-BDIZC
X XX

Végrehajtás: A CPX csak a fent jelzett három jelzőbit értékét állítja. Az utasítás címzési módjában meghatározott byte-ot a processzor kivonja az X regiszterből, és az N,Z,C bitek a művelet eredményének megfelelően állítódnak be.

CPY

Összehasonlítja az Y regiszter és az operandus tartalmát. SR az Y-M értékének megfelelően állítódik be.

Utasítás		Címzési mód	Hossz	Ciklusok száma
$C0	192	CPY #érték	2	2
$C4	196	CPY 0. lap	2	3
$CC	204	CPY abszolut	3	4

Állapotregiszter: NV-BDIZC
X XX

Végrehajtás: A CPY csak a fent jelzett három jelzőbit értékét állítja, hasonlóan a CPX utasításhoz.

DEC

Eggyel csökkenti az operandus tartalmát; M: = M-1.

Utasítás		Címzési mód	Hossz	Ciklusok száma
$C6	198	DEC 0. lap	2	5
$CE	206	DEC abszolut	3	6
$D6	214	DEC 0. lap, X	2	6
$DE	222	DEC abszolut, X	3	7

Állapotregiszter: NV-BDIZC
X X

Végrehajtás: A címzési módnak megfelelő byte értéke eggyel csökken, és az eredménynek megfelelően beállítódnak az N és Z bitek.

Példák: Az alábbi rövid rutin azt mutatja, hogyan lehet a 0. lapon levő két byte-os mutatót ($2D-2E) eggyel csökkenteni:

LDA $2D / BNE +2 / DEC $2E / DEC $2D

A DEC utasítás segítségével a RAM-ban tárolt számlálókat is használhatunk (nem csak az X és Y regisztert).

DEX

Az X regiszter tartalmát eggyel csökkenti; X: = X-1.

Utasítás	Címzési mód	Hossz	Ciklusok száma
$CA 202	DEX	1	2

Állapotregiszter: NV-BDIZC
X X

Végrehajtás A processzor eggyel csökkenti az X regiszter tartalmát. Az N bit magas lesz, ha az eredmény 7. bitje 1. A Z jelzőbit magas lesz, ha X valamennyi bitje 0. (Vagyis a csökkentés előtt az értéke 1 volt.)

Példák: A következő programrész 20 karaktert másol át a memóriába:

LDX #$14 / LOOP LDA MUT1,X / STA MUT2,X / DEX / BNE LOOP

Következő példánk törli a képernyőt, valamennyi karakterhelyre szóközt (= $20) írva:

```
033C             5       *=828
033C A2 00      10       LDX #$00
033E A9 20      20 LP    LDA #$20
0340 9D 00 04   30       STA $0400,X
0343 9D 00 05   40       STA $0500,X
0346 9D 00 06   50       STA $0600,X
0349 9D 00 07   60       STA $0700,X
034C CA         70       DEX
034D D0 EF      80       BNE LP
034F 60         90       RTS
0350           100       .END
```

DEY

Az Y regiszter tartalmát eggyel csökkenti; Y: = Y-1.

Utasítás	Címzési mód	Hossz	Ciklusok száma
$88 136	DEY	1	2

Állapotregiszter: NV-BDIZC
X X

Végrehajtás: Eggyel csökkenti az Y regiszter tartalmát, és az eredménytől függően beállítja az N és Z biteket.

Példák: A DEY utasítást, hasonlóan a DEX utasításhoz, elsősorban ciklusok szervezésére használjuk, de annál ritkábban. (Az Y regiszterrel ugyanis kevesebb utasítást lehet indexelni.) Példánk három egymást követő byte-ról dönti el, hogy mindhármuk nulla-e. A használt indexelés csak az Y regiszterrel lehetséges:

```
     LDY #$02
     LDA #$00
IDE  ORA (PTR),Y
     DEY
     BNE IDE
```

EOR

Az akkumulátor és az operandus közt bitenként végrehajtja a 'kizáró-vagy' logikai műveletet; A: = A EOR M

Utasítás		Címzési mód	Hossz	Ciklusok száma
$41	65	EOR (0. lap, X)	2	6
$45	69	EOR 0. lap	2	3
$49	73	EOR #érték	2	2
$4D	77	EOR abszolút	3	4
$51	81	EOR (0. lap), Y	2	5 *
$55	85	EOR 0. lap, X	2	4
$59	89	EOR abszolút, Y	3	4 *
$5D	93	EOR abszolút, X	3	4 *

Állapotregiszter: NV-BDIZC
X X

Végrehajtás: Az eredmény valamely bitje abban az esetben lesz csak magas, ha az akkumulátorban, illetve a címzési módnak megfelelő byte-ban ugyanazon a helyen levő két bit különböző. Ha például A = $52 és M = $AA, akkor (lévén A = %01010010, M = %10101010) A EOR M = %11111000 = $F8. Az N és Z bitek értelemszerűen állítódnak.

Példák: A C-64 VIC II chipje az inverz karaktereket a megfelelő byte 7. bitjének magasra állításával jelzi. A 7. fejezetben az EOR $80 utasítást a képernyő villogtatására használtuk. A billentyűzet ellenőrzésénél az EOR #$40 a siftelés jelzésére használható.

INC

Eggyel növeli az operandus tartalmát; M: = M + 1.

Utasítás		Címzési mód	Hossz	Ciklusok száma
$E6	230	INC 0. lap	2	5
$EE	238	INC abszolút	3	6
$F6	246	INC 0. lap, X	2	6
$FE	254	INC abszolút, X	3	7

Állapotregiszter: NV-BDIZC
X X

Végrehajtás: A címzési módnak megfelelő byte értéke eggyel nő, és az N és Z bitek értelemszerűen beállítódnak.

Példák: Az INC utasítást általában akkor használjuk, ha kettőnél (X,Y regiszterek) több számlálóra van szükség. Ilyen példa található a 7. fejezetben.

INX

Eggyel növeli az X regiszter tartalmát; X: = X + 1.

Utasítás	Címzési mód	Hossz	Ciklusok száma
$E8 232	INX	1	2

Állapotregiszter: NV-BDIZC
X X

Végrehajtás: Értelemszerű.

INY

Eggyel növeli az Y regiszter tartalmát; Y: = Y + 1.

Utasítás	Címzési mód	Hossz	Ciklusok száma
$C8 200	INY	1	2

Állapotregiszter: NV-BDIZC
X X

Végrehajtás: Értelemszerű.

JMP

A memória meghatározott helyén folytatja a program végrehajtását; PCL: = (PC + 1), PCH: = (PC + 2).

Utasítás	Címzési mód	Hossz	Ciklusok száma
$4C 76	JMP abszolút	3	3
$6C 108	JMP (abszolút)	3	3

Állapotregiszter: nem változik.

Végrehajtás: A JMP utasítás a GOTO utasítás gépi kódú megfelelője. Az abszolút címzés használata esetén a JMP kódját ($4C) követő első byte lesz az utasításszámláló alsó byte-ja, a második byte pedig az utasításszámláló felső byte-ja. A vezérlésátadás ennek megfelelően hajtódik végre.
Az abszolút indirekt címzés esetén a JMP kódját ($6C) követő két byte a memória egy címét határozza meg, ahonnan a processzor két byte-ot az utasításszámlálóba tölt.

Példák: Az 5-8. fejezet gépi kódú példái számos JMP utasítást tartalmaznak.

JSR

A memória meghatározott helyén folytatja a program végrehajtását, előbb azonban elmenti a visszatérési címet a verembe;(PC + 2),(PC + 2)L,SP: = SP-2,PCL = (PC + 1): PCH = (PC + 2).

Utasítás	Címzési mód	Hossz	Ciklusok száma
$20 32	JSR abszolút	3	6

Állapotregiszter: nem változik

Végrehajtás: A JSR utasítás a GOSUB gépi kódú megfelelője. Hasonlóan a BRK utasításhoz, a JSR is PC + 2 értékét tölti a verembe, először a felső, azután az alsó byte-ot. Ez a cím a JSR utasítás utolsó byte-jára mutat. Így az RTS utasításnak majd eggyel növelnie kell az utasításszámláló tartalmát.

Példák: Lásd az 5-8. fejezetek gépi kódú példáit!

LDA

Az operandusnak megfelelő byte-ot az akkumulátorba tölti; A: = M.

Utasítás		Címzési mód	Hossz	Ciklusok száma
$A1	161	LDA (0. lap, X)	2	6
$A5	165	LDA 0. lap	2	3
$A9	169	LDA #érték	2	2
$AD	173	LDA abszolút	3	4
$B1	177	LDA (0. lap), Y	2	5 *
$B5	181	LDA 0. lap, X	2	4
$B9	185	LDA abszolút, Y	3	4 *
$BD	189	LDA abszolút, X	3	4 *

Állapotregiszter: NV-BDIZC
X X

Végrehajtás: A címzési módnak megfelelő byte-ot az akkumulátorba tölti. N magas lesz, ha az akkumulátor 7. bitje az. A Z bit pontosan akkor lesz magas, ha nulla byte-ot töltöttünk az akkumulátorba.

Példák: A memória különböző részei közti adatátvitelhez a LDA/STA utasításpár többszöri használata szükséges. Az alábbi programrész 256 byte-ot másol át:

```
LDX #$00 / LOOP LDA $6000,X / STA $8000,X / DEX / BNE LOOP
```

A kétváltozós műveletek kivétel nélkül az akkumulátort használják, ezért először az első operandust az akkumulátorba kell tölteni.

LDX

A memória tartalmát az X regiszterbe tölti; X: = M.

Utasítás		Címzési mód	Hossz	Ciklusok száma
$A2	162	LDX #érték	2	2
$A6	166	LDX 0. lap	2	3
$AE	174	LDX abszolút	3	4
$B6	182	LDX 0. lap, Y	2	4
$BE	190	LDX abszolút, Y	3	4 *

Állapotregiszter: NV-BDIZC
X X

Végrehajtás: A címzési módnak megfelelő byte-ot az X regiszterbe tölti. Az N és Z bitek értelemszerűen állítódnak.

Példák: Az X változót gyakran használjuk segédváltozók tárolására. Az X regiszternek két olyan tulajdonsága is van, ami megkülönbözteti az A regisztertől. Egyrészt indexként használhatjuk (egyik leggyakoribb eset), másrészt a verem mutatóba ennek segítségével írhatunk. Így a LDX #$00/.../DEX/ BNE... illetve LDX #00/TXS utasítássorok igen gyakoriak.

LDY

Az operandus tartalmát az Y regiszterbe tölti; Y: = M

Utasítás		Címzési mód	Hossz	Ciklusok száma
$A0	160	LDY #érték	2	2
$A4	164	LDY 0. lap	2	3
$AC	172	LDY abszolút	3	4
$B4	180	LDY 0. lap, X	2	4
$BC	188	LDY abszolút, X	3	4 *

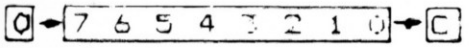
Állapotregiszter: NV-BDIZC
X X

Végrehajtás: A címzési mód által meghatározott byte-ot az Y regiszterbe tölti. Az N és Z bitek értelemszerűen állítódnak.

Példák: Az Y regisztert is elsősorban ciklusok szervezésére használjuk. Az Y-t azonban kevesebb címzési módban használhatjuk, mint az X regisztert, ezért az utóbbit általában előnyben részesítjük.

LSR

Az akkumulátor vagy memória tartalmát egy bittel jobbra tolja:



Utasítás		Címzési mód	Hossz	Ciklusok száma
$46	70	LSR 0. lap	2	5
$4A	74	LSR A	1	2
$4E	78	LSR abszolút	3	6
$56	86	LSR 0. lap, X	2	6
$5E	94	LSR abszolút, X	3	7

Állapotregiszter: NV-BDIZC
0 XX

Végrehajtás: A címzési módban meghatározott byte bitjeit egy hellyel jobbra tolja. A 0. bit tartalma átmásolódik a C jelzőbitbe, és a 7. bit helyére mindig 0 kerül. Ennek megfelelően N mindig alacsony lesz; Z értelemszerűen állítódik.

Példák: LSR/BCC ellenőrzi a 0. bitet és ugrik, ha az nem volt 1. LSR/LSR/LSR/LSR a felső négy bitet az alsó négy bitbe másolja át.

NOP

Üres utasítás.

Utasítás	Címzési mód	Hossz	Ciklusok száma
$EA 234	NOP	1	2

Állapotregiszter: nem változik.

Végrehajtás: Semmilyen hatása sincs, általában időzítéseknél, vagy helypótlónak használjuk.

ORA

Az akkumulátor és az operandus minden egyes bitjére végrehajtja a 'vagy' logikai műveletet; A: = A OR M.

Utasítás		Címzési mód	Hossz	Ciklusok száma
$01	1	ORA (0. lap, X)	2	6
$05	5	ORA 0. lap	2	3
$09	9	ORA #érték	2	2
$0D	13	ORA abszolút	3	4
$11	17	ORA (0. lap), Y	2	5
$15	21	ORA 0. lap, X	2	4
$19	25	ORA abszolút, Y	3	4 *
$1D	29	ORA abszolút, X	3	4 *

Állapotregiszter: NV-BDIZC
X X

Végrehajtás: Ha az akkumulátor vagy a címzési módnak megfelelő byte adott helyen levő bitjei közül valamelyik magas, az eredmény azon a helyen levő bitje is magas lesz. Az N és Z bitek értelemszerűen állítódnak.

PHA

Az akkumulátor tartalma a verembe töltődik; S: = A , SP: = SP-1.

Utasítás		Címzési mód	Hossz	Ciklusok száma
$48	72	PHA	1	3

Állapotregiszter: nem változik.

Végrehajtás: Az akkumulátor tartalmát a processzor a verem-mutatónak megfelelő helyre tölti, és a verem mutatót eggyel csökkenti.

Példák: Részeredmények tárolására a verem az egyik legalkalmasabb hely. Arra kell csak ügyelni, hogy például RTS előtt a részeredményeket távolítsuk el belőle, különben nem a megfelelő helyre tér vissza a vezérlés. A BASIC interpreter a GOSUB és a FOR utasítások végrehajtásához szükséges információt tárolja a veremben.

A következő programrész elmenti a verembe az A, X és Y regiszterek tartalmát:

PHA /TXA/ PHA/TYA/PHA.

PHP

Az állapotregiszter tartalmát a verembe tölti; S: = SR, SP: = SP-1.

Utasítás	Címzési mód	Hossz	Ciklusok száma
$08 8	PHP	1	3

Állapotregiszter: nem változik.

Végrehajtás: Az állapotregiszter tartalmát a verem mutatónak megfelelő helyre tölti; majd a verem mutatót eggyel csökkenti.

Példák: Az állapotregiszter a PHP/PLA utasításpárral tölthető az akkumulátorba. A BRK utasítás jelzőbitje csak ezen az úton kérdezhető le.

PLA

A verem tartalmát az akkumulátorba tölti; A: = S , SP: = SP + 1

Utasítás	Címzési mód	Hossz	Ciklusok száma
$68 104	PLA	1	4

Állapotregiszter: NV-BDIZC
X X

Végrehajtás: A processzor megnöveli eggyel a verem mutató értékét, és a megnövelt mutatónak megfelelő helyről egyetlen byte-ot tölt az akkumulátorba. A PLA utasítás a PHA utasítás 'inverze'. A jelzőbitek az LDA utasításhoz hasonlóan állítódnak be.

PLP

A verem tartalmát az állapotregiszterbe tölti; SP: = SP + 1, SR: = S.

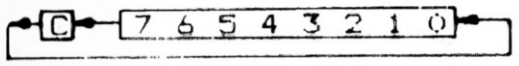
Utasítás	Címzési mód	Hossz	Ciklusok száma
$28 40	PLP	1	4

Állapotregiszter: NV-BDIZC
XX XXXXX

Végrehajtás: Hasonló a PLA végrehajtásához, azzal a különbséggel, hogy a verem tartalma az állapotregiszterbe kerül, s nem az akkumulátorba. Ennek megfelelően valamennyi jelzőbit értéke megváltozhat.

ROL

Ciklikusan eggyel balra tolja a memória vagy az akkumulátor és a C jelzőbit tartalmát



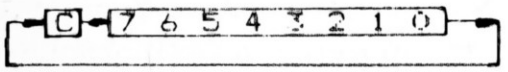
Utasítás		Címzési mód	Hossz	Ciklusok száma
$26	38	ROL 0. lap	2	5
$2A	42	ROL A	1	2
$2E	46	ROL abszolút	3	6
$36	54	ROL 0. lap, X	2	6
$3E	62	ROL abszolút, X	3	7

Állapotregiszter: NV-BDIZC
X XX

Végrehajtás: Az utasítás kilenc bitet ciklikusan balra tol egy bittel. Az egyik bit a C jelzőbit, a többi nyolc az utasítás címzési módjának megfelelő byte nyolc bitje. A bitek a fenti ábrának megfelelően tolódnak el. Az N, Z és C jelzőbitek értelemszerűen állítódnak.

ROR

Ciklikusan eggyel jobbra tolja a memória vagy az akkumulátor és a C jelzőbit tartalmát.



Utasítás		Címzési mód	Hossz	Ciklusok száma
$66	102	ROR 0. lap	2	5
$6A	106	ROR A	1	2
$6E	110	ROR abszolút	3	6
$76	118	ROR 0. lap, X	2	6
$7E	126	ROR abszolút, X	3	7

Állapotregiszter: NV-BDIZC
X XX

Végrehajtás: Az utasítás kilenc bitet egy bittel ciklikusan jobbra tol. Az egyik bit a C jelzőbit, a többi nyolc az utasítás címzési módjának megfelelő byte nyolc bitje. A bitek a fenti ábrának megfelelően tolódnak el. Az N, Z, C jelzőbitek értelemszerűen állítódnak.

RTI

Visszatérés a megszakítást kezelő rutinból; SR: = S, PCL: = S, PCH: = S, SP: = SP + 3.

Utasítás	Címzési mód	Hossz	Ciklusok száma
$40 64	RTI	1	6

Állapotregiszter: NV-BDIZC
XX XXXXX

Végrehajtás: Az utasítás három byte-ot tölt vissza a veremből; először az állapotregisztert, majd az utasításszámláló alsó és felső byte-ját. Ha a megszakítás óta ugyanannyi PL- és PH- utasítást hajtottunk végre, a megszakítás előtti állapotregisztert kapjuk, és az utasításszámlálóba a megszakításkor elmentett cím kerül. A program így a megszakítás előtti helytől folytatódhat tovább.

Példák: Az RTI utasítás általában a megszakításokat kezelő rutin vége szokott lenni, és csak akkor kerül végrehajtásra, ha legalább egy megszakítás generálódik. Önmagában is használhatjuk az RTI utasítást. Például az

LDA CH / PHA/ LDA CL / PHA / LDA SREG / PHA / RTI

utasítássorozat az SREG értékét az állapotregiszterbe tölti és feltétel nélküli vezérlésátadást hajt végre a CH, CL byte-ok által meghatározott címre.

RTS

Visszatérés alprogramból. PCL: = S, PCH: = S, SP: = SP + 2.

Utasítás	Címzési mód	Hossz	Ciklusok száma
$60 96	RTS	1	6

Állapotregiszter: nem változik.

Végrehajtás: Az RTS utasítás a verem tetején levő két byte-ot az utasításszámlálóba tölti, majd az utasításszámláló értékét megnöveli eggyel. Így a legutolsó JSR utasítástól folytatódik a program végrehajtása.

Példák: Az RTS általában a RETURN gépi kódú megfelelőjeként használható. Önmagában mint feltétel nélküli vezérlésátadó utasítás is használható, lásd a 4. fejezet bevezetőjét.

SBC

A memória tartalmát az átvitel bittel (C) együtt kivonja az akkumulátorból; A: = A-M-(1-C).

Utasítás		Címzési mód	Hossz	Ciklusok száma
$E1	225	SBC (0. lap, X)	2	6
$E5	229	SBC 0. lap	2	3
$E9	233	SBC #érték	2	2
$ED	237	SBC abszolút	3	4
$F1	241	SBC (0. lap), Y	2	5 *
$F5	245	SBC 0. lap, Y	2	4
$F9	249	SBC abszolút, Y	3	4 *
$FD	253	SBC abszolút, X	3	4 *

Állapotregiszter: NV-BDIZC
XX XX

Végrehajtás: A címzési módnak megfelelő byte kivonódik (az átvitellel együtt) az akkumulátor tartalmából. Ha ezután a C bit még mindig magas, az azt jelenti, hogy nincs átvitel, az akkumulátor nagyobb vagy egyenlő volt, mint amit kivontunk belőle. A chip a kivonást úgy hajtja végre, hogy a kivonandó egyes komplemensét és a C-t hozzáadja az akkumulátorhoz. N,V,Z és C értékei ennek megfelelően állítódnak be.

Példák: Kivonás előtt a C bitet magasra kell állítani (jelezve, hogy nincs átvitel):

SEC / LDA BYTE1 / SBC BYTE2

Két byte-os számok kivonásakor természetesen csak legelőször kell a C bitet állítani. A következő utasítássor a CL,CH címen levő 2 byte-os számból von ki $AA12-t:

SEC / LDA CL / SBC #$12 / LDA CH/SBC #$AA

SEC

Magasra állítja a C bitet; C: = 1.

Utasítás	Címzési mód	Hossz	Ciklusok száma
$38 56	SEC	1	2

Állapotregiszter: NV-BDIZC
1

Végrehajtás: értelemszerű.

SED

Magasra állítja a D bitet; D: = 1.

Utasítás	Címzési mód	Hossz	Ciklusok száma
$F8 248	SED	1	2

Állapotregiszter: NV-BDIZC
1

Végrehajtás: A D bit magasra állítása utasítja a processzort a BCD aritmetikai használatára. Az ADC és SBC utasítások **helyesen** számolnak ebben az esetben is, de csak a C jelzőbit állítódik helyesen. V, Z, N nem használható közvetlenül.

SEI

A maszkolható megszakítást letiltó bitet magasra állítja; I: = 1

Utasítás	Címzési mód	Hossz	Ciklusok száma
$78 120	SEI	1	2

Állapotregiszter: NV-BDIZC
1

Végrehajtás: Az I bit magasra állítása letiltja a maszkolható megszakításokat. A C-64 hardver megszakító rendszere – normál körülmények között – egy másodperc alatt 50 megszakítást generál. A SEI után ezekre – egészen a CLI utasítás végrehajtásáig – nem kerül sor.

Példák: A megszakításokat kezelő rutinok általában ezzel az utasítással kezdődnek, hogy megakadályozzák további megszakítások generálását.

STA

Az akkumulátor tartalmát a memóriába tölti; M: = A.

Utasítás		Címzési mód	Hossz	Ciklusok száma
$81	129	STA (0. lap, X)	2	6
$85	133	STA 0. lap	2	3
$8D	141	STA abszolút	3	4
$91	145	STA (0. lap), Y	2	6
$95	149	STA 0. lap, X	2	4
$99	153	STA abszolút, Y	3	5
$9D	157	STA abszolút, X	3	5

Állapotregiszter: nem változik.

Végrehajtás: Az akkumulátor tartalmát a címzési módnak megfelelő byte-ba tölti.

STX

Az X regiszter tartalmát a memóriába tölti; M: = X

Utasítás		Címzési mód	Hossz	Ciklusok száma
$86	134	STX 0. lap	2	3
$8E	142	STX abszolút	3	4
$96	150	STX 0. lap, X	2	4

Állapotregiszter: nem változik.

Végrehajtás: Az akkumulátor tartalmát a címzési módnak megfelelő byte-ba tölti.

STY

Az Y regiszter tartalmát a memóriába tölti; M: = Y

Utasítás		Címzési mód	Hossz	Ciklusok száma
$84	132	STY 0. lap	2	3
$8C	140	STY abszolút	3	4
$94	148	STY 0. lap, X	2	4

Állapotregiszter: nem változik.

Végrehajtás: az akkumulátor tartalmát a címzési módnak megfelelő byte-ba tölti.

TAX

Az akkumulátor tartalmát az X regiszterbe tölti; X: = A

TAY

Az akkumulátor tartalmát az Y regiszterbe tölti; Y: = A.

TXA

Az X regiszter tartalmát az akkumulátorba tölti; A: = X.

TYA

Az Y regiszter tartalmát az akkumulátorba tölti; A: = Y.

Utasítás		Címzési mód	Hossz	Ciklusok száma
$AA	170	TAX	1	2
$A8	168	TAY	1	2
$8A	138	TXA	1	2
$98	152	TYA	1	2

Állapotregiszter: NV-BDIZC
X X

Végrehajtás: értelemszerűen.

TSX

A verem mutató tartalmát az X regiszterbe tölti; X: = SP.

Utasítás	Címzési mód	Hossz	Ciklusok száma
$BA 186	TSX	1	2

Állapotregiszter: nem változik.

Végrehajtás: Az X regiszter tartalmába kerül a verem mutató aktuális értéke. A verem mutató értékét közvetlenül ezzel az egy utasítással lehet lekérdezni.

TXS

Az X regiszter tartalmát a verem mutatóba tölti; SP: = X

Utasítás	Címzési mód	Hossz	Ciklusok száma
$9A 154	TXS	1	2

Állapotregiszter: nem változik.

Végrehajtás: Az X regiszter tartalma lesz a verem mutató aktuális értéke. A PH, PL utasítások ennek megfelelően hajtódnak végre. A verem mutató értékét közvetlenül ezzel az egy utasítással lehet állítani. Például LDX #$FF/TXS általában a verem törlésének felel meg.

9.3 A Commodore-64 memóriaszervezése

A C-64 64 Kbyte RAM-ot és további 20 Kbyte ROM-ot, azaz csak olvasható memóriát tartalmaz. Ez utóbbi a BASIC interpretert, az operációs rendszert és a standard karakterkészletet tartalmazza. A C-64 a memória 4K-s darabját az I/O eszközökkel való kapcsolatra használja.

A 6510-es processzor számára természetesen mindebből egyidőben legfeljebb 64 Kbyte 'létezhet'. Hogy melyik, azt a 6510 I/O kapuja szabályozza, amelyik a memória $0001 címén található. A hozzá tartozó adat irány-regiszter a $0000 címen található. Ennek bitjeit — normál állapotban — a következőképpen kell beállítani:

Bit	5	4	3	2	1	0
Érték	1	0	1	1	1	1

(1 = kimenet, 0 = bemenet)

A kapu egyes bitjeinek a jelentése a következő:

LORAM (0.bit) A bit a BASIC ROM $A000-$BFFF 'jelenlétét' szabályozza. Ha a bit magas, a ROM-ot használhatjuk. Ha alacsony, a 'mögötte levő' RAM-ot 'látja' a processzor.

HIRAM (1.bit) A bit a KERNAL ROM ($E000-$FFFF) jelenlétét szabályozza. Ha a bit magas, a ROM-ot használhatjuk. Ha alacsony, a mögötte levő RAM-ot látja a processzor.

CHAREN (2.bit) A bit a standard karakterkészlet ROM-jának jelenlétét szabályozza. Ez ugyanott helyezkedik el, ahol az I/O eszközök ($D000-$DFFF). Ha a bit magas, a processzor az I/O eszközöket használja, és a karakter ROM elérhetetlen (lásd erről a 7. fejezet megfelelő részeit!). Ha a bit alacsony, a karakter ROM közvetlenül elérhető a processzor számára.

További hardver lehetőségek biztosítják, hogy a bővítőre (cartridge) csatlakoztatott ROM-ok elérhetők legyenek. Erre szolgálnak a bővítő EXROM és GAME vonalai. Ezek semlegesíthetik a CHAREN bit hatását.

3. bit A kazettás egység írás vonala.
4. bit A kazettás egység billentyűzetét olvasó vonal.
5. bit A kazettás egység motorját szabályozó vonal.

9.4 KERNAL alprogramok

Az előző paragrafusban is említettük, hogy a $E000-$FFFF címeken levő 8 Kbyte-os ROM-ot KERNAL ROM-nak is hívjuk, lévén ez a C-64 operációs rendszerét tartalmazó memóriarész. (Ez így nem teljesen igaz, mert a KERNAL valójában kisebb, a BASIC Interpreter egyes részei is ebben a ROM-ban helyezkednek el.)

A továbbiakban az ún. **KERNAL rutinokat** ismertetjük. Ezek azok a rutinok, melyek belépési pontjait a $FF81 címen kezdődő táblázat tartalmazza. Néhány rutin a 3. lapon levő RAM-címeket használja belépési pontként, így ezeket tetszés szerint átírhatjuk. Így például – a BASIC bolygatása nélkül – lehetőség nyílik egy nem CBM kompatibilis nyomtató illesztéséhez, mondjuk 5-ös egységszámmal. Ezek a rutinok a következők:

($31A) = OPEN
($31C) = CLOSE
($31E) = CHKIN
($320) = CHKOUT
($322) = CLRCHN
($324) = CHRIN
($326) = CHROUT
($328) = STOP
($32A) = GETIN
($32C) = CLALL
($330) = LOAD
($332) = SAVE

A KERNAL rutinok az átvitel (C) bitet használják hibajelzésre. Ha a bit alacsony (C = 0), a rutin hiba nélkül lefutott. Ha magas, a rutin nem hajtódott szabályszerűen végre és a hiba kódja ilyenkor az akkumulátorba (A regiszter) kerül. Ennek értékei a következők lehetnek:

Érték	Jelentés
0	A rutin futását a STOP lenyomásával megállítottuk.
1	Túl sok megnyitott file.
2	A file-t már megnyitottuk.
3	A file nincs megnyitva.
4	A file-t nem találja a rendszer.
5	Az egység nincs jelen.
6	A file nem egy input file.
7	A file nem egy output file.
8	A file neve hiányzik.
9	Nem megengedett egységszám.
240	Az RS232 megnyitása/lezárása módosította a BASIC munkaterület vége mutatót.

Az alprogramok nevei az eredeti Commodore nevek. Ezek értékét az assembler listákon meg kell adni. A felsorolásban ezenkívül megadtuk a rutinok által használt regisztereket is. Ezek egy részébe kell a bemeneti adatokat tölteni, illetve ezek adják az eredményt. Néhány rutin a RAM bizonyos részeibe is ír, illetve onnan olvas. Vannak rutinok, amelyek használata előtt további rutinok meghívására van szükség, ezeket soroltuk fel az 'előkészítő rutinok' címszó alatt.

ACPTR

Belépési pont: $FFA5 (65445)
Használt regiszterek: A, X
Előkészítő rutinok: TALK, TKSA

Az alprogram a TALK és az esetleges TKSA rutinokkal "BESZÉLŐ" állapotba hozott perifériális egységről olvas be egy byte-ot az A regiszterbe. Az esetleges hibáról a BASIC állapotváltozója ad tájékoztatást.

CHKIN

Belépési pont: $FFC6 (65478)
Használt regiszterek: A, X
Előkészítő rutinok: OPEN

A KERNAL OPEN alprogramja segitségével megnyitott bármely logikai file-t lehet input csatornaként definiálni a fenti rutin segitségével. A logikai file-t előzőleg meg kell nyitni, majd a logikai file számot az X regiszterbe kell tölteni. A CHRIN és a GETIN rutinok használata előtt – ha nem a billentyűzetről viszünk be adatokat – használnunk kell a CHKIN rutint.

CHKOUT

Belépési pont: $FFC9 (65481)
Használt regiszterek: A, X
Előkészítő rutinok: OPEN

A KERNAL OPEN alprogramja segítségével megnyitott bármely logikai file-t lehet output csatornaként definiálni a fenti rutin segítségével. A logikai file-t előzőleg meg kell nyitni, és a logikai file számot a CHKOUT meghívása előtt az X regiszterbe kell tölteni.

CHRIN

Belépési pont: $FFCF (65487)
Használt regiszterek: A, X
Előkészítő rutinok: OPEN, CHKIN

A rutin segítségével a CHKIN rutinnal már megnyitott csatornán egy byte-ot olvashatunk be, ami az A regiszterbe kerül. A billentyűzetről való input speciális formában valósul meg. A képernyőn megjelenik a villogó kurzor, és egy teljes sort gépelhetünk be. A RETURN billentyű megnyomása után a teljes sor a BASIC input pufferébe kerül, és a CHRIN onnan olvassa egyesével a karaktereket. A RETURN elérésekor az eljárás elölről kezdődik.

CHROUT

Belépési pont: $FFD2 (65490)
Használt regiszterek: A
Előkészítő rutinok: CHKOUT, OPEN

A rutin egy már megnyitott csatornára elküldi az A regiszterben levő byte-ot. Ha a CHKOUT, OPEN előkészítő rutinokat nem használjuk, az outputként elküldött byte a képernyőre kerül.

CIOUT

Belépési pont: $FFA8 (65448)
Használt regiszterek: A
Előkészítő rutinok: LISTEN, SECOND

A CIOUT rutin segítségével a már előzetesen 'HALLGATÓ' állapotban levő perifériális egységre az A regiszterben levő egyetlen byte-ot küldi el. A rutin egy 1 byte-os puffert használ, és a CIOUT minden egyes meghívásakor az **előző** byte-ot küldi el. Az utolsó byte az UNLSN meghívásakor kerül elküldésre.

CINT

Belépési pont: $FF81 (65409)
Használt regiszterek: A,X,Y
Előkészítő rutinok: -

A CINT rutin inicializálja a képernyő szerkesztőt és a VIC II chipet.

CLALL

Belépési pont: $FFE7 (65511)
Használt regiszterek: A,X
Előkészítő rutinok: -

A CLALL rutin lezárja az összes nyitott file-t. A táblázatok foglaltságát jelző mutatók törlődnek (lásd a 4. fejezetben az OPEN utasítást!). Ezzel egyidőben az I/O csatornák kezdeti (default) értékei visszaállítódnak.

CLOSE

Belépési pont: $FFC3 (65475)
Használt regiszterek: A,X,Y
Előkészítő rutinok: -

A CLOSE az akkumulátorban levő logikai file-számnak megfelelő file-t lezárja. Ekvivalens a BASIC CLOSE lf utasítással.

CLRCHN

Belépési pont: $FFCC (65484)
Használt regiszterek: A,X
Előkészítő rutinok: -

A rutin az I/O csatornák kezdeti (default) értékét állítja viszsza. Amennyiben a nyitott I/O csatornák egyike a soros busz volt, a megfelelő UNLSN, illetve UNTLK rutinok is végrehajtódnak.

GETIN

Belépési pont: $FFE4 (65508)
Használt regiszterek: A,X,Y
Előkészítő rutinok: CHKIN, OPEN

A rutin a billentyűzet, illetve az RS232-es csatorna pufferből egyetlen byte-ot az akkumulátorba tölt. A pufferekbe töltést a hardver megszakító rutinok végzik. Ha a csatorna a soros busz, a kazettás egység vagy a képernyő, akkor a megfelelő input rutin kerül végrehajtásra.

IOBASE

Belépési pont: $FFF3 (65523)
Használt regiszterek: X,Y
Előkészítő rutinok: -

A rutin az I/O eszközöket használó memóriarész kezdőcímével tér vissza az X, Y regiszterekben. X tartalmazza a cím alsó, Y pedig a felső byte-ját.

IOINIT

Belépési pont: $FF84 (65412)
Használt regiszterek: A,X,Y
Előkészítő rutinok: -

A rutin a C-64 valamennyi I/O eszközét inicializálja.

LISTEN

Belépési pont: $FFB1 (65457)
Használt regiszterek: A
Előkészitő rutinok: -

A rutin az A regiszterben levő (0-31 közé eső) számnak megfelelő perifériális eszközt utasítja, hogy álljon készen adatok fogadására. Ennek hatására a szóban forgó egység 'HALLGATÓ' állapotba kerül.

LOAD

Belépési pont: $FFD5 (65493)
Használt regiszterek: A,X,Y
Előkészítő rutinok: SETLFS, SETNAM

A rutin az input eszközről a memóriába tölt egy file-t. A LOAD a hasonló nevű BASIC utasítás gépi kódú analógja. A LOAD ellenőrzésre is használható. A betöltendő, vagy ellenőrizendő file adatait a SETLFS és SETNAM rutinok segítségével a megfelelő helyre be kell tölteni. Utána az A regiszterbe 0-t kell írni, ha tölteni, s 1-et, ha ellenőrizni akarunk. Az X,Y regiszterpár a töltés kezdőcímét tartalmazza. Ha a file-t úgy nyitottuk meg, hogy a <mód> paraméter értéke 0, a töltés ezen a címen kezdődik. Ha a paraméter értéke 1 volt, a file első két karaktere adja meg a töltés kezdőcímét. Szalagos file esetén ezt az információt a fejléc tartalmazza.

MEMBOT

Belépési pont: $FF9C (65436)
Használt regiszterek: X,Y
Előkészítő rutinok: -

A rutinok az X,Y regiszterpár tartalmát a 'BASIC munkaterület eleje' mutatóba tölti (normál értéke $0800), feltéve, hogy a C jelzőbit alacsony. Ellenkező esetben (C = 1) a mutató aktuális értéke kerül az X és Y regiszterbe.

MEMTOP

Belépési pont: $FF99 (65433)
Használt regiszterek: X,Y
Előkészítő rutinok: -

Ha a C jelzőbit alacsony, a rutin az X, Y regiszterpár tartalmát a 'BASIC munkaterület vége' mutatóba tölti. Ellenkező esetben (C = 1) a mutató aktuális értéke kerül az X és Y regiszterbe.

OPEN

Belépési pont: $FFC0 (65472)
Használt regiszterek: A,X,Y
Előkészítő rutinok: SETLFS, SETNAM

A rutin a SETLFS és SETNAM rutinokkal beállított paraméterű file-t megnyitja. Hasonló az OPEN BASIC utasításhoz.

PLOT

Belépési pont: $FFF0 (65520)
Használt regiszterek: A,X,Y
Előkészítő rutinok: -

A rutin segítségével írhatjuk/olvashatjuk a kurzor pozícióját. Ha a rutin meghívásakor a C jelzőbit magas, az eljárás az X,Y regiszterekbe tölti a kurzor helyét ($0 < X < 39$, $0 < Y < 24$). Ha a rutint úgy hívjuk meg, hogy C alacsony, az X,Y regiszterpár értékének megfelelően átállítja a kurzor pozícióját.

RAMTAS

Belépési pont: $FF87 (65415)
Használt regiszterek: A,X,Y
Előkészítő rutinok: -

A rutin teszteli a RAM-ot, és ennek megfelelően beállítja a memória elejét és végét jelző mutatókat. A $0000-$0101, illetve $0200-$03FF címekbe 0 byte-ot ír. A rutin lefoglalja a kazetta puffert, és a képernyő memória mutatóját $0400-ra állítja.

RDTIM

Belépési pont: $FFDE (65502)
Használt regiszterek: A,X,Y
Előkészítő rutinok: -

A rutin a regiszterekbe tölti a 0. lapon levő óra regisztereit. Az óra a másodpercek 60-ad részét számolja, és egy másodpercben mintegy 50-szer aktualizálja a hardver megszakító rendszer. Az A regiszter tartalmazza a legnagyobb, Y az azt követő, végül X a legkisebb helyiértékű byte-ot. RDTIM a BASIC TI változó olvasásának felel meg.

READST

Belépési pont: $FFB7 (65463)
Használt regiszterek: A
Előkészítő rutinok: -

Az I/O eszközök állapotát jelző, 0. lapon található byte-ot az akkumulátorba tölti. A READST rutin meghívása a BASIC ST változója olvasásának felel meg. Az egyes bitek jelentését a 9.4 fejezetben adtuk meg.

RESTOR

Belépési pont: $FF8A (65418)
Használt regiszterek: A,X,Y
Előkészítő rutinok:

A rutin az operációs rendszer által használt vektorokat újratölti.

SAVE

Belépési pont: $FFD8 (65496)
Használt regiszterek: A,X,Y
Előkészítő rutinok: SETLFS, SETNAM

A rutin a memória egy adott részét a SETLFS és SETNAM rutinok segítségével már specifikált file-ba elmenti. A SAVE alprogram meghívása előtt az A regiszterbe egy, a 0. lapon levő cím mutatóját kell tölteni. A cím az elmentendő memóriarész első byte-ja lesz. Az első – **már nem elmentendő** – byte címét az X,Y regiszterpár kell, hogy tartalmazza (X a kisebb, Y a nagyobbik helyiértékű byte-ot).

SCNKEY

Belépési pont: $FF9F (65439)
Használt regiszterek: A,X,Y
Előkészítő rutinok: IOINIT

A rutin ellenőrzi a C-64 billentyűzetét, és a megnyomott billentyű ASCII kódja a billentyűzet pufferbe kerül. A rutint csak abban az esetben érdemes meghívni, ha a C-64 hardver megszakító rendszerét kikapcsoltuk.

SCREEN

Belépési pont: $FFED (65517)
Használt regiszterek: X,Y
Előkészítő rutinok: -

A rutin az X,Y regiszterpárban a képernyő formátumával tér viszsza. Ez X=40, Y=25. A KERNAL nemcsak a C-64-en fut, és más gépeken ez a formátum eltérő lehet (pl. 80*25.)

SECOND

Belépési pont: $FF93 (65427)
Használt regiszterek: A
Előkészítő rutinok: LISTEN

A rutin a LISTEN rutin segítségével már 'HALLGATÓ' állapotban levő eszköznek elküld egy megnyitási mód parancsot. Az eredeti angol terminológia: secondary adress. Az OPEN BASIC utasítás 3. paramétere. A rutin TALK után **nem** használható.

SETLFS

Belépési pont: $FFBA (65466)
Használt regiszterek: A,X,Y
Előkészítő rutinok: -

Az I/O műveletekben szereplő file-ok adatainak a memória meghatározott részén kell elhelyezkedniük. A SETLFS rutin a processzor regisztereiben elhelyezett adatokat a megfelelő helyre átmásolja. Az egyes regiszterek a következőket jelentik:

X = logikai file száma
A = egységszám
Y = megnyitási mód

(az OPEN utasítás első három paramétere)

SETMSG

Belépési pont: $FF90 (65424)
Használt regiszterek: A
Előkészítő rutinok: -

A rutin segítségével utasíthatjuk a KERNAL-t, hogy rendszer vagy hibaüzeneteket nyomtasson (a megfelelő táblázatok máshol kezdődnek). Az akkumulátor 6. és 7. bitjével jelezhetjük, melyiket akarjuk kiválasztani. Ha a 7. bit magas, a KERNAL hibaüzenetet nyomtat.

SETNAM

Belépési pont: $FFBD (65469)
Használt regiszterek: A,X,Y
Előkészítő rutinok: -

A rutin a BASIC OPEN – nem kötelező – 4. paraméterét valamint a LOAD és a SAVE <név> részét tölti a megfelelő helyre. Az akkumulátor tartalmazza a paraméter hosszát. Az X,Y regiszterpár pedig a memóriában a paraméter-sztring kezdőcímét jelenti. Az X,Y értékét a rutin meghívása előtt be kell állítani. A SETNAM használata az OPEN KERNAL rutin előtt **kötelező**! Az A = 0 érték jelenti, hogy nincs paraméter.

SETTIM

Belépési pont: $FFDB (65499)
Használt regiszterek: A,X,Y
Előkészítő rutinok: -

A rutin a RDTIM rutin 'inverze'. Az A,X,Y regiszter tartalmát átmásolja a 0. lapon levő órába. A tartalmazza a legnagyobb helyiértékű, Y pedig a legkisebb helyiértékű byte-ot.

SETTMO

Belépési pont: $FFA2 (65442)
Használt regiszterek: A
Előkészítő rutinok: -

A rutin abban az esetben használható, ha a bővítő egység kimenetébe egy IEEE kártyát helyeztünk. Ebben az esetben a rutin segítségével az IEEE busz **timeout** jelzőjét állítjuk be. A C-64 64 ezredmásodpercig vár a válaszra, és ha az nem érkezik meg, hibajelzést generál és abbahagyja a (handshake) szinkronizálást. Ha a rutin meghívásához az A regiszter 7. bitje magas, a timeout jelző beállítódik. Ha a bit alacsony, a jelző kikapcsolódik.

STOP

Belépési pont: $FFE1 (65505)
Használt regiszterek: A,X
Előkészítő rutinok: UDTIM

A rutin megállapítja, hogy az UDTIM hívásakor a STOP billentyű benyomott állapotban volt-e vagy sem. Ha igen, a Z jelzőbit magas lesz, és az I/O csatornák a kezdeti (default) értékre állítódnak vissza. Abban az esetben, ha a STOP billentyű nem volt benyomva, visszatéréskor az utoljára használt sornak megfelelő byte-ot fogja tartalmazni.

TALK

Belépési pont: $FFB4 (65460)
Használt regiszterek: A
Előkészítő rutinok: -

Az akkumulátorban megadott egységszámú perifériát utasítja, hogy legyen 'BESZÉLŐ'.

TKSA

Belépési pont: $FF96 (65430)
Használt regiszterek: A
Előkészítő rutinok: TALK

A rutin a TALK rutin segítségével már 'BESZÉLŐ' állapotban levő eszköznek elküld egy megnyitási mód parancsot. Az eredeti angol terminológia: secondary adress. Az OPEN BASIC utasítás 3. paramétere. A rutin a LISTEN után **nem** használható.

UDTIM

Belépési pont: $FFEA (65514)
Használt regiszterek: A,X
Előkészítő rutinok: -

A rutin aktualizálja a 0. lapon levő órát. Általában csak akkor kell meghivni, ha a hardver megszakító rendszer működését felfüggesztettük. Egyben ellenőrzi a STOP billentyű megnyomását is (lásd a STOP rutint).

UNLSN

Belépési pont: $FFAE (65454)
Használt regiszterek: A
Előkészítő rutinok: -

A rutin valamennyi – előzőleg már 'HALLGATÓ' állapotban levő – perifériális egységet 'FIGYELŐ' állapotba hozza. A többi perifériális egység állapota nem változik.

UNTLK

Belépési pont: $FFAB (65451)
Használt regiszterek: A
Előkészítő rutinok: -

A rutin valamennyi – előzőleg már 'BESZÉLŐ' állapotban levő – perifériális egységet 'FIGYELŐ' állapotba hozza. A többi perifériális egység állapota nem válto-zik.

VECTOR

Belépési pont: $FF8D (65421)
Használt regiszterek: A,X,Y
Előkészítő rutinok: -

A rutin segítségével olvashatjuk/írhatjuk a KERNAL által használt – RAM-ban levő – valamennyi vezérlésátadást végző vektor értékét. Ha a rutin meghívásakor a C jelzőbit magas, a rendszer valamennyi vektora az X,Y regiszterpárban megadott címmel kezdő listába kerül. Ha a C bit alacsony, a lista tartalma a vektorba töltődik.

Példák a KERNAL rutinok használatára

A legtöbb rutin a leírás alapján rögtön használható. Amiről külön is szólunk, az a soros buszhoz kapcsolódó KERNAL, valamint a SAVE rutinok használata.

Soros busz

A soros buszra kötött eszközök háromféle üzemmódban dolgozhatnak, háromféle állapotban lehetnek. Ezek a 'PARANCS', a 'BESZÉLŐ' és a 'HALLGATÓ' állapotok. A soros buszon egyedül a C-64 maga lehet 'PARANCS' állapotban, a perifériális egységek csak 'BESZÉLŐ', illetve 'HALLGATÓ' állapotban lehetnek. Van egy negyedik állapot is, amibe minden, a soros buszon levő bekapcsolt egység kerül, ez a 'FIGYELŐ' állapot. Ebben az esetben az eszköz nem csinál semmit, de mihelyt a C-64-től megkapja a parancsot, 'HALLGATÓ' vagy 'BESZÉLŐ' állapotba kerül.

A buszra kötött mindegyik eszköz fogadja a buszon áramló adatokat. Az egységeknek egyedi azonosítójuk, egységszámuk van. A C-64 ezt az egységszámot felhasználva utasíthatja az adott egységet, hogy 'HALLGATÓ' vagy 'BESZÉLŐ' állapotba kerüljön.

Amikor a COMMODORE-64 arra utasítja az eszközt, hogy működjön 'BESZÉLŐ'-ként az eszköz adatokat kezd küldeni a soros buszra. Mikor a COMMODORE-64 arra utasítja az eszközt, hogy 'HALLGATÓ'ként működjön, akkor a megcímzett periféria felkészül a COMMODORE-64, vagy más, a buszon levő eszköz által küldött adatok fogadására. Egyidejűleg csak egy 'BESZÉLŐ' lehet a buszon; máskülönben az adatok elvesznek. Ugyanakkor egyidejűleg tetszőleges számú eszköz 'hallgathatja' az egyetlen 'BESZÉLŐ'-t.

A függelék részletesen ismerteti a soros buszon való írás/olvasás, illetve szinkronizálás idődiagramjait. Ezt kezelni tudó bármely eszköz a buszra csatlakoztatható.

A C-64 – az IEEE szabványtól némileg eltérő értelemben – a következő parancs byte-okat képes elküldeni:

A parancs byte	felső négy bitje									
		0	1	2	3	4	5	6	7	E	F
	0			0	16	0	16	0	16	0	16
	1	GTL	LLO	1	17	1	17	1	17	1	17
	2			2	18	2	18	2	18	2	18
	3			3	19	3	19	3	19	3	19
	4	SDC	PPU	4	20	4	20	4	20	4	20
alsó	5			5	21	5	21	5	21	5	21
négy	6			6	22	6	22	6	22	6	22
bitje	7			7	23	7	23	7	23	7	23
	8	GET	SPE	8	24	8	24	8	24	8	24
	9	TCT	SPD	9	25	9	25	9	25	9	25
	A			10	26	10	26	10	26	10	26
	B			11	27	11	27	11	27	11	27
	C			12	28	12	28	12	28	12	28
	D			13	29	13	29	13	29	13	29
	E			14	30	14	30	14	30	14	30
	F			15	UNT	15	UNT	15	31	15	31
					LAG		TAG		SCG		SCGSCG

ACG (Addressed Command Group)
- GET = Group Execute Trigger
- PPC = Parallel Poll Configure
- TCT = Take Control
- GTL = Go To Local
- SDC = Selected Device Clear

UCG (Universal Command Group)
- DCL = Devices Clear
- PPU = Parallel Poll Unconfigure
- SPE = Serial Poll Enable
- LLO = Local Lockout
- SPD = Serial Poll Disable

LAG (Listen Addres Group) — UNL = Unlisten All Devices

TAG (Talk Addres Group) — UNT = Untalk All Devices)

SCG (Secondary Command Group)

Például az OPEN 3,8,3 "PELDA,S,W" utasítás végrehajtása során a C-64 a következő műveleteket végzi el:

(i) A BASIC megfelelő tábláiba bekerüla 3,8,3.
(ii) A 3-as csatorna megnyitásának megfelelő parancs byteot elküldi az interpreter. Ez a $F3 byte elküldését jelenti.
(iii) A 8-as eszközt 'HALLGATÓ' állapotba hozza, ami a $48 parancsbyte elküldését jelenti. Ezután kerül elküldésre a "PELDA,S,W" üzenet.

Az OPEN, TALK, SECOND stb. rutinok a parancs-byte előállítását automatikusan elvégzik. Részletes – a lemezegységet kezelő példák – az 5.7 paragrafusban találhatók.

A SAVE rutin használata

Következő példánk egy BASIC program, melynek segítségével a memória tetszőleges részét elmenthetjük a lemezre. A file neve pontosan négy karakter kell hogy legyen; ez csak a program egyszerűsítése miatt éppen 4. Utána a RENAME DOS parancs segítségével persze tetszőlegesen átnevezhető.

```
 10 POKE 780,1:POKE 781,8:POKE 782,255
 20 SYS 65466: REM SETLFS KERNAL
 30 INPUT "<CLR>A FILE NEVE (4 KARAKTER)";P$
 40 IF LEN(P$)<>4 THEN 30
 50 POKE 780,4:POKE 781,0:POKE 782,16
 60 FOR J=1 TO 4
 70 POKE 4095+J,ASC(MID$(P$,J,1))
 80 NEXT J
 90 SYS 65469 :REM SETNAM KERNAL
100 INPUT "KEZDOERTEK";A
110 POKE 780,251:POKE 251,A-INT(A/256)*256:POKE 252,INT(A/256)
120 INPUT "VEGERTEK+1";A
130 POKE 781,A-INT(A/256)*256:POKE 782,INT(A/256)
140 SYS 65496 :REM SAVE KERNAL
```

9.5 Monitorok, assemblerek, fordítók

A C-64 alapgép gyakorlatilag nem támogatja a gépi kódú programok, programrészek írását. Egyetlen lehetőségünk van gépi kódú programok írására: a POKE utasítás használatával kell a gépi kódú programrészt byte-onként beírni a memóriába. A PET gépek tartalmaztak gépi kódú monitort, aminek a helye érdekes módon a C-64-en is megtalálható: a $C000-$CFFF memóriát a gép **soha**, **semmire** sem használja. Egyszerű megoldás olyan monitort szereznünk, amelyik ezen a helyen helyezkedik el, és azt használni.

A különböző **gépi kódú monitorok** általában a következő feladatokat oldják meg:

(i) a memória adott címének kiírása, módosítása;
(ii) a memória adott részének mozgatása;
(iii) a memória adott részének elmentése;
(iv) töltés perifériális eszközökről;
(v) disassemblálás;
(vi) egyszerű assemblálás mnemonikok használatával.

A 9.6 paragrafusban a **HELP+** BASIC bővítés gépi kódú monitorának használatát ismertetjük.

A – főleg hosszabb – gépi kódú programok írásának alapvető feltétele, hogy egy jó minőségű **assemblerrel** rendelkezzünk.

Az assemblerek – a mnemonikok használatán túlmenően – biztosítják a címkék használatát és lehetővé tesznek bizonyos szerkesztési funkciókat (linkage-editor funkciók). Fejlettebb változatai biztosítják programkönyvtár, makrók stb. használatát.

A BASIC programok futását többé-kevésbé meggyorsítják a C-64-hez vásárolható **fordítóprogramok**, amelyek a BASIC szöveget gépi kódú programmá alakítják át. Lefordítani természetesen csak kipróbált és elég gyakran használt programokat érdemes.

Mind a fordítók, mind az assemblerek **csak lemezegységgel együtt** használhatók.

9.6 HELP +

A HELP + a PRINT TECHNIK 8K-s BASIC bővítése,amelyik igen szerencsésen egyesíti magában a BASIC bővítés, a gépi kódú monitor és az assembler funkcióit. A HELP + — szemben például a SIMONS' BASIC-kel — nem nyújt a **futó** programok számára további lehetőségeket, viszont minden lehetséges módon támogatja a C-64-en való programozási és kipróbálási munkát. A HELP + vagy a SIMONS' BASIC használata két különböző programozási stílust testesít meg. A közöttük való helyes választás gyakran a feladat típusától is függ.

Ebben a paragrafusban a HELP + lehetőségeit ismertetjük a következő sorrendben:

(i) BASIC utasítások;
(ii) monitor;
(iii) assembler.

A HELP + új BASIC utasításai

Mint a bevezetésben is említettük, a HELP + a futó BASIC programoknak nem biztosít új utasításokat; ezért az alábbi utasítások **csak parancs-módban használhatók.**

#A < programnév > , < egységszám >

Az utasítás a memóriában tárolt programhoz hozzáfűzi a < programnév > sztringben megadott nevű és < egységszám > -ú egységen levő programfile-t. Az utasítás végrehajtása közben a LOAD utasításnak megfelelő kijelzést kapunk.

#H

Hibaüzenettel való megállás **után** a fenti parancs hatására a képernyőre kerül a hibát közvetlenül okozó sor. Inverz alakban nyomtatódik ki a programsornak az a része, amit az interpreter már nem tudott értelmezni.

#F

Szintaxis: #F < alapszó >
#F < változó >
#F " < szöveg > "

A parancs hatására a program azon sorainak száma listázódik ki a képernyőre, amelyek tartalmazzák a megfelelő <alapszó>-t, <változó>-t, illetve <szöveg>-et. A <CTRL> billentyű segítségével a listázás lassítható, a <SHIFT> billentyű lenyomásával pedig tetszőleges időre megállítható. A <STOP> billentyű lenyomása megszakítja a listázást.

#D

Szintaxis: azonos a LIST utasításával.

A parancs a paraméternek megfelelő sorokat törli a programból.

#G [<kezdőérték>,<növekmény>]

A parancs a <kezdőérték>-től kezdődően automatikus sorszámozást biztosít. A programsor begépelése és a <RETURN> lenyomása után a programsor tárolódik és a <növekmény>-nyel megnövelt sorszám jelenik meg a következő sor elején.
Az utolsó sor bevitele (<RETURN>-nel való lezárása) után üssünk be egy <SHIFT-RETURN>-t. Ez megszünteti az automatikus sorszámozást. #G kiadása esetén <kezdőérték> = 100, <növekmény> = 10.

#T

A parancs hatására a RUN, GOTO, GOSUB és CONT utasítások után a program **nyomkövetési módban** kerül végrehajtásra. A képernyő első sorában a program utoljára végrehajtott 6 utasításának a sorszáma látszik. A sorszámokat az interpreter a $0122-$0130 címeken tárolja. A nyomkövetési módnak a #E parancs kiadása vet véget.

#S

A parancs utasítja az interpretert, hogy a tárolt programot a RUN, GOTO, GOSUB, illetve CONT után lépésenkénti módban hajtsa végre. Az interpreter a SHIFT billentyű minden egyes megnyomására egy sort hajt végre. Hasonlóan a #T parancshoz, a képernyő első sorában az utoljára végrehajtott 6 programsor száma látható. A #S hatása csak a #E parancs kiadásával szüntethető meg.

#E

Megszünteti a nyomkövetési vagy lépésenkénti üzemmódot.

#R [<kezdőérték>,<növekmény>]

A paraméternek megfelelően újrasorszámozza a programot. A hivatkozások értéke is megfelelően változik. A #R estén <kezdőérték> = 100, <növekmény> = 10.

#V

Kilistázza a program egyszerű változóit és aktuális értéküket a képernyőre. A listázás a <CTRL>, a <SHIFT> és a <STOP> bilentyűkkel szabályozható, mint a #F utasítás esetében.

#M

Kilistázza a program tömbjeit és aktuális értéküket a képernyőre. A listázás a <CTRL>, a <SHIFT> és a <STOP> billentyűkkel szabályozható, mint a #F utasítás esetében.

#L [<sorszám>]

A programot képernyőként listázza ki a <sorszám> sortól kezdődően. A <RETURN> megnyomásával a következő képernyőt kapjuk, a <CRSR> egyesével viszi rá a képernyőre a program sorait. A <STOP> billentyű befejezi a listázást.

!# <hexadecimális szám>

Az utasítás hatására a képernyőn megjelenik a <hexadecimális szám> négy hexadecimális számjegyből álló szám, decimális alakban.

!$ <decimális szám>

Az utasítás hatására a képernyőn megjelenik a <decimális szám> öt számjegyből álló szám, hexadecimális alakban.

#B

A BASIC NEW parancs hatását semlegesíti, de a program változóinak értékét nem kapjuk vissza!

([<egységszám>]

Hatása ekvivalens az OPEN 255, <egységszám>:CMD 255 BASIC utasításával. Az <egységszám> meg nem adása esetén az <egységszám> értéke 4 lesz.

)

Lezárja a (utasításban megnyitott CMD file-t.

£ <egységszám>

A DOS utasításokban az interpreter ezek után az <egységszám> értékét használja. Ha nem adjuk ki az utasítást, az interpreter automatikusan 8-cal dolgozik.

#K

Törli a HELP + bővítést.

#C

Tömöríti a programot. A parancs törli a megjegyzéseket, a felesleges szóközöket és automatikusan végrehajt egy #R0,1 parancsot is. A tömörített program futása gyorsabb.

#U

A parancs ellenőrzi, nincs-e olyan GOTO, GOSUB stb. utasítás, amelyik egy nem létező sorra hivatkozik. Minden egyes ilyen esetben egy UNDEF'D STATEMENT ERROR IN <sorszám> üzenetet kapunk.

*****

Ekvivalens a ?FRE(0) utasítással.

DOS parancsok

Hasonlóan a DOS 5.1 BASIC beszúráshoz a HELP + is lehetőséget biztosít bizonyos lemezegységre vonatkozó utasítások **egyszerűbb** beírására. Ezek nem új utasítások, pusztán könnyebb beírni őket. A leírásban d az utasításban definiált egységszámot jelenti. Kezdő értéke 8. A <programnév> az alábbi utasításokban nem tartalmazza az idézőjelet (")!

/ <programnév>

BASIC ekvivalens: LOAD "<programnév>",d

% < programnév >

BASIC ekvivalens: LOAD " < programnév > ",d,1

↑ < programnév >

BASIC ekvivalens: LOAD " < programnév > ",d és RUN.

← < programnév >

BASIC ekvivalens: SAVE " < programnév > ",d. Egyetlen különbség, hogy ha a file már létezik, megjelenik az **OVERWRITE JA/NEIN** üzenet. A J billentyű lenyomása felülírja a file-t, az N billentyű nem írja felül.

< < programnév >

BASIC ekvivalens: VERIFY " < programnév > ",d

@ [< parancs >]

Az utasítás segítségével írható/olvasható a 15. csatorna. Paraméterek nélkül a képernyőre kerül a hibaüzenet. Paraméterrel használva a paraméter értéke a 15. csatornába kerül outputként. Például:

N0:NEV,ID	= 'megformáz' egy új lemezt;
I	= inicializálja a lemezegységet;
B-R: 3 0 18 0	= BLOCK-READ (lásd az 5. fejezetet).

$

Az utasítás hatására az interpreter a képernyőre listázza a lemezegység katalógusát. A tárolt program érintetlen marad.

>

A 15. csatornáról beolvas egyetlen byte-ot és kiírja hexadecimális alakban. (Felhasználásáról lásd az 5. fejezetet!)

Monitor

A HELP + egy beépített kódú monitort is tartalmaz. A monitorba a

]

parancs kiadásával léphetünk be. A kilépésig a BASIC interpretert és szövegszerkesztőt nem használhatjuk. Belépéskor a képernyő következő sorába
xxxx yy
alakú információ jelenik meg.
Az xxxx az aktuális monitor cím hexadecimális alakban, yy pedig annak tartalma, ugyancsak hexadecimális alakban. A $ nem kerül kijelzésre.

A monitorból az
=
parancs kiadásával léphetünk ki.

Lehetőségünk van annak eldöntésére, hogy a képernyőre vagy a nyomtatóra kérjük-e az információt. A
[
parancs hatására az összes információ a képernyőre kerül, a
]
parancs viszont a nyomtatóra írja ki az információkat.

Belépéskor a monitor a C-64 memóriájából olvas, illetve ír. Lehetőség van azonban a lemezegység memóriájába való írásra, illetve az onnan való olvasásra. A
>
parancs hatására a monitor a C-64 memóriájából olvas, míg a
<
parancs kiadása után a lemezegység memóriáját olvassa.

A következőkben ismertetjük a monitor utasításait.

A monitor címét és annak tartalmát a számjegy (0-9), illetve az A-F billentyűk segítségével írhatjuk. Egy billentyű lenyomására a hexadecimális számjegy balról besiftelődik vagy a monitor címbe, vagy a neki megfelelő regiszterbe.

<RETURN>

Eggyel növeli a monitor címet.

↑

Eggyel csökkenti a monitor címet.

*

A monitor címtől elkezdi a programot futtatni. Ha ez a C-64 memóriájában volt, akkor az RTS utasításig a monitor futása felfüggesztődik. Ha a lemezegység memóriájában, akkor a monitor fut tovább.

+

A monitor cím írható.

/

A monitor címnek megfelelő memória tartalma írható.

-

Visszaassemblálja a monitor címen kezdődő első utasítást.

<szóköz>

A monitor címtől kezdődően folyamatosan visszaassemblálja a memória tartalmát. A visszaassemblálás sebessége a <CTRL>, a <SHIFT>, illetve a <STOP> billentyűkkel befolyásolható éppen úgy, mint a #F utasítás esetében.

@

A memória adott részét átmásolja. Az utasítás csak a C-64 memóriájában levő részeket mozgatja. A parancs kiadása előtt a mozgatandó memóriarész kezdő- és végcímét a 0. lapon levő bizonyos mutatókba be kell tölteni:
$22-$23: az első átmásolandó byte címe
$24-$25: az első már át nem másolandó byte címe (= utolsó + 1).

A @ parancs kiadása után a másolás a monitor címtől kezdődik.

Assembler

A HELP+ tartalmaz egy kétmenetes assemblert is. A könyvünkben szereplő gépi kódú rutinok általában a HELP+ assembler segítségével készültek. Egy tipikus assembler lista a következő oldalon látható.

Tipikus assembler lista

```
1000            10      ;**************
1000            20      ;*              *
1000            30      ;* BASIC BESZURAS  *
1000            40      ;*              *
1000            50      ;* HIVASA:      *
1000            60      ;*     SYS 828  *
1000            70      ;**************
033C            80      *=$033C
033C            90      ;A CHARGET ATALLITASA
033C A2 03      100     LDX #$03
033E BD 46 03   110 LP  LDA CIM-1,X
0341 95 72      120     STA $0072,X
0343 CA         130     DEX
0344 D0 F8      140     BNE LP
0346 60         150     RTS
0347 4C 4A 03   160 CIM JMP $034A
034A            170     ; ITT KEZDODIK AZ UJ CHARGET
034A E6 7A      180 UJCIM INC $7A
034C D0 02      190     BNE KI
034E E6 7B      200     INC $7B
0350 20 79 00   210 KI  JSR $0079
```

Mint említettük, az assembler egy szerkesztőből, egy fordítóból (ez a valódi assembler) és egy töltőből áll. A HELP + a BASIC szövegszerkesztőjét használja assembler forrásnyelvi programok írására. Ezek szerkesztéséhez a HELP + #F, #G, #D stb. utasításai is használhatók. A forrásnyelvi programot – megírása után – a SAVE utasítással egy lemezes file-ba kell tölteni. Egy assembler forrásnyelvi programsor a következő alakú:

<sorszám> <címke> <op.kód> <operandus> <megjegyzés>

Az egyes részek közt legalább egy szóköznek kell lennie. A <sorszám> a BASIC utasítások sorszámával azonos.

A <**címke**> nem kötelező, legfeljebb hat alfanumerikus jelből állhat, amelyek közül az első betű kell hogy legyen. A <címke> nem lehet azonos a 65xx mikroprocesszor család 56 mnemonikjával (lásd a 9.2 paragrafusban), továbbá az A, X és Y betűkkel.

Az <**op.kód**> a 9.2 paragrafusban felsorolt mnemokikok bármelyike lehet. Az <op.kód>-tól függ, hogy milyen <operandussal>-sal használhatjuk. Az <operandus> lehet egy null-sztring is.

A sort végül egy **megjegyzés** fejezheti be, amit egy pontosvessző (;) vezet be. A sor a <sorszám>-on kívül vagy az <op.kód>-ot (és a hozzá tartozó <címzési mód>-ot), vagy egy <megjegyzés>-t kell, hogy tartalmazzon.

A következőkben azokat az értékmegadási módokat soroljuk fel, amelyeket az <operandus> mezőben használhatunk.

Decimális szám: Előjel nélküli egész szám, amelyik a 0-65535 tartományba esik.
Hexadecimális szám: A $ jellel bevezetett hexadecimális szám, amelynek a $0000-$FFFF tartományba kell esnie.
Oktális szám: A @ jellel bevezetett, csak a 0-7 jegyeket tartalmazó szám. Értékének a 0-65535 tartományba kell esnie.
Bináris szám: A % jellel bevezetett, csak a 0, 1 jegyeket tartalmazó, legfeljebb 16 számjegyből álló szám.
ASCII szám: Az idézőjellel bevezetett egyetlen karakter. A szám a karakter ASCII kódja lesz.
Szimbólikus cim: A szimbólikus cím képzésének szabályai megegyeznek a címke képzési szabályaival.
Bytepár alsó/felső byte-ja: A numerikus vagy szimbólikus formában megadott 0-65535 közötti érték utáni < jel az alsó, a > jel a felső byte értékét jelöli. Például $FF02< = $02, $FF02> = $FF.
Szimbólikus módban szereplő operandusok: A fenti számokból és címekből a + és - műveletek felhasználásával épülhetnek fel. A műveleteket a fordító balról jobbra haladva végzi el.

A C-64 13 lehetséges címzési módját a következőképpen kell az assembler programsorban használnunk:

Címzési mód	Írásmód	Megjegyzés
közvetlen	#oper	1
relatív	oper	2
0. lapos	oper	1
0. lapos az X index	oper, X	1
0. lapos az Y index	oper, Y	1
indexelt indirekt	(oper, X)	1
indirekt indexelt	(oper), Y	1
abszolút	oper	3
abszolút az X indexszel	oper, X	3
abszolút az Y indexszel	oper, Y	3
abszolút indirekt	(oper)	3

1. Az operandusnak 1 byte hosszúnak kell lennie. Az assembler ilyen esetben automatikusan a 0. lapos címzéseket választja. Ha például CIM = $00AA, akkor a LDA CIM utasítást a 0. lapos címzéssel fordítja az assembler. Ha CIM = $03AA, akkor automatikusan abszolút címzést kapunk.
2. A relatív ugrásoknál az assembler az **abszolút címet** várja 2 byte hosszan, és abból számítja ki a relatív ugrás nagyságát. Ha a **cím túl messze van** (>127), a fordítás közben hibajelzést kapunk.
3. Ezekben az esetekben az operandusnak csak 2 byte-on tárolhatónak kell lennie.

Direktívák: Az assembler forrásnyelvi program a fenti típusú sorokon kívül olyan sorokat is tartalmazhat, amelyek a fordítás menetét vezérlik. Ezek a következők:

.BYTE: a direktíva után operandusok következhetnek, vesszővel elválasztva. Az értéküknek megfelelő byte-ok épülnek be a tárgyprogramba.

.WORD: a direktíva után egy operandus következhet. Az ennek megfelelő 2 byte-os szám az alsó, felső byte-nak megfelelő sorrendben épül be a tárgyprogramba, pl. .WORD $FFD2 hatására a $D2 és a $FF byte-ok (ebben a sorrendben) tárolódnak.

.TEXT: a direktívát idézőjelek (' vagy ") közt követő szöveg karaktereinek ASCII kódjai épülnek be a tárgyprogramba.

.DISP: a direktívát idézőjelek (' vagy ") közt követő szöveg karaktereinek képernyő kódjai épülnek be a tárgyprogramba.

A fenti négy direktívát a programban szereplő adatok tárolására használhatjuk.

.END: a direktíva utasítja az assemblert a fordítás befejezésére.

.LOAD: a direktívában szereplő file-név fordításával folytatódik az assemblálás. A programnév nem lehet rövidebb a már assemblált program nevénél. A HELP + ezt az egyetlen, linkage-editorra emlékeztető direktívát ismeri.

Értékadás: Lehetőség van címkék, szimbólikus címek értékének definiálására. A direktíva alakja:

```
    *          = <oper>
<címke>        = <oper>
```

* az éppen aktuális címet mutatja, ahová a következő utasítást a fordító elhelyezi. <oper> tetszőleges, a + és - műveletek segítségével felépülő operandus lehet.

Fordítás

A fordítást a
[
parancs kiadásával indíthatjuk el. Az assembler annak a file-nak a nevét kéri, amiben a forrásnyelvi program van, majd további két paramétert kér.

HEXA KORR-POKE : az eltolás mértékét definiálja. A fordító a tárgyprogramot a memóriában nem a forrásnyelvi programban megadott helytől tárolja, hanem a HEXA KORR-POKE értékének megfelelően,a memóriában visszatolva. Erre akkor lehet szükség, ha a tárgyprogram például a HELP + helyén van. Ha a forrásnyelvi program a * = $8000 utasítással kezdődik és HEXA KORR-POKE értéke $1000 volt, akkor a tárgyprogram a memóriába $7000-tól folyamatosan töltődik.

AUSDRUCK-CODE : a listázás paramétereit adja meg. A válasz háromjegyű bináris szám. Az egyes bitek a következőket jelentik:

1. bit magas: a hibás sorokat is kilistázza
alacsony: a hibás sorokat nem listázza
2. bit magas: a teljes forrásnyelvi programot listázza
alacsony: csak a hibás sorokat listázza
3. bit magas: a szimbólum-táblázatot is kilistázza
alacsony: a szimbólum-táblát nem listázza ki.

Ha mindhárom bit alacsony (=0 vagy szóközt írtunk be), csak a program hossza és a hibák száma kerül kiírásra.

A tárgykódot közvetlenül a memóriába helyezi a fordító.

Ha az assemblálást bekapcsolt nyomtatóval indítottuk el, az assembler listák a nyomtatóra is kiíródnak. A nyomtatón az assembler egy lapra 72 sort ír.

9.7 Gépi kódú programrészek felhasználása

Gépi kódú alprogramok

Már az előző fejezetek példáiban is láttuk, hogy a gépi kódú programrészek felhasználása mennyire hasznos és szükséges. Most röviden összefoglaljuk, hogyan lehet a gépi kódú programrészeket a BASIC-ből felhasználni. Erre összesen négyféle lehetőség kínálkozik:

(i) A SYS utasítás
(ii) A USR függvény
(iii) A RAM-ban levő I/O vektorok megváltoztatása
(iv) A RAM-ban levő megszakító vektorok megváltoztatása.

Ezek közül legegyszerűbb a SYS utasítás használata. A SYS X utasításra a BASIC interpreter működése megáll, és az X címtől kezdődő gépi kódú rutin kezd el futni. A rutinnak értékeket a POKE utasítással lehet átadni, és a PEEK utasítással lehet eredményt átvenni. Az 5-8. fejezetekben számos példát mutattunk ennek használatára.

A USR függvényt ritkábban használjuk. Ennek két oka is van. Egyrészt igaz, hogy segítségével közvetlenül értéket lehet az alprogramnak átadni, de csak egyet. Másrészről meghívása előtt a 785, 786-os címekre be kell tölteni a gépi kódú rutin kezdőcímét. A USR(X) végrehajtása – a paraméterátadástól eltekintve – egy JMP (xxxx) utasítással ekvivalens. A JMP byte a 784-es címen található.

A C-64 KERNAL rutinjainak egy része úgy van megírva, hogy a 3. lapon található címeknek megfelelő helyre kerül a vezérlés. Ez azt jelenti, hogy a címek, vektorok átírásával – és a megfelelő programrészek megírásával – módosíthatjuk az I/O műveletek végrehajtását. Ez valójában az interpreter átírását jelenti, és csak gyakorlott programozók vállalkozhatnak rá.

A megszakításokat kezelő rutinok is a 3. lapon található vektorokat használják. A megszakítást kezelő rutint annyiban könnyebb átírni, hogy általában csak kiegészítjük a hardver megszakító rutint és azután a régit teljes egészében végrehajtjuk. Erre példa a 7. fejezetben található 'villogtató' program.

Végül pár szót szólunk arról, hová helyezhetjük a gépi kódú rutinjainkat. Ha nem túl hosszú programról van szó, legegyszerűbben a kazetta pufferjében tárolhatjuk; feltéve persze, hogy nem használjuk a kazettás egységet. Másik lehetőség a $C000$CFFF memóriarész használata, amit az interpreter nem használ semmire. További lehetőség, hogy a gépi kódú programrészeket a BASIC munkaterület végére helyezzük el. Ebben az esetben természetesen gondoskodnunk kell arról, hogy a BASIC ezt ne írja felül. (Hasonlóan ahhoz, ahogy ezt a bit-térkép vagy a programozható karakterkészlet használata esetén tettük.)

Ennek megoldási sémája a következő lehet:

```
10 IF A=0 THEN A=1:LOAD "GEPI KOD",8,1
20 POKE 55,ALSO:POKE 56,FELSO:CLR
30 <program>
```

Az ALSO és FELSO értékét úgy kell megválasztanunk, hogy a gépi kódú rész az ALSO + 256*FELSO cím után helyezkedjék el. A CLR végrehajtásakor az interpreter a BASIC munkaterület végét erre a címre állítja be.

Gyakori megoldás az, hogy a gépi kódú programrész byte-jait DATA utasításokban tároljuk, majd egy egyszerű ciklussal a helyére töltjük:

```
10 FOR I=V TO W
20 READ X:POKE V,X
30 NEXT
```

Nehezebb azonban ennek a fordítottja; a memória adott részén tárolt adatoknak megfelelő DATA utasítások automatikus előállítása. Következő programunk ezt végzi el a 7. fejezetben levő, sprite-okat előállító program kiegészítéseként. Ezt követően már nem kell a sprite adatait előállítani, hanem a megfelelő DATA utasítások segítségével a sprite – lényegesen hamarabb – előállítható.

```
2000 V=S*64:W=V+63
2010 X=6000
2020 G=PEEK(57)+256*PEEK(58)
2030 J=1:PRINT"<CLR><11*CRSR-LE>";X
2040 IF V=W THEN LIST 6000-
2050 I%(J)=PEEK(V):J=J+1:V=V+1
2060 IF V=W THEN GOTO 2090
2070 IF J=8 THEN GOTO 2090
2080 GOTO 2050
2090 PRINT"<HOME>";X;"DATA ";
2100 FOR I=1 TO J-1
2110 PRINT MID$(STR$(I%(I)),2);",";
2120 NEXT I:PRINT "<CRSR BALRA> "
2130 PRINT"V=";V":W=";W;":X=";X;"+5"
2140 PRINT"<3*CRSR-LE>GOTO";G
2150 POKE 198,5:POKE 631,19:POKE 632,13
2160 POKE 633,13:POKE 634,13:POKE 635,13
2170 END
```

A programrész a V, W címek közti byte-okat tartalmazó utasításokat állít elő. Egy DATA utasítás 7 byte-ot tartalmaz. A megfelelő DATA utasítás 2090 utasításában és a 2100-2120 ciklusban íródik a képernyőre. A DATA beírására, a program továbbindítására a billentyűzet puffert használjuk; ebbe egy sor, a <RETURN> billentyűnek megfelelő 13-ast POKE-olunk be. A 2020-as sorban G-be a sorszámot töltjük be. Bárhogyan is számoztuk a programunkat, G-be a megfelelő sorszám kerül. Így a 2140. sorban a megfelelő GOTO utasítás kerül kiírásra.

A programot a többszínű sprite-okat előállító program kiegészítésére használjuk. Lefuttatása után a 10-1990 sorokat törölhetjük, és megmaradnak a megfelelő DATA soraink. A program természetesen nemcsak sprite-adatok, de gépi kódú programrészek tárolására is használható:

```
 10 INPUT"<CLR>S-LAP=";S
 20 CIM=S*64
 30 GOSUB 100
 40 PRINT"KERI-E A BYTE-OK ERTEKET?"
 50 GET A$:IF A$="" THEN GOTO 50
 60 IF A$<>"I" THEN END
 70 GOTO 2000
100 FOR I=0 TO 20:READ A$
110 FOR J=0 TO 2:T=0
120 FOR K=0 TO 3:X=0
130 G$=MID$(A$,K+1+4*J,1)
135 IF G$<>"." THEN X=ASC(G$)-64
138 IF G$="." THEN X=0
140 T=T+X*2+(6-2*K):NEXT K
150 POKE CIM+3*I+J,T:NEXT J:NEXT I
160 RETURN
```

```
170 REM  012345678901
180 DATA ......AA....
190 DATA .....ABCA...
200 DATA ....AABBBAAA
210 DATA ...AABBBA...
220 DATA .....ABA....
230 DATA .AA..ABA....
240 DATA ABBA.ABA....
250 DATA .ABA.ABA....
260 DATA ..AA..ABA...
270 DATA ...AABBBA...
280 DATA ..ACBCBCBA..
290 DATA .ACBCBCBCBA.
300 DATA .ABCBCBCBCA.
310 DATA ..ABCBCBAA..
320 DATA ...AAAAA....
330 DATA ....A..A....
340 DATA ...A....A...
350 DATA ..A......A..
360 DATA ...A......A.
370 DATA ..AAA....AAA
380 DATA ..A..A..A..A
390 REM  012345678901
```

BASIC beszúrások

A BASIC interpreter működését legegyszerűbben a CHRGET rutin átírásával módosíthatjuk. A CHRGET rutin összesen 24 byte-ból áll és a 0. lapon, a $0073 címen kezdődik:

```
          20         ;       A ($7A) MUTATO CSOKKENTESE
E6 7A     30 CHRGET  INC $7A
D0 02     40         BNE CHRGOT
E6 7B     50         INC $7B
          60         ;
          70         ;       A BASIC BYTE BETOLTESE
AD 00 02  80 CHRGOT  LDA $0200
C9 3A     90         CMP #$3A    ; UGRAS, HA A>=$3A
B0 0A    100         BCS KI
C9 20    110         CMP #$20    ; SZOKOZ KIHAGYASA
F0 EF    120         BEQ CHRGET
38       130         SEC
         140         ;     A C JELZOBIT ALACSONY, HA
         150         ;     A EGY SZAMJEGY ASCII KODJA
E9 30    160         SBC #$30
38       170         SEC
E9 D0    180         SBC #$D0
60       190 KI      RTS
```

Ezt a rutint használja a BASIC interpreter, hogy a BASIC szöveg következő, vagy aktuális – szóköztől különböző – byte-ját az akkumulátorba töltse. A rutinnak két belépési pontja van:

JSR $0073 A BASIC szöveg **következő** byte-jával tér vissza
JSR $0078 A BASIC szöveg **aktuális** byte-jával tér vissza.

Az egyes jelzőbiteket a rutin a következőképpen állítja be. A C jelzőbit alacsony, ha az akkumulátorba egy számjegy ASCII értéke ($30-$39) került, különben magas. A Z jelzőbit csak akkor magas, ha az akkumulátor egy 0 byte-ot vagy a kettőspont (:) ASCII értékét tartalmaz.

Az interpreter a többi jelzőbitet nem használja. Egyik legegyszerűbb eljárás a BASIC interpreter működésének módosítására a CHRGET átírása. Az ezzel a technikával készülő BASIC bővítéseket hívjuk **beszúrásoknak** (wedge), ezzel a technikával dolgozik a DOS 5.1, a HELP + és a könyvhöz kapható oktatólemez.

A SIMONS' BASIC struktúrája bonyolultabb, ez a tokenizáló, listázó stb. rutinokat teljes egészében átírja. A SUPERGRAFIK is a beszúrási technikát alkalmazza.

A legegyszerűbb eset, amikor az új BASIC utasítások ugyanazzal a jellel (@, #, !) kezdődnek. A következőben példát adunk olyan BASIC beszúrásra, amelyik lehetővé teszi a $ parancs használatát, a lemez katalógusának a memóriába való töltésére (ez nem azonos a HELP + $ parancsával, hanem a LOAD "$",8 paranccsal egyenértékű!).

```
033C             10          *=$033C
033C A2 03       20          LDX #$03
033E BD 46 03    30 LP       LDA CIM-1,X
0341 95 72       40          STA $0072,X
0343 CA          50          DEX
0344 D0 F8       60          BNE LP
0346 60          70          RTS
0347 4C 48 03    80 CIM      JMP $034A
034A E6 7A       90 UJCIM    INC $7A
034C D0 02      100          BNE KI
034E E6 7B      110          INC $7B
0350 20 79 00   120 KI       JSR $0079
0353 C9 24      130          CMP #$24
0355 D0 11      140          BNE VISSZA
0357 98         150          TYA
0358 48         160          PHA
0359 A0 09      170          LDY #$09
035B B9 6B 03   180 LP1      LDA SZOVEG,Y
035E 91 7A      190          STA ($7A),Y
0360 88         200          DEY
0361 30 03      210          BMI LP2
0363 4C 5B 03   215          JMP LP1
0366 68         220 LP2      PLA
0367 A8         230          TAY
0368 4C 79 00   240 VISSZA   JMP $0079
036B 4C 4F 41   250 SZOVEG   .TEXT "LOAD"
036F 22         260          .BYTE 34
```

```
0370 24        270        .TEXT "$"
0371 22        280        .BYTE 34
0372 2C 38     290        .TEXT ",8"
0374 00        300        .BYTE 0
0375           310        .END3
```

9.8 Egysoros 'INPUT' rutin

A BASIC INPUT utasításnak a Commodore gépeken való realizálása nem megfelelő, erről a 4. fejezetben az INPUT ismertetésekor részletesen szóltunk. A jelen részben egy olyan – gépi kódú – szubrutint adunk, amelyik ezeket a hiányosságokat kiküszöböli.

A rutin lehetővé teszi, hogy a képernyő – X,Y koordinátákkal – adott pontjától egy L hosszúságú mezőt definiáljunk, s abba tetszőleges adatot olvassunk be. A vezérlő karakterek egy része hatástalan marad, mások pedig a megszokottól eltérően működnek. A rutint a

```
SYS 49152,Y,X,L,S$
```

BASIC paranccsal lehet meghívni, ahol X és Y a vizszintes és függőleges koordináták (0<=X<=39,0<=Y<=24), L a mező hossza, S$ pedig egy **már létező, pontosan L hosszú sztring**. Ezt legyegyszerűbben a

```
SP$ = "                                        "(Pontosan 40 darab szóköz!)
S$ = RIGHT$(SP$,L)
```

parancsok segítségével hozhatjuk létre.

A rutin a képernyő adott helyére kiírja az S$ sztringet, majd a kurzor a mező első pozíciójára kerül. Ezután – hasonlóan a teljes képernyős szerkesztőhöz - lehetőség van az S$ új értékének a beírására. Szemben azzal azonban a rutin biztosítja, hogy a mezőn kívülre semmilyen körülmények közt sem kerülhetünk. Pl. ha a kurzor a mező elején van, akkor a <CRSR-BALRA> billentyű hatástalan.

A szerkesztésre az alábbi vezérlő billentyűket használhatjuk:

- <CRSR-JOBBRA> - egy karakterhellyel jobbra lépteti a kurzort;
- <CRSR-BALRA> - egy karakterhellyel balra lépteti a kurzort;
- <INST> - beszúr egy karaktert. A mező végén levő karakter elvész;
- <DEL> - törli a kurzor előtt levő karaktert, s a kurzort egy hellyel előbbre lépteti. A mező végén egy szóköz karakter jelenik meg.
- <F7> - törli a kurzor alatti karaktert, s a mező maradék részét előre lépteti. A mező végére egy szóköz karakter kerül.
- <HOME> - a mező elejére állítja a kurzort.
- <CLR> - törli (szóközökkel tölti fel) a mezőt.
- <F1> - visszaállítja az S$ eredeti értékét.

A rutinból a <CRSR-FEL>, <CRSR-LE>, <RETURN> és az <F3> billentyűk megnyomásával lehet kilépni. A 49155 címre ennek megfelelően 1,2,3 vagy 4 kerül.

A rutin gépi kódú listája az alábbi:

```
*=$c000
;
; sys 49152,x,y,l,s$
; a kepernyo x,y pontjatol l hosszan bekeri az s$ valtozot
; aminek mar leteznie kell legalabb l hosszan
; 49155 a viszateres kodja
; szerkesztes:
;  <home>   = elso karakter
;  <clr>    = mezo torlese
;  <jobbra> = kurzor jobbra
;  <balra>  = kurzor balra
;  <f1>     = s$ kezdeti erteke
;  <fel>    = kilepes 1
;  <le>     = kilepes 2
;  <return> = kilepes 3
;  <f3>     = kilepes 4
;  <ins>    = karakter beszuras
;  <del>    = karakter torles
;  <f7>     = kurzor alatti karakter torlese
;
fac    = $61    ; lebegőpontos akkumulator
chrget = $73    ; BASIC szöveg következő karaktere
chrgot = $79    ; BASIC szöveg aktuális  karaktere
mutato = $fc    ; az eredeti sztring karaktereire mutat
strmut = $47
hossz  = $fe    ; a beolvasando sztring hossza
kp     = $ff    ; a mezo aktualis karaktere
frmevl = $ad9e  ; BASIC kifejezés kiértékelése
facint = $b1aa  ; valós --> egész konverzió
adress = $b08b  ; valtozó címének megkeresése
chkout = $ffc9  ; KERNAL rutinok
chrout = $ffd2
clrchn = $ffcc
chkin  = $ffc6
getin  = $ffe4
plot   = $fff0
;
       jmp beolv ; változók átlépése
;
error  .byte 0   ; visszatérési kód
xkoord .byte 0   ; a mező kezdete (X)
ykoord .byte 0   ; a mező kezdete (Y)
billen .byte 0   ; az aktuális billentyű kódja
;
; paraméterek beállítása
beolv  jsr chrget ; következő BASIC jel
       jsr frmevl ; formula kiértékelése
       jsr facint ; egész számmá konvertálás
       ldx $65    ; az alsó byte
       stx ykoord ;  tárolása
```

```
        jsr chrgot ; az utolsó BASIC jel
        cmp #','   ;  vessző kell hogy legyen
        beq tov1
        rts        ; ha nem --> vége
tov1    jsr chrget ; következő BASIC jel
        jsr frmevl ; formula kiértékelése
        jsr facint ; egész számmá konvertálás
        ldy $65    ; az alsó byte
        sty xkoord ;  tárolása
        jsr chrgot ; az utolsó BASIC jel
        cmp #','   ;  vessző kell, hogy legyen
        beq tov2
        rts        ; ha nemm --> vége
tov2    jsr chrget ; következő BASIC jel
        jsr frmevl ; formula kiértékelése
        jsr facint ; egész számmá konvertálás
        ldx $65    ; az alsó byte
        stx hossz  ;  tárolása
        jsr chrgot ; az utolsó BASIC jel
        cmp #','   ;  vessző kell, hogy legyen
        beq tov3
        rts        ; ha nem --> vége
;
tov3    jsr chrget ; következő BASIC jel
        jsr adress ; változó címe
        ldy #$1    ;  ($47) mutat a sztring leíróra
        lda ($47),y ; sztirng mutató alsó byte
        sta mutato ;  tárolása
        iny
        lda ($47),y ; sztirng mutató felső byte
        sta mutato+1 ; tárolása
        lda #$0
        sta kp     ; a mező elejének beállítása
        jsr copy   ; a sztring pufferbe másolása
        jsr main   ; szerkeszto rutin
        jmp recopy ; puffer visszamásolaá sa
;
; a kurzor mezo elejére állítása
gohome ldx xkoord ; a plot KERNAL paramétereinek
        ldy ykoord ;  beállítása
        clc        ; C=0: kurzor állítás
        jsr plot
        rts
;
; kiirja a puffert a mezőbe
write   ldx #$00   ; ciklus eleje
cikl1   lda puffer,x ; A-ban a kiirandó karakter
        jsr chrout ; a karakter kiírása
        inx        ; ciklusváltozó növelése
        cpx hossz  ; vége-e a ciklusnak
        bne cikl1  ; nem --> folytatás
        rts        ; igen --> vége
;
; x -szel eltolva irja ki
gotox   txa        ; X átmásolása
        clc
        adc ykoord ; y-nal való megnövelése
```

```
       tay         ; y-ban való elmentése
       ldx xkoord  ; x beállítása
       clc         ; C=0: kurzor állítás
       jsr plot
       rts
;
; A szerkesztő ciklus
;  Visszatér a mező elejére. kiírja a puffert,
;  majd kp-vel eltolva bekapcslja a kurzort,
;  s beolvas egyetlen billentyűt. A kurzort kikapcsolja s
;  értelmezi a billentyűt.
;
main   jsr gohome ; mező elejére lépés
       jsr write  ; puffer tartalmának kiírása
       ldx kp     ; X-ben a kurzor relativ helye
       jsr gotox  ; kurzor mozgatása
       jsr kbe    ;  és bekapcsolása
cbill  jsr getin  ; van-e lenyomott billentyű?
       beq cbill  ; nincs --> vissza
       sta billen ; tárolása
       jsr kki    ; kurzor kikapcsolása
; Speciális billentyűk keresése
       ldx #$00   ; ciklus eleje
keres  lda spec,x ; következő speciális billentyű
       cmp billen ; azonos-e a tárolttal?
       beq megvan ; igen --> végrehajtás
       inx        ; ciklusváltozó növelése
       cpx #darab ; ciklus vége-e?
       bne keres  ; nem --> folytatás
; További vezérlő karakterek törlése
       lda billen ; a billentyű
       and #%01100000 ; vezérlő karakterek kiszűrése
       beq main   ; ha az volt --> újból a ciklus
       ldx kp     ; ha igen, mező végén
       cpx hossz  ;  vagyunk-e?
       beq main   ; igen --> nem írható be
       lda billen ; ha nem, akkor
       sta puffer,x ; beírjuk a pufferbe
       inc kp     ; s megnöveljük a kp-t
       jmp main   ; következő billentyű feldolgozására
; speciális billentyűk végrehajtása
megvan lda felso,x ; a visszatérési cím
       pha        ;   felső byte-ja a veremben
       lda also,x ; a visszatérési cím
       pha        ;  alsó byte-ja a veremben
       rts        ; vezérlésátadás
;
; <clr>
clear  lda #$20   ; törlő karakter
       ldx #$00   ; ciklus eleje
       stx kp     ; kp a mező elejére
cclear sta puffer,x ; puffer következő karakterének törlése
       inx        ; ciklusváltozó növelése
       cpx hossz  ; ciklus vége?
       bne cclear ; nem --> folytatás
       jmp main   ; vissza
;
```

```
; <home>
home    lda #$00
        sta kp      ; kp nullázása
        jmp main    ; vissza
;
; <jobbra>
jobbra ldx hossz  ; hossz és
        cpx kp      ; kp összehasonlítása
        bne job1    ; ha nem egyenlő --> kp megnövelhető
        jmp main    ; ha kp=hossz --> vissza
        job1 inc kp ; kp növelése
        jmp main    ; vissza
;
; <balra>
balra  ldx kp
        cpx #$00    ; kp=0?
        bne ball    ; ha nem --> kp csökkenthető
        jmp main    ; ha igen --> vissza
        ball dec kp ; kp csökkentése
        jmp main    ; vissza
;
; <del>
del     ldx kp      ; kp és
        cpx hossz   ;  hossz összehasonlítása
        bne dell    ; ha eltér --> ugrás
        dec kp      ; kp=hossz --> kp csökkentése
        ldy kp
        jmp del3    ; a puffer végére szóköz írása
dell    cpx #$00    ; X=0?
        bne del4    ; X<>0 --> del4
        jmp main    ; vissza
del4    dec kp      ; ciklus eleje
        ldy kp      ;
cdel    lda puffer,x ; egy karakter
        sta puffer,y ;  előre másolása
        inx         ; ciklusváltozók csökkentése
        iny
        cpx hossz   ; ciklus vége-e?
        bne cdel    ; nem --> folytatás
del3    lda #$20    ; puffer végére szóköz
        sta puffer,y ; beírása
vege1   jmp main    ; vissza
;
; <f7>
f7      ldx kp      ; kp és
        cpx hossz   ;  hossz összehasonlítása
        beq vege1   ; kp=hossz --> nem lehet törölni
        ldy kp
        inx         ; X=kp+1
        cpx hossz   ; X=hossz?
        beq del3    ; igen --> puffer végére szóköz
        bne cdel    ; nem --> elöre tolás
;
; <ins>
ins     ldx hossz   ; hossz és
        cpx kp      ;  kp összehasonlítása
        beq vege1   ; hossz=kp --> nem lehet beszúrni
```

```
       dex         ; kezdőértékek beállítása
       ldy hossz
       dey
       cpy kp      ; utolsó karakter-e?
       beq vege2   ; igen --> szóköz beírása
cins   dey         ; ciklusváltozó beállítása
       lda puffer,y ; egy karakter másolása
       sta puffer,x
       dex         ; X csökkentése
       cpx kp      ; elértük-e a kurzort?
       bne cins    ; nem --> folytatás
vege2  lda #$20    ; kurzor helyére
       sta puffer,y ;szóköz beírása
       jmp main
; <return>
return lda #$3     ; visszatérési kód
       bne vege3
fel    lda #$1     ; visszatérési kód
       bne vege3
le     lda #$2     ; visszatérési kód
       bne vege3
f3     lda #$4     ; visszatérési kód
vege3  sta error   ; tárolása
       rts
;
; <f1>
f1     jsr copy    ; a sztring átmásolása a pufferbe
       jmp main
;
copy   ldy #$0     ; ciklus eleje
ccopy  lda (mutato),y; szting karaktere A-ban
       sta puffer,y  ;  tárolása a pufferben
       iny         ; ciklusváltozó növelése
       cpy hossz   ; vége?
       bne ccopy   ; nem --> folytatás
       rts         ; igen --> vége
;
kbe    sei         ; kurzor bekapcsolása
       lda #$00
       sta $cc
       cli
       rts
;
kki    sei         ; kurzor kikapcsolása
       ldy $d3
       lsr $cf
       bcc ktov1
       ldx $0287
       lda ($d1),y
       eor #$80
       jsr $ea1c
ktov1  lda #$ff
       sta $cc
       cli
       rts
; puffer másolása a sztingbe
recopy ldy #$0     ; ciklus eleje
```

```
crcopy lda puffer,y ; puffer karaktere A-ban
       sta (mutato),y ; átíras a sztringbe
       iny            ; ciklusváltozó növelése
       cpy hossz      ; vége?
       bne crcopy     ; nem --> folytatás
       rts            ; igen --> vissza
;
; Speciális billentyűk száma
darab = 12
; Speciális billentyűk ASCII kódjai
spec  .byte 19 ; <home>
.byte 147  ; <clr>
.byte 29   ; <jobbra>
.byte 157  ; <balra>
.byte 133  ; <f1>
.byte 145  ; <fel>
.byte 17   ; <le>
.byte 13   ; <ret>
.byte 134  ; <f3>
.byte 148  ; <inst>
.byte 20   ; <del>
.byte 136  ; <f7>
; rutinok alsó címe - 1
also .byte <home-1, <clear-1, <jobbra-1
     .byte <balra-1, <f1-1, <fel-1, <le-1, <return-1
     .byte <f3-1, <ins-1, <del-1, <f7-1
; rutinok felső cime
felso .byte >home, >clear, >jobbra
      .byte >balra, >f1, >fel, >le, >return
      .byte >f3, >ins, >del, >f7
; Puffer a mező karaktereinek
puffer * = * +256
.end
```

A gépi kódú programrész használatára az 5.5.2 részben adott program egy lényegesen javított változatát mutatjuk be. A BASIC program végén adjuk meg a fenti rutin töltő programját is. Az természetesen más programokban is felhasználható.

A személyi nyilvántartó program használata elött (de csak egyszer) az alábbi generáló programot kell lefuttatni:

```
100 REM ************
110 REM * SZEMELYI.GEN *
120 REM ************
130 :
140 REM ADATBAZIS GENERALASA
150 OPEN 15,8,15
160 PRINT#15,"S0:ADATOK"
170 OPEN 2,8,2,"0:ADATOK,L,"+CHR$(150)
180 PRINT#15,"P"+CHR$(2)+CHR$(1)+CHR$(0)+CHR$(1)
190 :
200 N=0: PRINT#2,N
210 CLOSE 2: CLOSE 15
```

A személyi nyilvántartó program listáját az alábbiakban közöljük. Vigyázat: a 'print' utasításokban szereplő, s nem értelmes szöveget alkotó karakterek jelentése a következő:

A	<C=-L.Gry> Világosszürke
Á	<C=-Pur> Lila
E	<CTRL-Wht> Fehér
S	<CLR>
q	<CRSR-LE>
R	<CTRL-RVSON>
r	<CTRL-RVSOFF>
Q	<CRSR-FEL>
[	<CRSR-JOBBRA>

Az alábbi karakterek nagy betűk/grafikus jelek karakterkészlet használata esetén grafikus jelekként jelennek meg:

U	<SHIFT-U>
C	<SHIFT-C>
I	<SHIFT-I>
J	<SHIFT-J>
K	<SHIFT-K>
B	<SHIFT-B>
M	<C=-Q>
L	<C=-W>

Az érintett karakterek helyett a megfelelő vezérlő billentyűket kell beírni!

```
1000 :
1010 REM ********************
1020 REM * SZEMELYI ADATBAZIS *
1030 REM ********************
1040 :
1050 REM N$  : NEV
1060 REM S$  : SZULETESI HELY
1070 REM SI  : SZULETES EVE
1080 REM N$  : FERFI/NO
1090 :
1100 GOSUB 1410
1110 :
1120 REM MENU KIIRASA
1130 REM ============
1140 PRINT "SQQQ      AUCCCCCCCCCCCCCCCCCCCCCCCCCCCCI"
1150 PRINT "      B  ÁSZEMELYI NYILVANTARTASA  B"
1160 PRINT "      MCCCCCCCCCCCCCCCCCCCCCCCCCCCCCL"
1170 PRINT "      B     REKORDSZAM :    :     B"
1180 PRINT "Q]]]]]]]]]]]]]]]]]]]]]]]]]]]É";RIGHT$("  "+STR$(N),3);"A"
1190 PRINT "      MCCCCCCCCCCCCCCCCCCCCCCCCCCCCCL"
1200 PRINT "      B                            B"
1210 PRINT "      B UJ EMBER FELVITELE.....1 B"
1220 PRINT "      B LISTAZAS..............2 B"
1230 PRINT "      B MODOSITAS.............3 B"
1240 :
```

```
1250 PRINT "      B TORLES.................4 B"
1260 PRINT "      B VEGE...................5 B"
1270 PRINT "      B                          B"
1280 PRINT "      JCCCCCCCCCCCCCCCCCCCCCCCCCK"
1290 M$="5"
1300 PRINT "      B MELYIKET VALASZTJA?   : :B"
1310 PRINT "      JCCCCCCCCCCCCCCCCCCCCCCCCCKE"
1320 SYS 49152,31,16,1,M$
1330 IF (M$<"1") OR (M$>"5") THEN GOTO 1290
1340 M = VAL(M$): PRINT"A";
1350 :
1360 ON M GOSUB 1550,1670,1880,2380,2010
1370 GOTO 1140
1380 :
1390 REM ADATBAZIS MEGNYITASA
1400 REM ********************
1410 :
1420 POKE 53281,0
1430 POKE 53280,0
1440 SP$="                                                                "
1450 LE$="QQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQ"
1460 JO$="]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]"
1470 PRINT"Sá"
1480 V$=",": NMAX=254
1490 OPEN 15,8,15: GOSUB 3070
1500 OPEN 2,8,2,"0:ADATOK"
1510 PRINT#15,"P"+CHR$(2)+CHR$(1)+CHR$(0)+CHR$(1)
1520 INPUT#2,N
1530 RETURN
1540 :
1550 REM UJ EMBER FELVITELE
1560 REM ******************
1570 IF N=NMAX THEN RETURN
1580 GOSUB 2820: I%=N+1
1590 PRINT "SQQQQQQQQQQQQQQQQQQ"
1600 PRINT"           UCCCCCCCCCCCCCCCI"
1610 PRINT"           B   EFELVITELA   B"
1620 PRINT"           B  <F3> = VEGE B"
1630 PRINT"           JCCCCCCCCCCCCCCCK"
1640 PRINT"É";: X=9: Y=15: GOSUB 3140: PRINT RIGHT$("   "+STR$(I%),3);"A"
1650 GOSUB 3010: GOSUB 2150
1660 GOSUB 2740: N=N+1: RETURN
1670 :
1680 REM LISTAZAS
1690 REM ********
1700 IF N=0 THEN RETURN
1710 PRINT"S": X=9: Y=16: GOSUB 3140: PRINT"ÁLISTAZASA": GOSUB 3770
1720 GOSUB 2820
1730 IF I%<1 THEN I%=1
1740 IF I%>N THEN I%=N
1750 GOSUB 2660: GOSUB 2070
1760 PRINT "SQQQQQQQQQQQQQQQQQQ"
1770 PRINT"        UCCCCCCCCCCCCCCCCCCCCCCCCCI"
1780 PRINT"        B R+r ELORE    R-r HATRA     B"
1790 PRINT"        B RHr MASOLAT  RRETURNr VEGE]B"
1800 PRINT"        JCCCCCCCCCCCCCCCCCCCCCCCCCK"
1810 GET A$: IF A$="" THEN 1810
```

```
1820 IF A$="+" THEN I%=I%+1 : GOTO 1730
1830 IF A$="-" THEN I%=I%-1 : GOTO 1730
1840 IF A$="H" THEN GOTO 2560
1850 IF A$<>CHR$(13) THEN GOTO 1810
1860 RETURN
1870 :
1880 REM ADATOK MODOSITASA
1890 REM ******************
1900 PRINT"S": X=9: Y=15: GOSUB 3140: PRINT"ÁMODOSITASA": GOSUB 3770
1910 GOSUB 2660: GOSUB 2820: GOSUB 2070
1920 PRINT "SQQQQQQQQQQQQQQQQQ"
1930 PRINT"            UCCCCCCCCCCCCCCI"
1940 PRINT"            B   EMODOSITASA   B"
1950 PRINT"            B  <F3> = VEGE  B"
1960 PRINT"            JCCCCCCCCCCCCCCCK"
1970 GOSUB 2150: GOSUB 2740: RETURN
1980 :
1990 REM ADATBAZIS LEZARASA
2000 REM ******************
2010 PRINT#15,"P"+CHR$(2)+CHR$(1)+CHR$(0)+CHR$(1)
2020 PRINT#2,N
2030 CLOSE2: CLOSE15: END
2040 :
2050 REM KIIRO SZUBRUTIN
2060 REM AAAAAAAAAAAAAAA
2070 PRINT"É";: X=9: Y=15: GOSUB 3140: PRINT RIGHT$("   "+STR$(I%),3)
2080 X=11: Y=8: GOSUB 3140: PRINT N$
2090 X=13: Y=19: GOSUB 3140: PRINT S$
2100 X=14: Y=19: GOSUB 3140
2110 SI$=MID$(STR$(SI)+"    ",2,4): PRINT SI$
2120 X=16: Y=13: GOSUB 3140: PRINT NE$
2130 PRINT"A": RETURN
2140 :
2150 REM BEOLVASO SZUBRUTIN
2160 REM ******************
2170 I=1: PRINT"É"
2180 ON I GOSUB 2260,2270,2280,2320
2190 J=PEEK(49155): IF J=4 THEN RETURN
2200 IF J=1 THEN I=I-1
2210 IF J>1 THEN I=I+1
2220 IF I>4 THEN I=1
2230 IF I<1 THEN I=4
2240 GOTO 2180
2250 :
2260 SYS 49152,8,11,29,N$: RETURN
2270 SYS 49152,19,13,18,S$: RETURN
2280 SYS 49152,19,14,4,SI$
2290 SI=VAL(SI$)
2300 IF (SI$<"1800") OR (SI$>"2000") THEN GOTO 2280
2310 RETURN
2320 SYS 49152,13,16,1,NE$
2330 IF NOT ((NE$="N") OR (NE$="F")) THEN GOTO 2320
2340 RETURN
2350 :
2360 REM TORLES
2370 REM ******
2380 PRINT"S": X=9: Y=16: GOSUB 3140: PRINT" ÁTORLESA": GOSUB 3770
```

```
2390 J%=I%: GOSUB 2660: GOSUB 2840: GOSUB 2070
2400 PRINT "SQQQQQQQQQQQQQQQQQQ"
2410 PRINT"        UCCCCCCCCCCCCCCCCCCCCCCCCI"
2420 PRINT"        B  TOROLJEM (I/N)? E: :A   B"
2430 PRINT"        JCCCCCCCCCCCCCCCCCCCCCCCCK"
2440 A$="N"+"": PRINT"É"
2450 SYS 49152,28,20,1,A$: PRINT"A"
2460 IF (A$="I") OR (A$="N") THEN GOTO 2480
2470 GOTO 2450
2480 IF A$<>"I" THEN RETURN
2490 I%=N: GOSUB 2660
2500 I%=J%: GOSUB 2740
2510 N=N-1
2520 RETURN
2530 :
2540 REM NYOMTATAS
2550 REM *********
2560 OPEN 4,4
2570 PRINT#4,"REKORDSZAM=";I%: PRINT#4
2580 PRINT#4,"NEV.............. ";N$
2590 PRINT#4,"SZULETESI HELY... ";S$
2600 PRINT#4,"SZULETES EVE.....";SI
2610 PRINT#4,"NEME............. ";
2620 IF NE$="N" THEN PRINT#4,"NO"
2630 IF NE$="F" THEN PRINT#4,"FERFI"
2640 PRINT#4: CLOSE 4
2650 :
2660 REM EGY REKORD BEOLVASASA
2670 REM **********************
2680 PRINT#15,"P"+CHR$(2)+CHR$(I%+1)+CHR$(0)+CHR$(1)
2690 INPUT#2,N$
2700 PRINT#15,"P"+CHR$(2)+CHR$(I%+1)+CHR$(0)+CHR$(31)
2710 INPUT#2,SI,NE$,S$
2720 RETURN
2730 :
2740 REM EGY REKORD KIIRASA
2750 REM ********************
2760 PRINT#15,"P"+CHR$(2)+CHR$(I%+1)+CHR$(0)+CHR$(1)
2770 PRINT#2,N$
2780 PRINT#15,"P"+CHR$(2)+CHR$(I%+1)+CHR$(0)+CHR$(31)
2790 PRINT#2,SI;V$;NE$;V$;S$
2800 RETURN
2810 :
2820 REM FIX ADATOK KIIRASA
2830 REM ******************
2840 PRINT"SQQQQQA"
2850 PRINT"UCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCI";
2860 PRINT"B            ÁSZEMELYI ADATOKA            B";
2870 PRINT"MCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCL";
2880 PRINT"B  REKORDSZAM :   :                      B";
2890 PRINT"MCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCL";
2900 PRINT"B  NEV E:                               :A B";
2910 PRINT"B                                        B";
2920 PRINT"B  SZULETESI HELY E:                    :A B";
2930 PRINT"B  SZULETES EVE   E:    :A                B";
2940 PRINT"B                                        B";
2950 PRINT"B  NO/FERFI E: :A                        B";
```

```
2960 PRINT"JCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCK";
2970 RETURN
2980 :
2990 REM URES ADATOK
3000 REM ***********
3010 N$=RIGHT$(SP$,29)
3020 S$=RIGHT$(SP$,18)
3030 SI$=RIGHT$(SP$,4)
3040 NE$="N"+""
3050 RETURN
3060 :
3070 RESTORE
3080 CIM = 49152 : S=0
3090 READ A:  IF A=-1 THEN GOTO 3120
3100 POKE CIM,A: CIM = CIM +1 : S=S+A
3110 GOTO 3090
3120 IF S=70084 THEN RETURN
3130 PRINT"S ROSSZAK A DATA-BELI ADATOK!": CLOSE 15: END
3140 PRINT"S";RIGHT$(LE$,X);RIGHT$(JO$,Y);: RETURN
3150 :
3160 DATA 76,7,192,0,0,0,0,32
3170 DATA 115,0,32,158,173,32,170,177
3180 DATA 166,101,142,5,192,32,121,0
3190 DATA 201,44,240,1,96,32,115,0
3200 DATA 32,158,173,32,170,177,164,101
3210 DATA 140,4,192,32,121,0,201,44
3220 DATA 240,1,96,32,115,0,32,158
3230 DATA 173,32,170,177,166,101,134,254
3240 DATA 32,121,0,201,44,240,1,96
3250 DATA 32,115,0,32,139,176,160,1
3260 DATA 177,71,133,252,200,177,71,133
3270 DATA 253,169,0,133,255,32,127,193
3280 DATA 32,141,192,76,170,193,174,4
3290 DATA 192,172,5,192,24,32,240,255
3300 DATA 96,162,0,189,219,193,32,210
3310 DATA 255,232,228,254,208,245,96,138
3320 DATA 24,109,5,192,168,174,4,192
3330 DATA 24,32,240,255,96,32,102,192
3340 DATA 32,113,192,166,255,32,127,192
3350 DATA 32,140,193,32,228,255,240,251
3360 DATA 141,6,192,32,147,193,162,0
3370 DATA 189,183,193,205,6,192,240,29
3380 DATA 232,224,12,208,243,173,6,192
3390 DATA 41,96,240,209,166,255,228,254
3400 DATA 240,203,173,6,192,157,219,193
3410 DATA 230,255,76,141,192,189,207,193
3420 DATA 72,189,195,193,72,96,169,32
3430 DATA 162,0,134,255,157,219,193,232
3440 DATA 228,254,208,248,76,141,192,169
3450 DATA 0,133,255,76,141,192,166,254
3460 DATA 228,255,208,3,76,141,192,230
3470 DATA 255,76,141,192,166,255,224,0
3480 DATA 208,3,76,141,192,198,255,76
3490 DATA 141,192,166,255,228,254,208,7
3500 DATA 198,255,164,255,76,46,193,224
3510 DATA 0,208,3,76,141,192,198,255
3520 DATA 164,255,189,219,193,153,219,193
```

```
3530 DATA 232,200,228,254,208,244,169,32
3540 DATA 153,219,193,76,141,192,166,255
3550 DATA 228,254,240,247,164,255,232,228
3560 DATA 254,240,235,208,221,166,254,228
3570 DATA 255,240,232,202,164,254,136,196
3580 DATA 255,240,12,136,185,219,193,157
3590 DATA 219,193,202,228,255,208,244,169
3600 DATA 32,153,219,193,76,141,192,169
3610 DATA 3,208,10,169,1,208,6,169
3620 DATA 2,208,2,169,4,141,3,192
3630 DATA 96,32,127,193,76,141,192,160
3640 DATA 0,177,252,153,219,193,200,196
3650 DATA 254,208,246,96,120,169,0,133
3660 DATA 204,88,96,120,164,211,70,207
3670 DATA 144,10,174,135,2,177,209,73
3680 DATA 128,32,28,234,169,255,133,204
3690 DATA 88,96,160,0,185,219,193,145
3700 DATA 252,200,196,254,208,246,96,19
3710 DATA 147,29,157,133,145,17,13,134
3720 DATA 148,20,136,230,213,237,251,120
3730 DATA 106,110,102,114,68,9,53,192
3740 DATA 192,192,192,193,193,193,193,193
3750 DATA 193,193,193,81,81,-1
3760 :
3770 REM REKORDSZAM BEKERES
3780 REM ********************
3790 PRINT "SQQQQQQQ          UCCCCCCCCCCCCCCCCCI"
3800 PRINT "          B]]]]]]]]]]]]]]]]]]B"
3810 PRINT "          KCCCCCCCCCCCCCCCCCC³"
3820 PRINT "          B REKORDSZAM :    : B"
3830 PRINT "          JCCCCCCCCCCCCCCCCCCK"
3840 I$="   "+"": PRINT"É":
3850 SYS 49152,24,11,3,I$
3860 I%=VAL(I$)
3870 IF I%<1 OR I%>N THEN I$=RIGHT$("  "+STR$(N),3): GOTO 3850
3880 PRINT"A": RETURN
```

10. Az Easy Script használata

Az Easy Script szövegszerkesztő talán a legsikeresebb Commodore 64-es alkalmazói program, amelyik tökéletesen illeszkedik a gép adottságaihoz, s egyben komoly szövegszerkesztést is lehetővé tesz. A Commodre 64-es könyv eddigi kiadásai is ennek a programnak a magyarított változatával készültek. A program hardver igénye:

- Commodore-64 központi egység;
- VC-1541-es lemezegység;
- Easy Script cartridge, vagy programlemez
- a program által kezelt nyomtató.

A program eredetileg cartridge-en került forgalmazásra, de vannak lemezes változatai is. Elvileg csak kazettával is lehetne dolgozni, de ez gyakorlatilag rendkívül lassú.

A betöltés (vagy cartridge esetén a bekapcsolás) után bejelentkezik a rendszer. Megtudjuk, hogy a **Precision Software LTD**. készítette, 1982-ben. Rögtön ezután kell megadni a következő értékeket:

ENTER TEXT WIDTH: a szövegfile **szélessége**. Az értéke a 40-240 intervallumba kell, hogy essen. Meghatározza a szövegfile tárolását a memóriában. **Semmi köze ahhoz, hogy a szöveg kiíráskor hogyan jelenik meg a nyomtatón**. Normál esetben 40-et lehet választani. Nagyobb táblázatok beírásakor célszerű az értékét a táblázat sorszélességének választani. Alapérték: 40

(D)ISK OR (T)APE? Lehetőség van arra, hogy megadjuk, a szövegfile-okat **lemezre** vagy **kazettára** mentsük el. A magam részéről a szalagot semmiképp sem javaslom!

PRINTER TYPE (0-4): a használt nyomtató típusa. Alapérték: 0 (CBM kompatibilis). Ha nem ezt választjuk, akkor a program megkérdezi, hogy a nyomtatóval milyen a kapcsolat:
(R)S232 (C)ENTRONICS OR (S)ERIAL ? Válaszként a <r>, <c> vagy <s> billentyűk valamelyikét kell megnyomni. Ha az RS232 -es csatornát adjuk meg, akkor még megkérdezi a parancs- és kontrolbyte értékét is. Ezeket a nyomtató ismeretében kell megadni.

Az alapparaméterek bármikor újra definiálhatók a <STOP-RESTORE> billentyűzés után! A memóriában tárolt szöveg ilyenkor **nem** vész el!

Az Easy Script bizonyos műveletek befejezését és a hibákat "bípeléssel" jelzi. Ezt az <F1>* billentyűzéssel ki/be kapcsolhatjuk.

Az írás, a keret és a háttér színeit a <CTRL−1>,<CTRL−2> illetve a <CTRL−3> billentyű megnyomásával az értékek bármelyikét beállíthatjuk.

Mind a hangjelzés, mind a színek állítása csak az ún. szerkesztő üzemmódban kerülhet sor.

A rendszer általános jellemzői

A szövegszerkesztő hagyományos elemei a népszerű EASY SCRIPT szintaktikáját és rendszerépítését követik. Ennek megfelelően a kiírandó szöveget ún. **szövegfile**-ba kell írni, ami a szöveg kiírását vezérlő karaktersorozatokat, illetve szövegközi vezérlő jeleket is tartalmazhat. A szöveg beírása, javítása közben **nem** látjuk a képernyőn, hogy a szöveg végül is milyen alakban fog kinyomtatódni. Ennek oka a kevés memória.

A rendszer három üzemmódban dolgozik:

- Szerkesztő (alap) módban. Ekkor nyílik lehetőség a szövegfile beírására, illetve módosítására. A beírás lehetőségei gyakorlatilag megegyeznek a C-64 teljes képernyős szerkesztőjének lehetőségeivel.
- Parancs módban. Az <f1> billentyű lenyomásával ún. **parancs módba** lépünk be. További egykarakteres alparancsokat használhatunk. A parancsok lehetővé teszik a szöveg, vagy egy részének elmentését, a már elmentett szövegek betöltését, vagy beszúrását, szövegrészek áthelyezését. Speciális parancsok szolgálnak a szöveg kiírására. A használható parancsokat egy mellékletben külön is összefoglaltuk.
- Lemez módban. Az <f4> billentyű lenyomásával **lemez módba** kerülünk. A VC-1541-es DOS parancsokat használhatjuk. Törlő jellegű parancs esetén a rendszer visszakérdez, hogy biztosak vagyunk-e benne. A $ parancs a képernyőre listázza a katalógust, a +$ parancs pedig a katalógust még be is tölti a szövegfile-ba.

Körlevelek készítése

A szövegfile-ban az <f1> **B** parancs kiadásával helyezhetünk el **üres blokkokat**. Abban az esetben, ha ezeket a blokkokat fel akarjuk nyomtatáskor információval tölteni, akkor a nyomtatás alparancsai közt az F-es (fill file) is meg kell adni.

Lehetőség van a nyomtatástól függetlenül feltölteni a blokkokat, erre az <f1> **V** parancs szolgál. Az esetleg már feltöltött blokkokat a kurzor helyétől kiüríti az <f1> SHIFT-**V** parancs.

Szövegek tárolása

Abban az esetben ha egy szövegfile-t, vagy annak egy részét egy már létező file-ba akarunk elmenteni, akkor a nevet megelőzően be kell írnunk a '@0:' karaktereket is!

Szerkesztő (alap) mód

Szerkesztő módban nyílik lehetőség a szöveg file megírására. A képernyő jól elkülönülten két részre oszlik: az első sorba közvetlenül nem lehet írni, ez az ún. **parancs sor**. Alatta a szöveg 24 sora látszik. Normál körülmények között az első sorban a

EDIT :MODE L:232 C:039

alakú üzenet látszik. L a kurzor sorának, C pedig oszlopának számát mutatja. Ha a képernyő méretet 40-nél nagyobbra választottuk, akkor a képernyő mintegy ablak csúszik a szövegen, s mindig csak valamelyik 40 oszlopa látszik. Ha valamilyen parancsot hajtottunk végre, akkor az első billentyű megnyomásáig a parancs végrehajtására utaló üzenet jeleník meg. Kivételt a lemezegységről olvasott rendszerüzenetek jelentenek. Ezek angol nyelvűek.

A kurzor mozgatása

A szövegfile-ba a következő karakter a kurzor helyénél kerül be. A kurzort tetszés szerint pozícionálhatjuk a kurzor vezérlő billentyűk, a <HOME> és a <CLR> billentyűk segítségével. A <HOME> hatására a kurzor a bal felső sarokba kerül. A <CLR> a kurzort a file elejére pozicionálja, ami egyben azt is jelenti, hogy a szövegfile első 24 sora jelenik meg a képernyőn.

A <←>, visszanyíl billentyű a kurzortól visszafelé az első sor vagy szóvégre pozícionálja a kurzort. A <SHIFT-RETURN> a következő sor elejére állítja be a kurzort.

Paragrafus vége

A szöveget folyamatosan kell begépelni a szövegszerkesztőbe, még akkor is, ha a képernyő szélén kettévágódik egy szó. A rendszer automatikusan gondoskodik arról, hogyha egy szó nem fér ki, akkor azt kiíráskor már új sorba kezdje. Ez alól az általános szabály alól csak egy kivétel van: a bekezdés vagy paragrafus vége. Ezt külön kell jelölni a <RETURN> beírásával. Ennek hatására a szövegbe egy inverz alakú kisebb jel kerül, jelezve a paragrafus végét. A kurzor a következő sor elejére kerül.

Beírási módok

A szövegek beírását többféle módon szabályozhatjuk. Normál esetben az újjonan beírt karakter a régit felülírja. Lehetőségünk van azonban ún. **beszúrási** üzemmódban dolgozni. Ekkor a karakter beszúródik a kurzor pozíciójára, s a teljes paragrafus odébb csúszik. A beszúrási üzemmód be, illetve kikapcsolása az <f1>-**I** parancs kiadásával történik. Ha beszúrási üzemmódban vagyunk, akkor a parancs sorban egy I betű figyelmeztet erre.

Az <f5> billentyű flip-flop kapcsolóként működik. Első megnyomásakor a rendszer átáll a **nagybetűs** írásmódra. Ezt a képernyő parancs sorában megjelenő 'C' betű is jelzi. Második megnyomásakor visszatér normál üzemmódba. Nagybetűs írásmód esetén ha az <a> billentyűt megnyomjuk, akkor a képernyőre (és a szövegfile-ba) 'A' betű kerül. A <SHIFT-a> (azaz a 'nagy' A) megnyomására viszont normál 'a'. A számjegyek és az egyéb billentyűk szerepe nem változik.

Számtáblázatok szerkesztéséhez használhatjuk az ún. **decimális** módot. Ebben az esetben a beírt karakterek a kurzortól balra íródnak be egészen egy <,>, <.> vagy <szóköz> lenyomásáig. Így lehetőség van arra, hogy a tabulálással együtt a tizedesjel helyét állandóra lehessen venni. A decimális mód ki/be kapcsolását az <f6> billentyű megnyomásával végezhetjük el. A decimális mód érvényességére a parancs sorban levő D betű figyelmeztet.

Tabulálás

Bekezdésekhez, táblázatok szerkesztéséhez használhatjuk a tabulálást. Lehetőség nyílik mind vízszintes, mind függőleges tabulálásra. A tabulálási pozíciókat megfelelő parancsokkal lehet beállítani. A tabulálást magát az <f7> billentyű lenyomásával érhetjük el. Ha a tabuláláskor decimális üzemmódban voltunk, akkor a beírás automatikusan decimális üzemmódba kerül, egészen az első <;>, <,> vagy <szóköz> leütéséig.

Karakterek beszúrása/törlése

Az <INST> billentyűvel a kurzor pozícióra szúrhatunk be egy szóközt, illetve <DEL> billentyűvel a kurzor pozícióján levő karaktert törölhetjük. Ennek megfelelően mozog a teljes paragrafus is.

Parancs mód

A parancs mód biztosít lehetőséget a memóriában levő szöveggel való legkülönfélébb manipulációkra. Ezek közül is a leglényegesebbek a szövegfile tárolása, illetve visszatöltése, valamint kiíratása.

A parancsok általában egyetlen karakterből állnak, amelyeknek azután további

– ugyancsak egy karakteres – alparancsai lehetnek. Néhány kivételes esetben – pl. a file-neveknél – tetszőleges szöveget írhatunk be a parancsba. A parancsba való beíráshoz a billentyűzet speciális kibővítését nem használhatjuk, ezek a karakterek (pl. az A) közvetlenül nem írhatók be parancsba. Ezen az <f2> billentyű használata segít. Ennek hatására a parancssorba kerül a szövegfile-ban levő, első idézőjelek közti szöveg. Ha az <f2> billentyűt még egyszer megnyomjuk, akkor a rendszer megkeresi a második idézőjelek közti szövegrészt stb. Ilyen módon lehet pl. karaktersorozatok cseréjében ékezetes betűket is szerepeltetni. File névben nem célszerű ékezetes betűt használni, mert az a katalógus BASIC-ben történő kilistázásakor igen csúnya lesz!

Szöveg részének beállítása

A legtöbb művelet nem az egész szövegfile-on, hanem annak egy részén is elvégezhető. Ehhez azonban be kell állítani a szöveg részét. Ez a memória egy darabja lesz, ha átírjuk benne a szöveget, a rész tartalma ugyan változik, de a rész maga nem.

Állítsuk a kurzort a rész első karakterére, majd adjuk ki az <f1> <r> parancsot. Ezután a kurzort vigyük el a rész utolsó karakterére, s ezután nyomjuk meg a <RETURN> billentyűt. Ezzel a részt definiáltuk.

Cserélő karaktersorozatok definiálása

Lehetőség van karaktersorozatok megkeresésére, illetve cseréjére. Ehhez meg kell adni, hogy milyen karaktersorozatot, milyenre cseréljen. Ehhez adjuk ki az <f1> <s> parancsot. Először a cserélendő (keresendő) karaktersorozatot kell beírni, majd azt, amire cserélni akarjuk. A beírást a <RETURN> benyomásával kell befejezni. Ha ékezetes betűket, vagy speciális jeleket akarunk keresni/cserélni, akkor használjuk az <f2> gomb lehetőségét.
A karaktersorozatok definiálására még semmi sem történik!

Szöveg(rész) tárolása

A szövegfile-t az <f1> <f> parancs kiadásával irathatjuk ki lemezre. Ha a már beállított részt akarjuk lemezre menteni, akkor az <f1> <shift-f> parancsot adjuk ki. Mindkét esetben megkérdezi az elmentendő file nevét. Ha a lemezen már létezik ilyen file, akkor megkérdezi, hogy felülírja-e. Az <y> billentyű megnyomására felülírja, különben nem.

Szövegfile betöltése

A lemezen már tárolt szövegek az <f1> <L> parancs kiadásával tölthetők be a memóriába. A töltés a kurzor pozíciójától kezdődik. Ha beszúrási üzemmódban vagyunk, akkor a file tartalma beszúródik, ha nem, a memória tartalma felülíródik.

Szövegrész áthelyezése

A kurzort állítsuk arra a pozícióra, ahová a szövegrészt át akarjuk helyezni, majd adjuk ki az <f1> <x> parancsot. A szövegrész a megfelelő helyre kerül.

Szövegrész megismétlése

A kurzort állítsuk arra a helyre, ahová a szövegrészt meg akarjuk ismételni, majd adjuk ki az <f1> <a> parancsot. A szövegrész eredeti helyén változatlanul megmarad, s a kurzor helyére automatikusan beszúródik még egy példányban.

Karaktersorozat keresése

Először állítsuk be az <f1> <s> parancs segítségével a keresendő karaktersorozatot, majd adjuk ki az <f1> <h> parancsot. A 'Keresés:' megjelenésekor <m> -et kell válaszolni, ha a memóriában, <l>-t, ha összefűzött file-okban akarunk keresni. Memóriában való keresésnél a keresés a kurzor pozíciójától az első megtalálásig tart. Összefűzött file-ok esetében a keresés a megadott első file-tól kezdődik. Ha előzőleg a <c> billentyűt is megnyomtuk, akor a keresés folyamatosan halad előre.

Karaktersorozat cseréje

Először a cserélendő karaktersorozatokat kell a <f1> <s> paranccsal beállítani, majd kiadni a <f1> <@> parancsot. A 'Csere!:' megjelenésekor <m> -et kell válaszolni, ha a memóriában, <l>-t, ha összefűzött file-okban akarunk cserélni. A memóriában a csere a kurzor pozíciójától a file végéig tart, összefűzött file-ok esetén valamennyi file-ban megtörténik a csere, és a rendszer a módosított file-okat változatlan névvel másolja vissza.

Tabulálási pozíciók beállítása

A tabulálással kapcsolatban számos parancs van. A tabulálás beállítása előtt a kurzort a kívánt tabulálási pontra állítjuk, majd kiadjuk az <f1> <t> parancsot. A 'Tabulálás beállítása' felirat megjelenésekor <h>-t kell válaszolni ha vízszintes, <v>-t ha függőleges tabulálást állítunk be. Ha a <RETURN>-t nyomjuk meg, visszatérünk szerkesztő módba. Ha a tabulálás beállításakor decimális módban vagyunk, a szóbanforgó tabulálási pont decimális lesz. Ha tehát az <f7> megnyomásával rátabulálunk, automatikusan decimális módba kerülünk.

Tabulálási pontot hasonlóan lehet törölni, csak az <f1> <c> parancsot kell kiadni.

Egyszerre valamennyi vízszintes vagy függőleges tabulálási pontot törölhetjük az <f1> <z> parancs kiadásával, s ezt követően a <h> vagy <v> megnyomásával.

Sorok beszúrása/törlése

Parancs módban az <INST>, illetve <DEL> billentyűk segítségével egy sort szúrhatunk be, illetve törölhetünk. A parancs a <RETURN> megnyomásáig ismételhető.

Sorok törlése

Az <f1> <d> parancs kiadása után az aktuális kurzor helyétől tetszőleges részét törölhetjük a memóriának. Ehhez a kurzort a törlendő rész utolsó karakterére kell mozgatni, majd megnyomni a <RETURN> billentyűt.

Logikai törlés

Lehetőség van logikai egységek törlésére. Ehhez az <f1> <e> parancsot kell kiadni. A 'Törlés L' felirat megjelenése után több alparancs adható ki:

<a> a teljes memóriát törli;
<s> a kurzortól töröl az első .-ig;
<r> a kurzortól törli a teljes szöveget;
<p> a kurzortól töröl a paragrafus végéig.

A kurzor mozgatása

Lehetőség van a kurzor helyzetének gyors megváltoztatására is. Szerkesztő módban tehát nem kell soronként eljutni arra a helyre ahová akarunk:

<f1> <szóköz>	24 sor lefelé
<f1> <shift-szóköz>	24 sor felfelé
<f1> <kurzor le>	folyamatos mozgatás lefelé
<f1> <kurzor fel>	folyamatos mozgatás felfelé
<f1> <kurzor jobbra>	folyamatos mozgatás jobbra
<f1> <kurzor balra>	folyamatos mozgatás balra

A folyamatos mozgatás a <SHIFT> billentyű lenyomásával gyorsítható. A mozgatás a memória végéig, vagy a <STOP> billentyű lenyomásáig tart.

A memória adott sorára is ugorhatunk az <f1> <g> parancs kiadásával. Az 'Ugrás sor#' megjelenésekor be kell írni a kívánt sor számát. Használhatjuk az <e> betűt is, ennek hatására a kurzor az utolsó sorba kerül.

Körlevél készítése

A rendszer lehetőséget biztosít körlevelek készítésére. Ehhez a szövegfile-ba ún. blokkokat kell elhelyezni. Ezekbe töltődik be a címfile tartalma. A töltés, illetve ürítés ellenőrizhető.

Ha a szövegbe egy blokkot akarunk behelyezni, akkor az <f1> <b> parancsot kell kiadni. Ennek hatására két négyzetből álló téglalap kerül a szövegfile-ba. A két négyzet közé kerül majd a szöveg.

A blokkok feltöltését az <f1> <v> parancs kiadásával érhetjük el. Válaszként a címfile nevét kell megadni. Ha erre <RETURN>-nel válaszolunk, akkor a kurzor a következő blokkra ugrik. Így manuálisan is feltölthetők a blokkok. A már feltöltött blokkokat az <f1> <shift-v> parancs kiadásával érhetjük el. A blokkok feltöltése a file kiírásakor automatikusan megtörténik.

Tájékoztató

Az <f1> <u> parancs kiadásával a képernyőre kerül egy, a parancsok hatását összefoglaló táblázat. Alap módba a <STOP> megnyomásával térhetünk vissza.

Kiíratás

A legvégére hagytuk a szövegfile kiírásának módját, ez lévén a legbonyolultabb parancs. A parancsot az <f1> <o> billentyűzésével adhatjuk meg. Ezután több alparancs következhet, amelyek a kiírás módját határozzák meg:

X példányszám. A parancs után a rendszer megkérdezi a példányszámot. A beírást a <RETURN>-nel kell befejezni.

C folytonos kiíratás. Az alparancs hatására valamennyi oldal kiírásra kerül. Ha nem adjuk ki, egyszerre csak egy oldal kerül kiírásra.

L összefűzött file-ok. Ha nem adjuk ki ezt az alparancsot, akkor a memória tartalma kerül kiírásra. Ha megadjuk akkor a rendszer kéri majd a lánc első filejának a nevét.

F cím file használata. A rendszert utasítja, hogy a nyomtatás előtt az adott cím-file-ból töltse fel a blokkokat. Ha a feltöltés után még van szöveg a cím file-ban, akkor a szöveg – a blokkok új tartalmával – újból és újból kiíródik.

A képernyő jobb felső sarkában megjelenő betűk emlékeztetnek az alparancsok használatára.

A kiírás megkezdése az alábbi alparancsok kiadásával kezdhető meg:

P nyomtató. A szövegfile megszerkesztett tartalma a bejelentkező képernyőn kiválasztott típusú nyomtatón jelenik meg. Ha ez EPSON volt, akkor a rendszer az ott kiválasztott karakterkészletet használja. Összefűzött file-ok esetén az adatlemezen rajt kell lennie a megfelelő "kar.*" file-nak. Ha a memóriából nyomtatunk, akkor a rendszerlemezt elég behelyezni a lemezegységbe. Az MPS801/803-as nyomtató használata esetén erre nincs szükség! Ha nem adtuk ki a <c> alparancsot, akkor csak egy oldalnyi szöveg kerül kiírásra. A <p> megnyomásával a következő oldal is kiirathatő. <shift-p> hatására a kiírás folyamatos lesz.

V képernyő. A megszerkesztett formátumot megnézhetjük a képernyőn. A képernyő mint egy ablak mozgatható a szövegen. A képernyő mozgatására a <kurzor jobb>, <kurzor bal> billentyűket értelemszerűen használhatjuk. A <RETURN> lenyomása a képernyőt az első oszlopra mozgatja. Az <f7> 20, az <f5> 40 oszloppal mozgatja jobbra a képernyőt. A szövegben előre a <C=> gomb lenyomásával lehet mozogni. Ha a <szóköz> billentyűt lenyomjuk, a szöveg a következő lap elejéig görög. Ez a <szóköz> újbóli lenyomásával megállítható. Ha nem adtuk ki a <c> alparancsot, akkor a <c> megnyomásával kapjuk a következő oldalt. Ha a <p> billentyűt nyomjuk meg, akkor a következő oldal a nyomtatón jelenik meg. Ha a <shift-p> parancsot adjuk ki, akkor a nyomtatás folyamatos lesz.

S nyomtató. A megszerkesztett szöveg a lemezre kerül ASCII formában. A rendszer kéri a file nevét.

Vezérlő karakterek

A vezérlő karaktersorozatokat az <f3> billentyű megnyomásával kell bevezetni. Ennek hatására a képernyőn egy inverz csillag jelenik meg. A legközelebbi paragrafus vége jelig minden karaktert a vezérlő sorozat részének tekint a rendszer. A vezérlő sorozatban csak a szövegrészek tartalmazhatnak szóközt!
A továbbiakban x egy maximum 3 jegyű számot jelent, <szöveg> tetszőleges szöveg, amelyik nem tartalmaz kettőspontot és paragrafus vége jelet. A használható vezérlő sorozatok a következők:

pl x az egy lapra nyomtatható sorok fizikai száma
tl x az egy lapra nyomtatható sorok száma. A szövegszerkesztő összesen az itt megadott sort ír egy lapra. A lap többi sora üres marad. Ebből jönnek le a fejléc sorai, de nem számít bele a lábjegyzet sora
lm x bal margó beállítása.
rm x jobb margó beállítása.
ma x jobb margó visszatolása (egy sorra hat).
ra x jobb szélre tömörítés beállítása. Normál esetben, mivel a rendszer nem vágja ketté a szavakat, a kiírás jobb széle cakkos lesz. X = 1 esetben a szavakat a jobb szélre illeszti, s a bal oldal lesz cakkos.
ju x jobb oldali szélre igazítás. A fent jelzett cakkos szélt el lehet kerülni a szavak közé illesztett kellő számú szóközzel. Ezt az eljárást az x = 1 érték aktivizálja, x = 0 pedig hatástalanítja.
cn x középre állítás. Az egy sorba kerülő sort középre állítja. X = 1 aktivizálja ezt a módot, x = 0 kikapcsolja.
lf x 'line feed' küldése. x = 1 esetén a sor végén a nyomtatóra még egy CHR$(10) karakter is kiíródik. x = 0 ezt hatástalanítja.
ln x x darab üres sor. Ha az oldalra még egyetlen karakter sem került kiírásra hatástalan.
sp x soremelés nagysága. A rendszer minden soremelés után még x darab sort emel. x = 0 hatástalanítja.
fp x ha a lapon nincs már x üres sor, akkor a szöveg kiírása új oldalon kezdődik. Ha x = 0, akkor mindenképp új lapot kezd. Különben hatástalan.
hl x fejléc és lábjegyzet sor bal margója.
hr x fejléc és lábjegyzet sor jobb margója.
p# x lapszámozás kezdőértékének beállítása.
ps <szöveg> írás felfüggesztése. Ha a kiírás a nyomtatóra történik, akkor a <szöveg> az első sorba kiíródik. A kiíratás a <c> megnyomásával folytatható.
lk:<szöveg> összefűzött file jelzése. Ha a nyomtatásnál megadtuk az <l> parancsot, akkor a memóriába töltődik a <szöveg> nevű file, s vele folytatódik a feldolgozás. A 'lk:' vezérlő sorozatnak a szövegfile utolsó sorának kell lennie!
hd x:<szöveg>,<szöveg>,<szöveg> fejléc szövegei. x-1 a fejléc utáni üres sorok száma. x = 0 esetén nem leszfejlécsor.
ft x:<szöveg>,<szöveg>,<szöveg> lábjegyzetsor szövegei. A lábjegyzet alúról az x. sorba íródik. x = 0 esetén nincs lábjegyzetsor.

FÜGGELÉK

F.1 BASIC alapszavak

A függelék felsorolja a BASIC alapszavakat, a fenntartott változókat, illetve ezek rövidítését. A 'képernyő' címszó alatt a rövidítés használatakor a képernyőn látható alakot adjuk meg; feltéve, hogy a 'kis betűk/nagy betűk' karakterkészletet használjuk. A 'nagy betűk/grafikák' karakterkészlet használata esetén a siftelt (emelt) betűk helyén grafikus jelek láthatók. Például aB helyett ◫ jelenik meg.

Parancs	Rövidítés	Képernyő	Függvénytípus
ABS	A SHIFT B	aB	Numerikus
AND	A SHIFT N	aN	
ASC	A SHIFT S	aS	Numerikus
ATN	A SHIFT T	aT	Numerikus
CHR$	C SHIFT H	cH	Sztring
CLOSE	CL SHIFT O	clO	
CLR	C SHIFT L	cL	
CMD	C SHIFT M	cM	
CONT	C SHIFT O	cO	
COS	nincs	cos	Numerikus
DATA	D SHIFT A	dA	
DEF	D SHIFT E	dE	
DIM	D SHIFT I	dI	
END	E SHIFT N	eN	
EXP	E SHIFT X	eX	Numerikus
FN	nincs	fn	
FOR	F SHIFT O	fO	
FRE	F SHIFT R	fR	Numerikus
GET	G SHIFT E	gE	
GET#	nincs		
GOSUB	GO SHIFT S	goS	
GOTO	G SHIFT O	gO	
IF	nincs	if	
INPUT	nincs	input	
INPUT#	I SHIFT N	iN	
INT	nincs	int	Numerikus
LEFT$	LE SHIFT F	leF	Sztring
LEN	nincs	len	Numerikus
LET	L SHIFT E	lE	
LIST	L SHIFT I	lI	
LOAD	L SHIFT O	lO	
LOG	nincs	log	Numerikus

Parancs	Rövidítés	Képernyő	Függvénytípus
MID$	M SHIFT I	ml	Sztring
NEW	nincs	new	
NEXT	N SHIFT E	nE	
NOT	N SHIFT O	nO	
ON	nincs	on	
OPEN	O SHIFT P	oP	
OR	nincs	or	
PEEK	P SHIFT E	pE	Numerikus
POKE	P SHIFT O	pO	
POS	nincs	pos	Numerikus
PRINT	?	?	
READ	R SHIFT E	rE	
REM	nincs	rem	
RESTORE	RE SHIFT S	reS	
RETURN	RE SHIFT T	reT	
RIGHT$	R SHIFT I	rI	Sztring
RND	R SHIFT N	rN	Numerikus
RUN	R SHIFT U	rU	
SAVE	S SHIFT A	sA	
SGN	S SHIFT G	sG	Numerikus
SIN	S SHIFT I	sI	Numerikus
SPC	S SHIFT P	sP	Speciális
SQR	S SHIFT Q	sQ	Numerikus
STATUS	ST	st	Numerikus
STEP	ST SHIFT E	stE	
STOP	S SHIFT T	sT	
STR$	ST SHIFTR	stR	Sztring
SYS	S SHIFT Y	sY	
TAB	T SHIFT A	tA	Speciális
TAN	nincs	tan	Numerikus
THEN	T SHIFT H	tH	
TIME	TI	ti	Numerikus
TIME$	TI$	ti$	Sztring
TO	nincs	to	
USR	U SHIFT S	uS	Numerikus
VAL	V SHIFT A	vA	Numerikus
VERIFY	V SHIFT E	vE	
WAIT	W SHIFT A	wA	

F.2 BASIC hibaüzenetek

Ez a függelék a C-64 BASIC interpreter által küldött hibaüzeneteket tartalmazza, illetve az azokat **előidéző okokat** ismerteti. A hibaüzenet mindig egy **?** kiírásával kezdődik és az **ERROR** (hiba) szóval végződik. Amennyiben a hiba programfutás közben történt, annak a sornak a száma is kiíródik, ahol az interpreter a hibát észlelte. **Nem biztos**, hogy ténylegesen az a sor a hiba okozója.

BAD DATA

Egy megnyitott file-ból vagy DATA utasításból sztring típusú adat érkezett, a program viszont számadatot várt.

BAD SUBSCRIPT

A program egy tömb olyan elemére hivatkozott, amely túlmutat az egyáltalán lehetséges, vagy a DIM utasításban megadott határon.

BREAK IN...

A program végrehajtása a <STOP> gomb lenyomása, vagy egy STOP utasítás végrehajtása miatt félbeszakadt. Az üzenet tartalmazza annak a sornak a számát, ahol a program megállt.

CAN'T CONTINUE

A CONT paranccsal nem tudjuk folytatni a programot, mert
1/ a programot még nem indítottuk el a RUN paranccsal;
2/ hiba után akarjuk továbbinditani a programot;
3/ időközben átszerkesztettük a programot.

DEVICE NOT PRESENT

A megcímzett I/O eszköz nincs a rendszerben vagy nem üzemképes (pl. nincs bekapcsolva).

DIVISION BY ZERO

Nullával osztottunk.

EXTRA IGNORED

Túl sok adatot gépeltünk be az INPUT utasítás után, s csak az első néhány került elfogadásra.

FILE NOT FOUND

Az OPEN vagy LOAD utasításban megadott file nem szerepel a lemez katalógusában, vagy a szalagon való keresés közben az interpreter END-OF-TAPE (szalag vége) fejlécet (header-t) talált.

FILE NOT OPEN

A CLOSE, CMD, PRINT#, INPUT# vagy GET# által használni kívánt file-t még nem nyitottuk meg.

FILE OPEN

Egy olyan file megnyitására történt kísérlet, amelyet már egyszer megnyitottunk, de még nem zártunk le.

FORMULA TOO COMPLEX

A feldolgozás alatt levő kifejezést több részre kell bontani ahhoz, hogy a rendszer ki tudja értékelni. (Pl. a kifejezés túl sok zárójelet tartalmaz).

ILLEGAL DIRECT

Az utasítás parancs (direkt) módban nem használható.

ILLEGAL QUANTITY

A szám, amelyet egy függvény vagy utasítás argumentumaként használtunk, kívül esik a megengedett értékhatárokon.

LOAD

A szalagról vagy lemezről való töltés az adathordozó hibája miatt megszakadt. Elsősorban szalagról való töltés esetén fordulhat elő.

NEXT WITHOUT FOR

Next utasítást használtunk egy neki megfelelő, a végrehajtásban azt megelőző FOR utasítás nélkül. Okozhatják hibásan egymásba skatulyázott ciklusok; vagy az, ha a NEXT utáni változó név nem felel meg a FOR utasítás ciklusváltozójának (elírtuk).

NOT INPUT FILE

Egy outputként kijelölt file-t INPUT# vagy GET# utasításban akartunk használni.

NOT OUTPUT FILE

Egy inputként kijelölt file-ba akartunk PRINT# utasítással írni.

OUT OF DATA

A READ utasítás nem tudott a DATA-kból adatot olvasni, mert már az összeset felhasználtuk.

OUT OF MEMORY

Nincs elegendő RAM memória a program vagy a változók számára. Akkor is ezt a hibajelzést kapjuk, ha túl sok FOR ciklust skatulyáztunk egymásba, vagy a megengedettnél nagyobb mélységű a GOSUB-ok hívása.

OVERFLOW

A számítási műveletek eredménye nagyobb, mint a megengedett legnagyobb szám, amely 1,70141884E + 38.

REDIM'D ARRAY

Egy tömböt újból definiálni akartunk, holott csak egyszer lehet. Ha egy tömbváltozó a tömb dimenzionálását megelőzően előfordul a programban, akkor automatikus tömbdeklaráció hajtódik végre, így egy ezt követő DIM utasítás már hibát eredményez!

REDO FROM START

Az INPUT utasítás végrehajtása során számjegy helyett karakter adatot írtunk be. Írjuk újra be az adatokat, most már helyesen, és a program futása folytatódik.

RETURN WITHOUT GOSUB

RETURN utasítást hajtottunk végre azt megelőző GOSUB hívás nélkül.

STRING TOO LONG

Az eredményül kapott sztring több, mint 255 karaktert tartalmaz.

SYNTAX ERROR

Az utasítás nem felismerhető a C-64 BASIC számára: hiányzó zárójel, hibás kulcsszó stb.

TYPE MISMATCH

Szám helyett sztringet (vagy fordítva) használtunk.

UNDEF'D FUNCTION

Felhasználó által definiált függvényre hivatkoztunk, de azt még nem definiáltuk a DEF FN utasítás segítségével. Okozhatja egyszerű elgépelés.

UNDEF'D STATEMENT

Nem létező sorra történt hivatkozás GOTO,GOSUB,RUN,IF illetve ON utasításokból.

VERIFY

A szalagra vagy lemezre másolt program nem egyezik meg a memóriában levővel.

F.3 DOS üzenetek

A 20-nál kisebb hibakódú üzeneteket figyelmen kívül lehet hagyni. Ezek közül csak a 0 és az 1 kódszámnak van közvetlen jelentése.

0 OK

Minden rendben

1 FILES SCRATCHED

A SCRATCH (törlés) DOS-parancs végrehajtása után kapjuk ezt az üzenetet. Az üzenet közli a törölt file-ok számát is.

20 READ ERROR

A blokk bevezető információja hiányzik. A DOS a kívánt blokk elején levő információt (header) nem tudja azonosítani. Ennek az oka egy nem létező szektorszám is lehet. Elképzelhető, hogy a 'header' megsérült.

21 READ ERROR

A blokk szinkronizációs byte-jai hiányoznak. A DOS képtelen a blokk kezdetét jelentő szinkronizációs jelet megtalálni az adott sávban (track-en). Az oka lehet az író/olvasó fej hibás mozgása, a lemez hiánya, vagy nem formázott lemez használata.

22 READ ERROR

Az adatblokk hiányzik. A DOS egy nem megfelelően felírt blokkot kísérelt meg olvasni vagy ellenőrizni. A hiba a B-R és B-W DOS parancsok esetén fordul elő.

23 READ ERROR

Ellenőrző összeg hiba. A DOS a blokkot be tudta olvasni a lemezegység pufferébe, de az ellenőrző összeg nem egyezik meg a blokkban tárolttal.

24 READ ERROR

Byte dekódolási hiba. A blokkot, vagy az előtte levő bevezető információt a DOS-nak sikerült a pufferba beolvasnia, de egy nem megengedett bitképet talált.

25 WRITE ERROR

Az írás ellenőrzése közben a DOS eltérést talált a memóriájában tárolt és a blokkba kiírt adatok közt.

26 WRITE PROTECT ON

A DOS ezt a hibaüzenetet küldi, ha egy blokk lemezre való írását kíséreltük meg, és az írást engedélyező rés a lemezen nem volt kivágva.

27 READ ERROR

A DOS a blokk bevezető információjának (header) olvasása közben ellenőrző összeg hibát észlelt.

28 WRITE ERROR

Túl hosszú adatblokk. A DOS-nak egy adatblokk írásának a befejezése után meghatározott időn belül érzékelnie kell a következő blokk szinkronizációs jelét. Ha ez nem történik meg, a fenti hibajelzést kapjuk.

29 DISK ID MISMATCH

Ez az üzenet egyaránt jelentheti, hogy a lemezt még nem formáztuk meg, vagy pedig azt, hogy a blokk bevezető információja tönkrement, s ezért a DOS nem megfelelő ID számot olvasott be.

30 SYNTAX ERROR

A 15.csatornán elküldött üzenetet a DOS képtelen értelmezni. (Például a parancsban a kelleténél több file név szerepel).

31 SYNTAX ERROR

A DOS a 15.csatornán elküldött parancsot nem tudja azonosítani. A parancsnak az első pozíción kell kezdődnie.

32 SYNTAX ERROR

A parancs hosszabb, mint 58 karakter.

33 SYNTAX ERROR

Érvénytelen file név. A SAVE vagy OPEN utasításokban nem megfelelő paraméterek szerepelnek.

34 SYNTAX ERROR

A file név hiányzik. A DOS nem találta meg a parancsban a file nevét. (Például a kettőspont hiányzik a parancsból.)

39 SYNTAX ERROR

A 15.csatornán elküldött parancs azonosíthatatlan.

50 RECORD NOT PRESENT

Az INPUT# vagy GET# utasítások használata során a relatív file utolsó rekordján túl akartunk olvasni.

51 OVERFLOW IN RECORD

A PRINT# utasítás végrehajtása során elértük a rekord határt, és így a be nem fért byte-ok elvesztek.

52 FILE TOO LARGE

Relatív file használata kőzben olyan nagy rekordszámot használtunk, amelyik a lemez betelését eredményezné.

60 WRITE FILE OPEN

Egy írásra megnyitott file-t, annak lezárása nélkül, olvasásra kíséreltünk meg újból megnyitni.

61 FILE NOT OPEN

Ezt az üzenetet akkor kapjuk, amikor egy, a DOS által még meg nem nyitott file-t kíséreltünk meg használni.

62 FILE NOT FOUND

A file, amire hivatkoztunk egy BASIC utasításban vagy DOS parancsban, nem szerepel a lemez katalógusában.

63 FILE EXISTS

Az újonnan létrehozni kívánt file neve már szerepel a katalógusban.

64 FILE TYPE MISMATCH

A megadott file típusa nem egyezik meg a katalógusban szereplő típussal.

65 NO BLOCK

Az üzenet jelzi, hogy a B-A DOS parancs segítségével lefoglalni kívánt blokk már foglalt. A sáv- és szektorszám a legelső, nagyobb indexű szabad blokk adatait tartalmazza. Ha mindkettő 0, akkor ilyen szabad blokk már nincs.

66 ILLEGAL TRACK AND SECTOR

A DOS végrehajtása közben egy nem létező sáv- és szektorszám párosításnak megfelelő blokkot akartunk használni

67 ILLEGAL SYSTEM T OR S

A DOS egyáltalán nem létező sávot vagy szektort próbált meg használni.

70 NO CHANNEL

A kívánt csatorna nem elérhető, vagy már az összes csatornát használjuk. A DOS egyszerre 5 szekvenciális vagy 6 közvetlen elérésű file-t tud kezelni.

71 DIRECTORY ERROR

A lemezen levő BAM nem egyezik meg a DOS memóriájában levővel. Ilyenkor az I parancs kiadásával célszerű újrainicializálni a lemezegységet.

72 DISC FULL

Nincs szabad blokk a lemezen, vagy betelt a katalógus. Ha a katalógusban kijelzett blokkszámok összege kevesebb mint 664, célszerű a 'V' parancs kiadása. Ezel az 'elveszett' blokkokat vissza lehet nyerni.

73 CBM DOS V2.6 1541

A DOS 1.x és 2.x variánsai egymás lemezeit kölcsönösen olvassák, de nem írják felül. A fenti üzenet azt jelenti, hogy egy, a nem megfelelő verziójú DOS-szal formázott lemezre akarunk írni. Bekapcsolás után ugyanezt az üzenetet kapjuk.

74 DRIVE NOT READY

Nincs lemez a lemezmeghajtóban.

75 FORMAT SPEED ERROR

Ha a lemez formázása közben a 8250-es lemezegység úgy érzékeli, hogy a lemez sebessége a normálistól több mint 1%-kal eltér, a formázás abbamarad, és a fentl hibajelzést kapjuk.

76 CONTROLLER ERROR

A lemezegység hardver hibájának érzékelésekor kapjuk a fenti hibajelzést. A készüléket ki kell kapcsolni és megjavíttatni.

F.4 SIMONS' BASIC hibaüzenetek

BAD MODE

Valamilyen utasítás paraméterét nem a megengedett módon használjuk. (Ez lehet típus vagy nagyság eltérése, a paraméter hiánya.)

NOT HEX CHARACTER

A $ jel után **négy** hexadecimális számjegy kell, hogy következzék (0-9,A-F).

NOT BINARY CHARACTER

A % jel után **nyolc** bináris számjegy kell, hogy következzék (0 vagy 1).

UNTIL WITHOUT REPEAT

Az interpreter egy UNTIL utasítást kísérelt meg végrehajtani, de nem találja a hozzá tartozó REPEAT utasítást.

END LOOP WITHOUT LOOP

Az interpreter egy END LOOP utasítást kisérelt meg végrehajtani, de nem találja a hozzá tartozó LOOP utasítást.

END PROC WITHOUT EXEC

Az interpreter egy END PROC utasítást kísérelt meg végrehajtani, de nem találja a hozzá tartozó EXEC utasítást.

PROC NOT FOUND

A CALL vagy EXEC utasításban megnevezett címke nem létezik.

NOT ENOUGH LINES

A sprite vagy karakter alakjának definiálására szolgáló @ utasítások száma kevés.

BAD CHAR FOR A MOB

A sprite vagy karakter alakját definiáló @ utasítás nem megengedett karaktert tartalmaz.

STACK TOO LARGE

Nem megfelelően struktúrált programok esetén előfordulhat, hogy az interpreter képtelen azonosítani, mi is hiányzik a veremből. Ekkor kapjuk a fenti hibaüzenetet.

F.5 Szabvány BASIC programok átalakítása C64 BASIC-re

Ha nem C-64 BASIC-ben írt BASIC programokkal rendelkezünk és ezeket C-64-on akarjuk futtatni, akkor előbb néhány apróbb változtatást kell elvégeznünk. Néhány tanácsot adunk, hogy ezt az átalakítást megkönnyítsük.

Sztring típusú tömbök

A C-64 BASIC a DIM A$(I,J) utasítást úgy értelmezi, hogy egy kétdimenziós, (I+1)*(J+1) méretű tömböt definiáltunk. A standard BASIC-ben ez egy egy dimenziós (J elemszámú) tömböt definiál, melynek minden egyes eleme egy I hosszúságú sztring. Ezért ezeket a deklarációkat át kell írni; DIM A$(I,J)-ből például egyszerűen DIM A$(J) lesz

A C-64 BASIC a sztringműveletekre a MID$, RIGHT$, illetve a LEFT$ függvényeket használja. A standard BASIC és más BASIC-ek is megengedik az A$(I TO J) használatát, ami az A$ sztring I-ik karakterétől J-ik karakteréig terjedő sztringet jelenti. Az A$(I TO J)=X$ értékadás ebben az esetben azt jelenti, hogy ezt a részt az X$-ra cseréljük ki. Ezt az utasítást C-64-en így valósítjuk meg:

```
A$=LEFT$(A$,I-1)+X$+MID$(A$,J+1)
```

Többszörös értékadás

Néhány BASIC interpreter megengedi a 10 LET B=C=0 típusú értékadást, melynek hatására B is és C is 0 lesz. A C-64-en ilyen értékadás nincs. A fenti utasítást a gép LET B=(C=0)-nak értelmezi és attól függően, hogy C nulla volt-e vagy sem, B értéke -1, illetve 0 lesz. A kívánt hatást a 10 B=0:C=0 utasítással érhetjük el.

Több utasításból álló sorok

Nem mindegyik BASIC interpreter használja a kettőspontot az egy sorban levő utasítások elválasztására. Ezeket a C-64-en természetesen kettősponttal kell elválasztani.

MAT függvények

Elsősorban minigépeken futó BASIC interpreterek lehetővé teszik mátrix műveletek használatát. Ezeket C-64-en külön alprogramok megírásával lehet csak előállítani.

F.6 Matematikai függvények

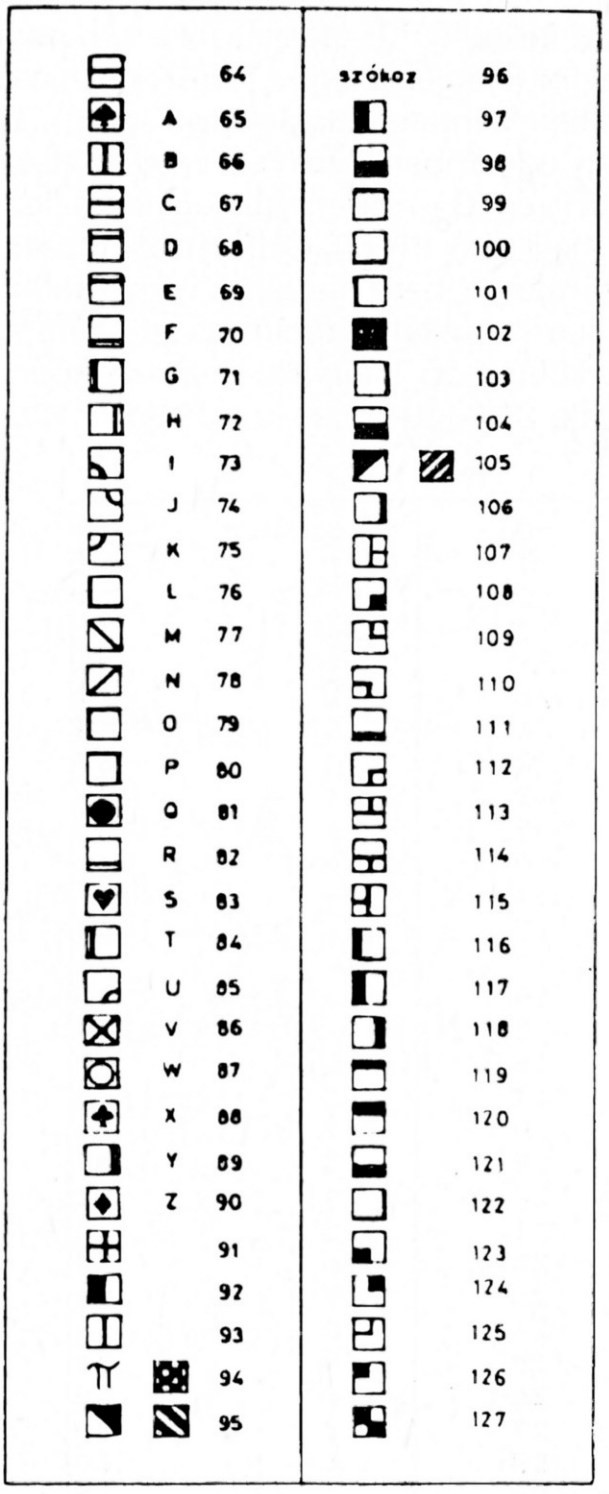
```
100 REM SZEKANS(X)
110 DEF FN SEC(X)=1/COS(X)
120 REM
130 REM KOSZEKANS(X)
140 DEF FN CSC(X)=1/SIN(X)
150 REM
160 REM KOTANGENS(X)
170 DEF FN COT(X)=1/TAN(X)
180 REM
190 REM ARKUSZ SZINUSZ
200 DEF FN ARCSIN(X)=ATN(X/SQR(-X*X+1)
210 REM
220 REM ARKUSZ KOSZINUSZ
230 DEF FN ARCCOS(X)=-ATN(X/SQR(-X*X+1))+<pi>/2
240 REM
250 REM ARKUSZ SZEKANS
260 DEF FN ARCSEC(X)=ATN(X/SQR(X*X-1))
270 REM
280 REM ARKUSZ KOSZEKANS
290 DEF FN ARCCSC(X)=ATN(X/SQR(X*X-1))+(SGN(X)-1)*<pi>/2
300 REM
310 REM ARKUSZ KOTANGENS
320 DEF FN ARCOT(X)=ATN(X)+<pi>/2
330 REM
340 REM SZINUSZ HIPERBOLIKUSZ
350 DEF FN SINH(X)=(EXP(X)-EXP(-X))/2
360 REM
370 REM KOSZINUSZ HIPERBOLIKUSZ
380 DEF FN COSH(X)=(EXP(X)+EXP(-X))/2
390 REM
400 REM TANGENS HIPERBOLIKUSZ
410 DEF FN TANH(X)=EXP(-X)/(EXP(X)+EXP(-X))*2+1
420 REM
430 REM AREA SZINUSZ HIPERBOLIKUSZ
440 DEF FN ARCSINH(X)=LOG(X+SQR(X*X+1))
450 REM
460 REM AREA KOSZINUSZ HIPERBOLIKUSZ
470 DEF FN ARCCOSH(X)=LOG(X+SQR(X*X-1))
480 REM
490 REM AREA TANGENS HIPERBOLIKUSZ
500 DEF FN ARCTANH(X)=LOG((1+X)/(1-X))/2
```

F.7 Képernyő kódok

Az alábbi táblázatban megtalálhatjuk a C-64 teljes karakterkészletét. A táblázat megadja, hogy milyen kódot kell a POKE utasítással a képernyő memóriába beírnunk ahhoz, hogy a kívánt karakter jelenjen meg. A PEEK utasítással kiolvasott adatokról megállapíthatjuk, hogy mely karakternek felelnek meg. Két karakterkészlet áll rendelkezésünkre, azonban egyidőben csak az egyiket használhatjuk. A karakterkészletek átváltása a SHIFT és C= billentyűk egyidejű lenyomásával történik. A BASIC-ből PRINT CHR$(14), illetve PRINT CHR$(142) utasításokkal állíthatjuk be a kívánt karakterkészletet. Bármelyik betű, karakter vagy grafikus jel inverz alakban is megjeleníthető. Az inverz karaktereket a táblázatban szereplő értékek 128-cal való megnövelésével kapjuk. A következő intervallumba eső kódszámú karakterek a két karakterkészletben azonosak: 27-64,91-93,96-104,106-121,123-127.

@	0	szóköz	32
A a	1	!	33
B b	2	"	34
C c	3	#	35
D d	4	$	36
E e	5	%	37
F f	6	&	38
G g	7	'	39
H h	8	(	40
I i	9	)	41
J j	10	*	42
K k	11	+	43
L l	12	,	44
M m	13	-	45
N n	14	.	46
O o	15	/	47
P p	16	0	48
Q q	17	1	49
R r	18	2	50
S s	19	3	51
T t	20	4	52
U u	21	5	53
V v	22	6	54
W w	23	7	55
X x	24	8	56
Y y	25	9	57
Z z	26	:	58
[	27	;	59
<font>	28	<	60
]	29	=	61
↑	30	>	62
vissza-nyíl	31	?	63

Képernyő kódok (folytatás):



F.8 ASCII kódok

Ez a függelék a PRINT CHR$(X) utasítás hatására megjelenő karaktereket tartalmazza táblázatos formában, X összes lehetséges értékére. Ugyanígy visszakereshető a táblázatból az is, hogy milyen számkódot kapunk a PRINT ASC("S") utasítás eredményéül, ha S tetszőleges karakter. A táblázat hasznos segítséget nyújt a GET utasítással beolvasott karakterek kiértékeléséhez, üzemmód váltáshoz, és olyan karakterek kiírásához, amelyek nem szerepelhetnek idézőjelben (pl. a karakterkészlet kapcsoló.)

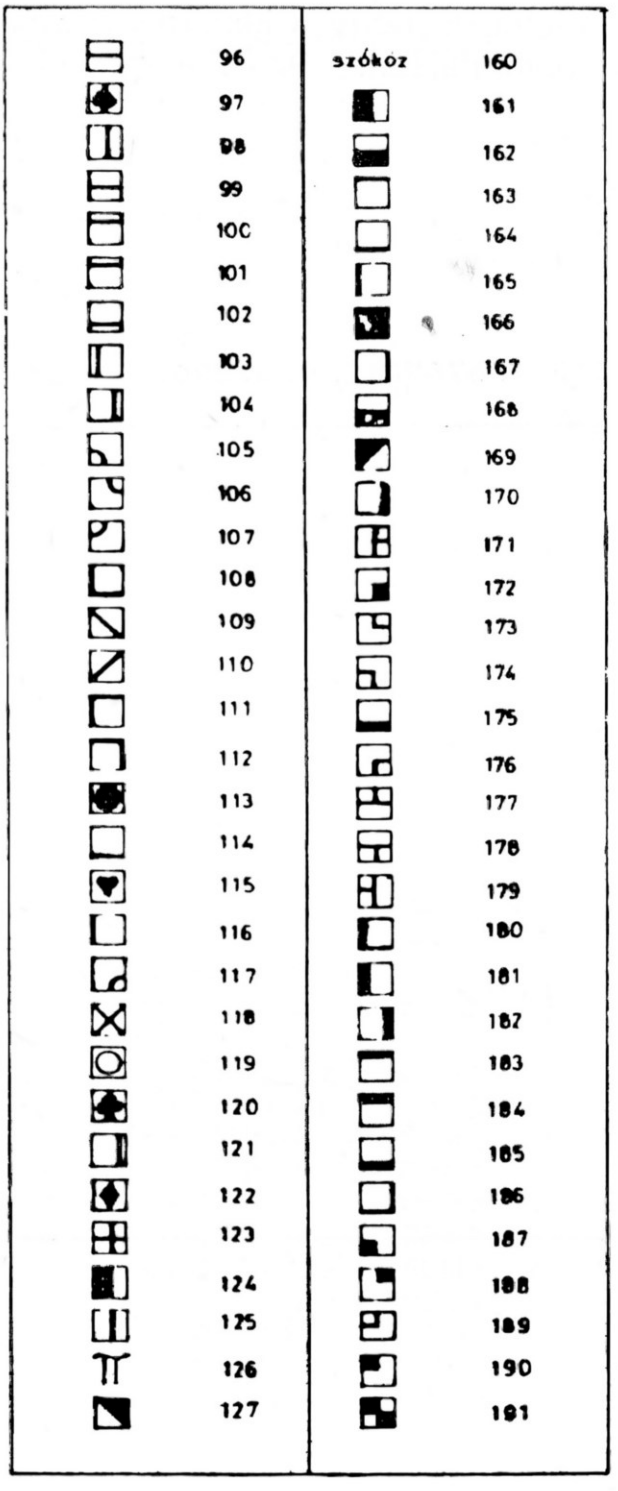
A karakterek egy része vezérlő karakter. Ezek 'kiírása' a megfelelő vezérlő funkció végrehajtását jelenti.

Alfanumerikus és vezérlő kódok:

	0	szóköz	32
	1	!	33
	2	"	34
	3	#	35
	4	$	36
<fehér>	5	%	37
	6	&	38
	7	'	39
<C=-CTRL> tiltás	8	(	40
<C=-CTRL>	9	)	41
engedély	10	*	42
	11	+	43
	12	,	44
<RETURN>	13	-	45
<betűk>	14	.	46
	15	/	47
	16	0	48
<CRSR LE>	17	1	49
<RVSON>	18	2	50
<HOME>	19	3	51
<DEL>	20	4	52
	21	5	53
	22	6	54
	23	7	55
	24	8	56
	25	9	57
	26	:	58
	27	;	59
<piros>	28	<	60
<CRSR JOBBRA>	29	=	61
<zöld>	30	>	62
<kék>	31	?	63

@	64		128
A	65	<narancs>	129
B	66		130
C	67		131
D	68		132
E	69	f1	133
F	70	f3	134
G	71	f5	135
H	72	f7	136
I	73	f2	137
J	74	f4	138
K	75	f6	139
L	76	f8	140
M	77	<Shift RETURN>	141
N	78	<grafikák>	142
O	79		143
P	80	<feketa>	144
Q	81	<CRSR FEL>	145
R	82	<RVSOFF>	146
S	83	<CLR>	147
T	84	<INST>	148
U	85	<barna>	149
V	86	<vil.piros>	150
W	87	<szürke1>	151
X	88	<szürke2>	152
Y	89	<vil.zold>	153
Z	90	<vil.kék>	154
[	91	<szürke3>	155
<font>	92	<bibor>	156
]	93	<CRSR BALRA>	157
↑	94	<sárga>	158
vissza-nyíl	95	<türkisz>	159

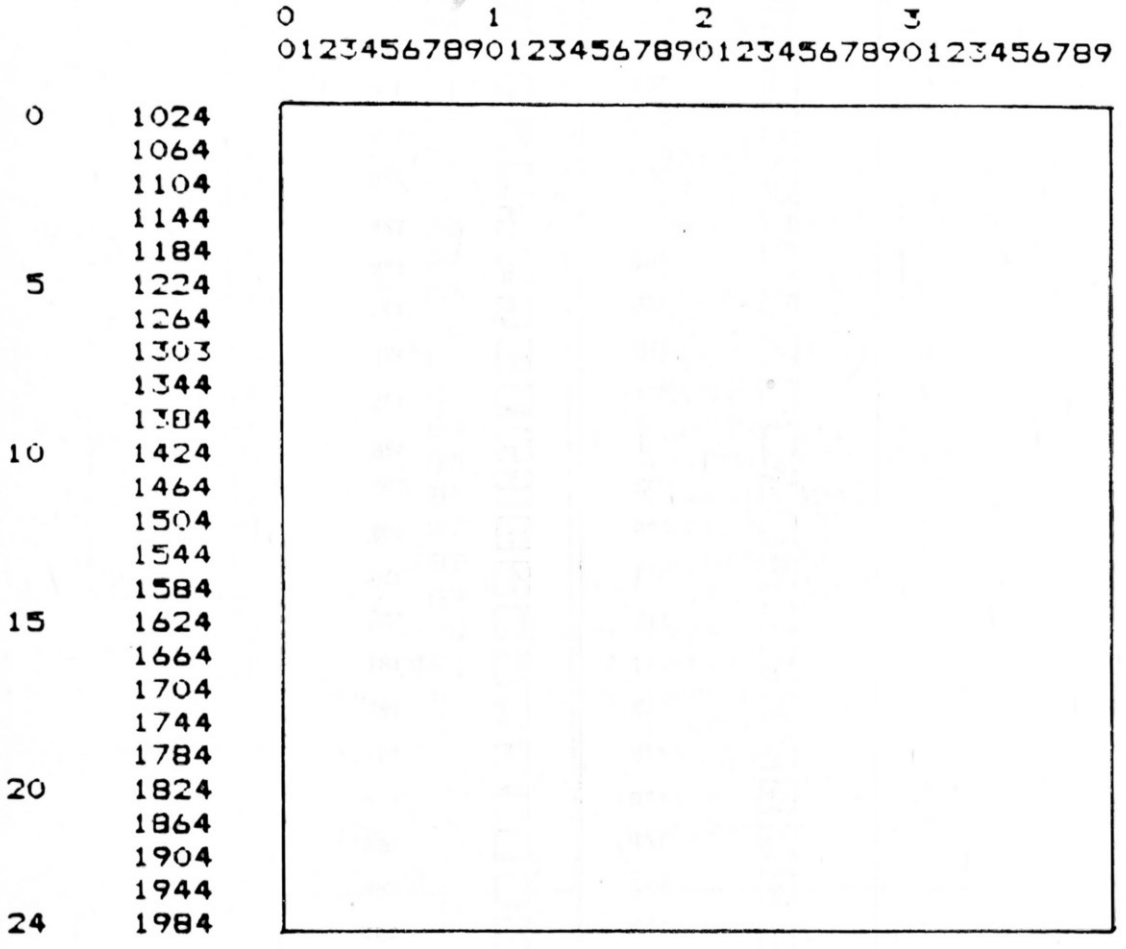
ASCII kódok (folytatás: grafikus jelek):



F.9 Képernyő és szín memória

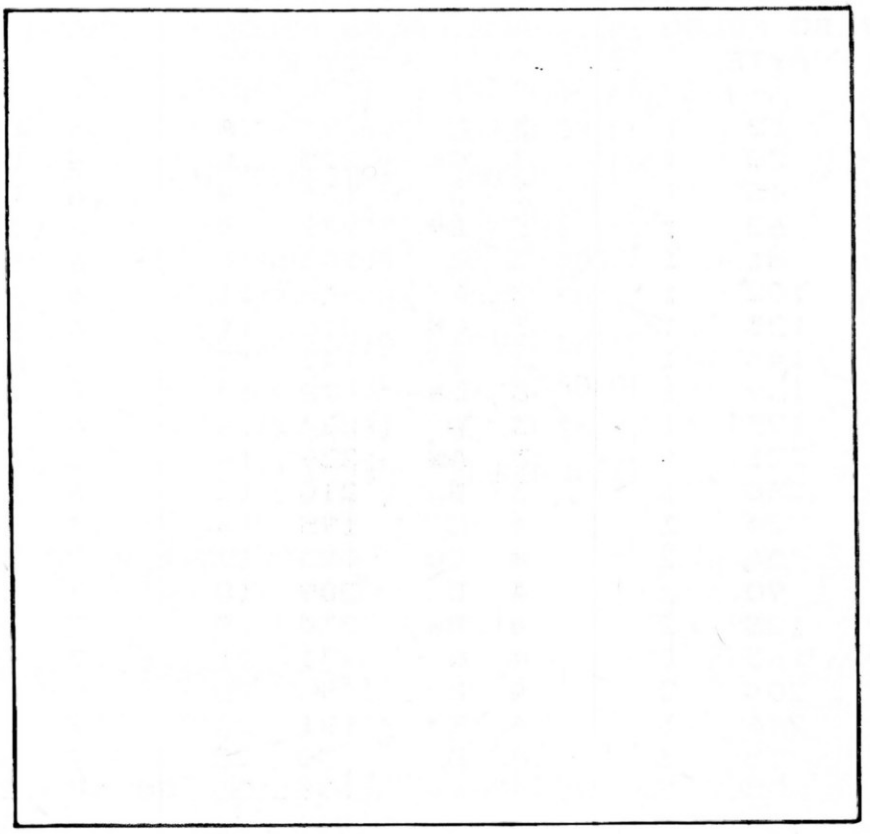
A következő ábrák azt mutatják, hogy a képernyő egyes karakterhelyeinek melyik cím felel meg a C-64 memóriájában, illetve melyik cím definiálja a szóbanforgó karakterhely színét.

Képernyő memória



Szín memória

```
                0         1         2         3
                0123456789012345678901234567890123456789
 0    55296
      55336
      55376
      55416
      55456
 5    55496
      55536
      55576
      55616
      55656
10    55696
      55736
      55776
      55816
      55856
15    55916
      55956
      55996
      56016
      56056
20    56096
      56136
      56176
      56216
24    56256
```



Kód	Szín	Billentyűzés	
0	feketa	<CTRL 1>	<BLK>
1	fehér	<CTRL 2>	<WHT>
2	vörös	<CTRL 3>	<RED>
3	ciánkék(türkiz)	<CTRL 4>	<CYN>
4	bíbor (lila)	<CTRL 5>	<PUR>
5	zöld	<CTRL 6>	<GRN>
6	kék	<CTRL 7>	<BLU>
7	sárga	<CTRL 8>	<YEL>
8	narancs	<C= 1>	
9	barna	<C= 2>	
10	rózsaszín	<C= 3>	
11	szürke1	<C= 4>	
12	szürke2	<C= 5>	
13	világoszöld	<C= 6>	
14	világoskék	<C= 7>	
15	szürke3	<C= 8>	

F.10 Frekvencia értékek

HANG		ALSO BYTE	FELSO BYTE
0	C	12	1
0	C#	28	1
0	D	45	1
0	D#	62	1
0	E	81	1
0	F	102	1
0	F#	123	1
0	G	145	1
0	G#	169	1
0	A	195	1
0	A#	221	1
0	B	250	1
1	C	24	2
1	C#	56	2
1	D	90	2
1	D#	125	2
1	E	163	2
1	F	204	2
1	F#	246	2
1	G	35	3
1	G#	83	3
1	A	134	3
1	A#	187	3
1	B	244	3
2	C	48	4
2	C#	112	4
2	D	180	4
2	D#	251	4
2	E	71	5
2	F	152	5
2	F#	237	5
2	G	71	6
2	G#	167	6
2	A	12	7
2	A#	119	7
2	B	233	7

HANG		ALSO BYTE	FELSO BYTE
3	C	97	8
3	C#	225	8
3	D	104	9
3	D#	247	9
3	E	143	10
3	F	48	11
3	F#	218	11
3	G	143	12
3	G#	78	13
3	A	24	14
3	A#	239	14
3	B	210	15
4	C	195	16
4	C#	195	17
4	D	209	18
4	D#	239	19
4	E	31	21
4	F	96	22
4	F#	181	23
4	G	30	25
4	G#	156	26
4	A	49	28
4	A#	223	29
4	B	165	31
5	C	135	33
5	C#	134	35
5	D	162	37
5	D#	223	39
5	E	62	42
5	F	193	44
5	F#	107	47
5	G	60	50
5	G#	57	53
5	A	99	56
5	A#	190	59
5	B	75	63

HANG		ALSO BYTE	FELSO BYTE
6	C	15	67
6	C#	12	71
6	D	69	75
6	D#	191	79
6	E	125	84
6	F	131	89
6	F#	214	94
6	G	121	100
6	G#	115	106
6	A	199	112
6	A#	124	119
6	B	151	126
7	C	30	134
7	C#	24	142
7	D	139	150
7	D#	126	159
7	E	250	168
7	F	6	179
7	F#	172	189
7	G	243	200
7	G#	230	212
7	A	143	225
7	A#	248	238
7	B	46	253

F.11 Memória térkép

1. BASIC rendszerváltozók	0-1023	$0000-$03FF
2. Kazetta puffer	828-1019	$033C-$03FB
3. Képernyő memória *	1024-2023	$0400-$07E7
4. Sprite-mutatók *	2040-2047	$07F8-$07FF
5. BASIC munkaterület	2048-40959	$0800-$9FFF
6. BASIC ROM	40960-49151	$A000-$BFFF
7. CHARACTER ROM	53248-57343	$D000-$DFFF
8. VIC regiszterek	53248-53294	$D000-$D02E
9. SID regiszterek	54272-54300	$D400-$D41C
10. Szín memória	55296-56319	$D800-$DBFF
11. CIA#1 regiszterek	56320-56335	$DC00-$DC0F
12. CIA#2 regiszterek	56576-56831	$DD00-$DD0F
13 KERNAL ROM	57344-65535	$E000-$FFFF

A â jelölt memóriarészek helye változtatható.

F.12 I/O eszközök csatlakozói

Röviden összefoglaljuk, milyen ki/bemeneti csatlakozókkal rendelkezik a C-64 számítógép. Ezek sorrendben a következők:

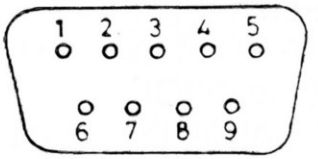
1/ játék csatlakozók (2 db.),
2/ bővítő egység (cartridge) csatlakozója,
3/ audio/video kimenet,
4/ soros busz (lemezegység, nyomtató),
5/ modulátor kimenet (antenna csatlakozó),
6/ kazettás egység csatlakozója,
7/ felhasználói I/O csatlakozója (user port).

Az egyes csatlakozókhoz tartozó szoftver/hardver elemeket a 6. fejezetben ismertettük. Ebben a függelékben a csatlakozók kiosztását, idődiagramokat és néhány, a könyvhöz szorosan nem tartozó tulajdonságát ismertetjük ezeknek az eszközöknek.

Játék csatlakozók

A játék csatlakozókkal mind botkormány, mind potméter csatlakoztatható. Az egyes lábak kiosztása a következő:

Láb	1.JÁTÉK I/O JEL	2.JÁTÉK I/O JEL
1	JOYA0	JOYB0
2	JOYA1	JOYB1
3	JOYA2	JOYB2
4	JOYA3	JOYB3
5	POTAY	POTBY
6	BUTTON A/LP	BUTTON B
7	+5V	+5V max 50mA
8	GND	GND
9	POT AX	POT BX



Bővítő egység (cartridge)

A csatlakozó egy 44 érintkezős "anya", ami a C-64 hátoldalán helyezkedik el. A csatlakozó a C-64 rendszer olyan bővítésére használható fel, amelynek szüksége van a processzor adat- és cím-buszára. A csatlakoztatott egység meghibásodása a C-64 chipjeit is tönkreteheti. Az egyes lábak számozása a következő:

Láb	Jelölés	Funkció
1	GND	Rendszer föld
2	+5VDC	A felhasználói port és a CARTRIDGE
3	+5VDC	fogyasztása nem haladja meg a 450 mA-t
4	$\overline{\text{IRQ}}$	Megszakítás kérelem (6510 lábára)
5	R/W	READ/WRITE
6	DOT CLOCK	8.18 MHz-es video órajel
7	$\overline{\text{I/O1}}$	I/O blokk 1 ($DE00-$DEFF, meghajtás nélküli)
8	GAME	TTL input
9	$\overline{\text{EXROM}}$	TTL input
10	$\overline{\text{I/O2}}$	I/O blokk 2 ($DF00-$DFFF meghajtott LS TTL kimenet)
11	ROML	8K dekódolt RAM/ROM blokk ($8000, meghajtott LS TTL kimenet)
12	BA	Busz engedélyezése (VIC-II áramkör)
13	$\overline{\text{DMA}}$	Közvetlen memória elérési kérelem

14	D7	Adat busz 7. bit
15	D6	Adat busz 6. bit
16	D5	Adat busz 5. bit
17	D4	Adat busz 4. bit
18	D3	Adat busz 3. bit
19	D2	Adat busz 2. bit
20	D1	Adat busz 1. bit
21	D0	Adat busz 0. bit
22	GND	Rendszer föld

A	$\overline{\text{GND}}$	Rendszer föld
B	$\overline{\text{ROMH}}$	8K dekódolt RAM/ROM blokk ($E000, meghajtott)
C	$\overline{\text{RESET}}$	6502 Reset láb meghajtott output/ meghajtás nélküli input)
D	$\overline{\text{NMI}}$	6502 nem maszkolható megszakítás meghajtott TTL output/ meghajtás nélküli input
E	Φ2	Rendszer órajel
F	A15	Cím busz 15. bit
H	A14	Cím busz 14. bit
J	A13	Cím busz 13. bit
K	A12	Cím busz 12. bit
L	A11	Cím busz 11. bit
M	A10	Cím busz 10. bit
N	A9	Cím busz 9. bit
P	A8	Cím busz 8. bit
R	A7	Cím busz 7. bit
S	A6	Cím busz 6. bit
T	A5	Cím busz 5. bit
U	A4	Cím busz 4. bit
V	A3	Cím busz 3. bit
W	A2	Cím busz 2. bit
X	A1	Cím busz 1. bit
Y	A0	Cím busz 0. bit
Z	GND	Rendszer föld

A felülvonás az aktív vonalakat jelenti.

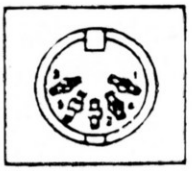
Néhány csatlakozó láb szerepéről külön is szólunk. A 6. lábon jelenik meg a 8.18 MHz-es video órajel. A rendszer összes többi órajele erről a frekvenciáról van leosztva.

A 12. láb a VIC-II chip BA (Bus Available) jelű, a busz használatát engedélyező vonala. Három Fi2 órajellel azelőtt válik alacsonnyá, hogy a VIC-II átveszi a rendszerbusz vezérlését, és mindaddig alacsony is marad, mig a VIC-II be nem fejezi az információ feldolgozását.

A 13. láb a DMA (Direct Memory Access) vonal. Amikor ez a vonal alacsony lesz, a 6510 processzor felfüggeszti a memória írását/ olvasását, és így egy külső processzor számára lehetővé válik, hogy átvegye a vezérlést a rendszerbuszok felett. A DMA vonalat csak a Fi2 órajel alacsony értékénél szabad 0-ba vinni. Mivel a VIC-II áramkör továbbra is folytatja a képernyő frissitését, a külső processzor müködését össze kell hangolni a VIC-II időzitésével. (Lásd a VIC-II chip időzités diagramját!)

Audio/Video kimenet

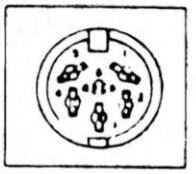
Láb	Jelzés
1	LUM
2	GND
3	AUDIO OUT
4	VIDEO OUT
5	AUDIO IN



Soros I/O csatlakozó

A soros buszra csatlakozó berendezések 'daisy chain'-be csatolhatók, ami lehetővé teszi, hogy egyszerre több eszközzel is kommunikálhasson a C-64. A busz használatáról részletesen a 6. fejezetben szóltunk.

Láb	Jelzés
1	SRQ IN
2	GND
3	ATN IN/OUT
4	CLK IN/OUT
5	DATA IN/OUT
6	RESET



SRQ (Serial Service Request In)

A soros buszon levő bármely eszköz képes ezt a vonalat alacsony logikai szintre vinni, jelezvén kiszolgálási igényét a C-64-nek.

ATN IN/OUT (Serial Attention In/Out)

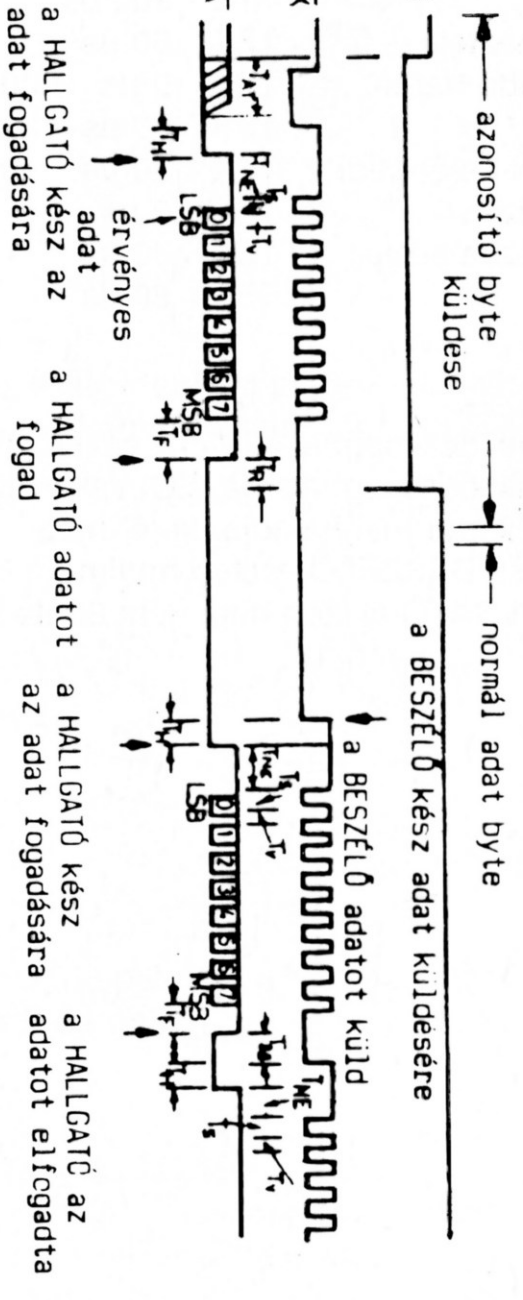
A C-64 ezt a jelet egy parancssorozat indításának a jelzésére használja. Amikor a C-64 alacsonyra viszi ezt a jelet, az összes, a buszon levő eszköz a C-64 által küldött címre figyel. A megcímzett eszköz egy adott időtartamon belül köteles válaszolni, máskülönben a C-64 úgy tekinti, hogy a megcímzett eszköz nincs rajta a buszon. A soros I/O időzítéseit a következő oldalon látható ábra foglalja össze. Az egyes időintervallumok a következők:

Leírás	Jel	Min.	Tip.	Max.
ATN válasz (1)	TAT	-	-	1000 us
HALLGATÓ késleltetése	TH	0	-	akármennyi
nem EOI válasz (2)	TNE	-	40 us	200 us
BESZÉLŐ jelzőbit (3)	TS	20 us	70 us	-
érvényes adat	TV	20 us	20 us	
szinkronizáció (4)	TF	0 us	20 us	1000 us
szinkronizáció az ATN végéig	TR	20 us	-	-
két byte közti idő	TBB	100 us		- -
EOI válaszidő	TYE	200 us		250 us -
EOI válaszidő késleltetése	TEI	60 us	-	-
BESZÉLŐ válaszidő határ	TRY	0 us	30 us	60 us
byte nyugtázás	TPR	20 us	30 us	
BESZÉLŐ az ATN-t elengedi	TTK	20 us	30 us	100 us
BESZÉLŐ nyugtázás	TDC	0 us	-	-
BESZÉLŐ nyugtázás késlelt.	TDA	80 us	-	-
EOI nyugtázás	TFR	60 us	-	-

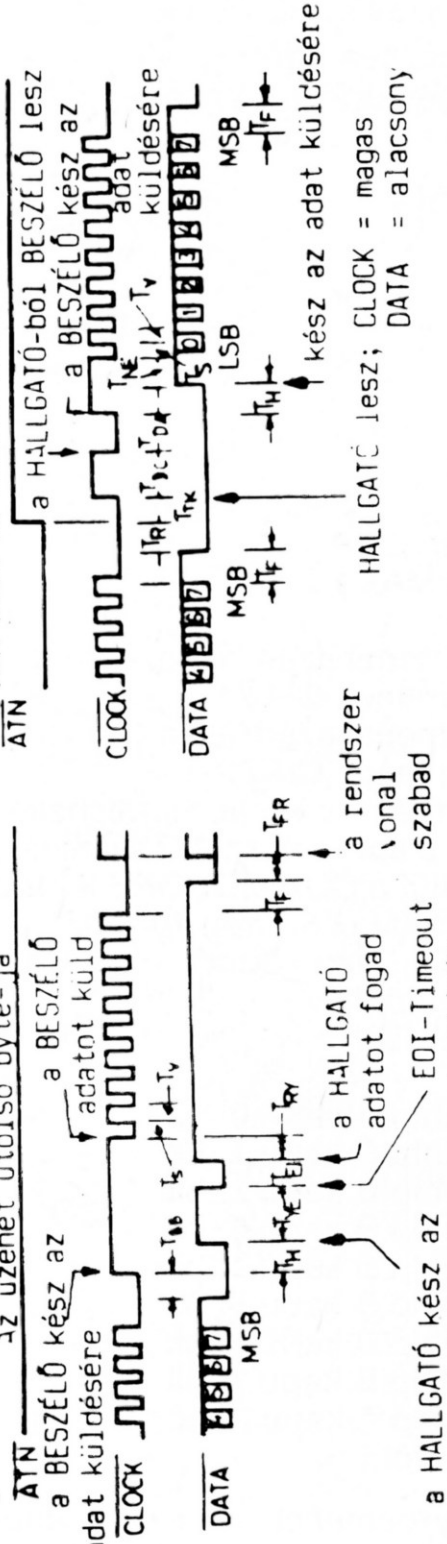
Megjegyzések:

1./ Ha a maximális időt meghaladja, "az eszköz nem létezik" hibajelzést kapunk.
2./ Ha a maximális időt meghaladja, EOI válasz szükséges.
3./ Ha a maximális időt meghaladja, keret hiba.
4./ TV és TPR külső BESZÉLŐ esetén minimum 60 us kell hogy legyen.
5./ TEI külső HALLGATÓ esetén minimum 80 us kell hogy legyen.

Soros busz időzítése

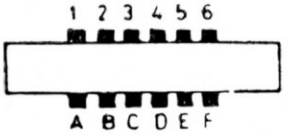


Soros busz időzítése

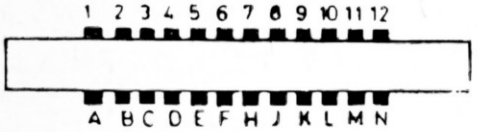


Kazettás egység csatlakozója

Láb	Funkció
A 1	GND
B 2	+5V
C 3	Kazetta motor
D 4	Kazetta olvasás
E 5	Kazetta írás
F 6	Billentyűzet



Felhasználói I/O csatlakozója



Láb	Jelölés	Funkció
1	GND	Rendszer föld
2	+5V	(100 mA MAX.)
3	RESET	
4	CNT1	Soros kimenet számlálója, CIA#1
5	SP1	Soros kimenet, CIA#1
6	CNT2	Soros kimenet számlálója CIA#2
7	SP2	Soros kimenet, CIA#2
8	PC2	Szinkronizációs vonal, Handshake CIA#2
9	Soros ATN	Ez a láb a soros busz ATN vonalára csatlakozik
10	9 VAC + fázis	Közvetlenül a COMMODORE 64 transzformátorára van kötve (Max 50 mA)
11	9 VAC-fázis	
12	GND	Rendszer föld
A	GND	Rendszer föld
B	FLAG2	
C	PB0	CIA#2 chip B kapu 0. bit
D	PB1	CIA#2 chip B kapu 1. bit
E	PB2	CIA#2 chip B kapu 2. bit
F	PB3	CIA#2 chip B kapu 3. bit
H	PB4	CIA#2 chip B kapu 4. bit
J	PB5	CIA#2 chip B kapu 5. bit
K	PB6	CIA#2 chip B kapu 6. bit
L	PB7	CIA#2 chip B kapu 7. bit
M	PA2	CIA#1 chip A kapu 2. bit
N	GND	Rendszerföld

A FLAG2 egy lefutó élre érzékeny bemenet, amely általános célú megszakító bemenetként használható. A FLAG2 vonalon minden egyes lefutó él bebillenti a megszakító bitet. Ha a megszakítás engedélyezve is volt, ez egy megszakítás kérést (IRQ) generál.

F.13 A 6510 mikroprocesszor specifikációja

A C-64 alapjául szolgáló mikroprocesszor egy egyszerű és olcsó mikroszámítógép, ami elsősorban a kis rendszerekhez kapcsolódó feladatok megoldására használható. Egy 8 bites kétirányú I/O kapu tartozik a chiphez, melynek átviteli regisztere a $0000 címen, az adatirányítás regisztere a $0001 címen van elhelyezve. A háromállapotú címbusz lehetővé teszi a közvetlen memória hozzáférést (DMA-t). A belső processzor-architektúra azonos a MOS-technológiával készült 6502-es mikroprocesszorral, és így szoftver-kompatibilisek.

A 6510 legfontosabb jellemzői

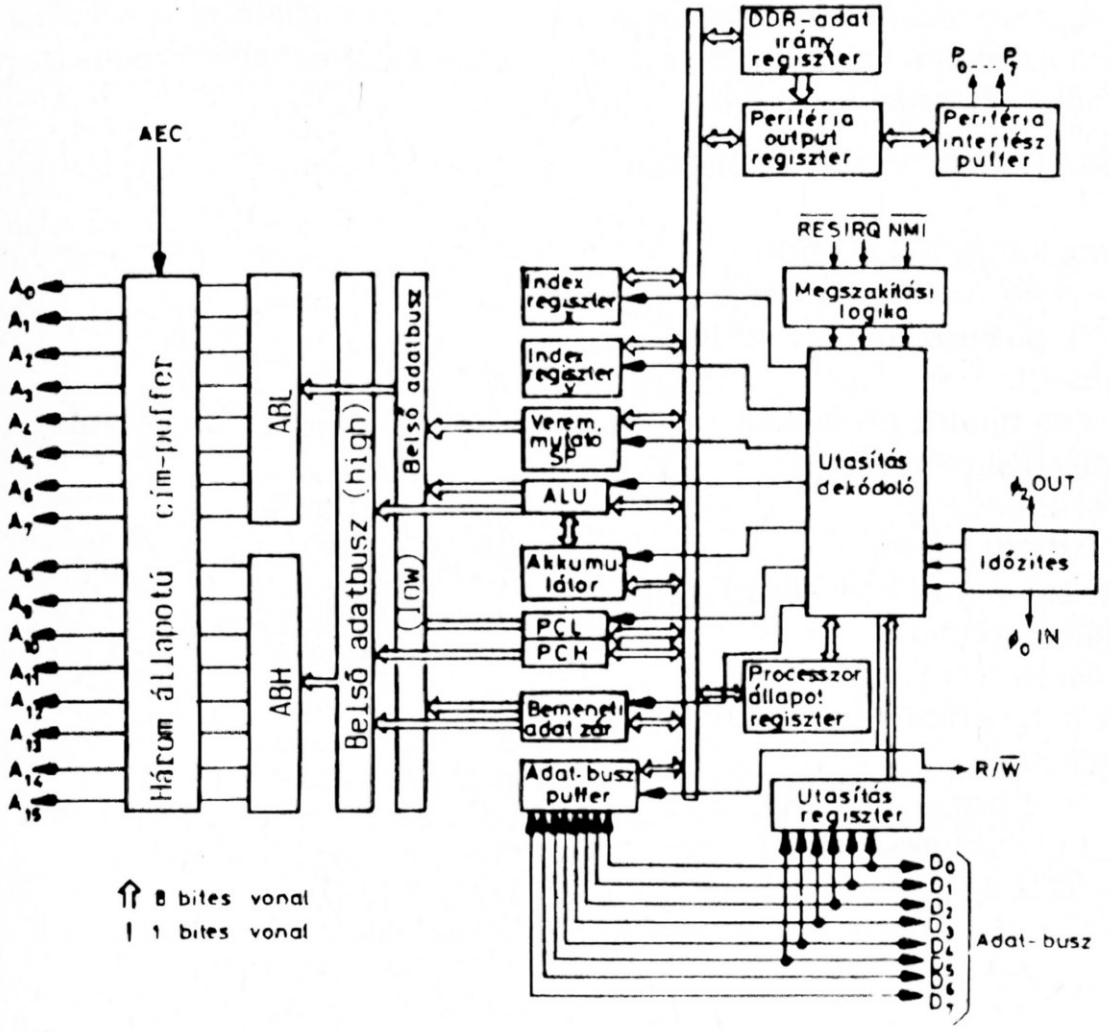
- Nyolcbites, kétirányú I/O-port
- Egyetlen, +5V-os tápfeszültség
- Nyolcbites, párhuzamos működés
- 56 utasítás
- Decimális és bináris aritmetika
- 13 fajta címzési mód
- Indexelési lehetőség
- Programozható verem
- Változtatható verem-hosszúság
- Megszakítási rendszer
- Nyolcbites, kétirányú adatbusz
- 64 Kbyte-nyi memória-tartomány címezhető
- DMA-lehetőség
- Busz-kompatibilitás az M6800-zal
- Pipeline architektúra
- 1 vagy 2 MHz-es órajel használata
- Bármely típusú vagy sebességű memória használható

Elektromos paraméterek

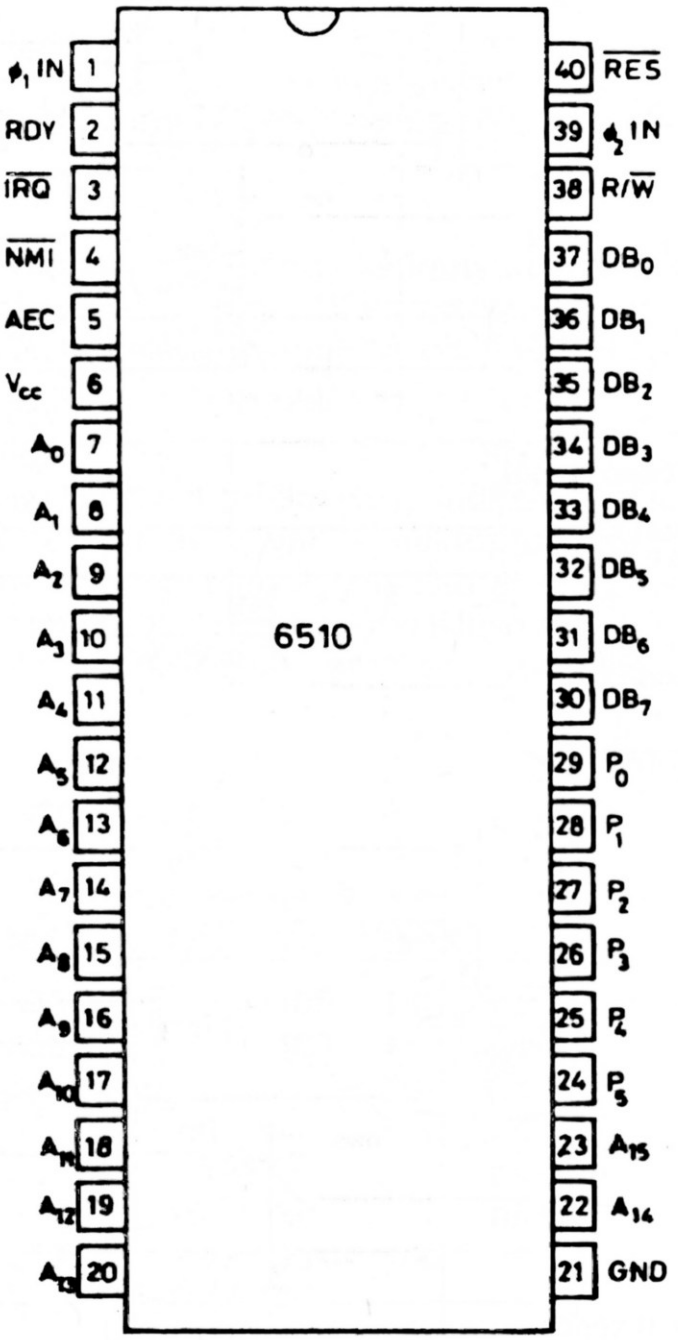
Paraméter	Jel	Érték	Mértékegység
Tápfeszültség	V_{CC}	-0.3-tól +7.0-ig	Vdc
Bemenő feszültség	V_{in}	-0.3-tól +7.0-ig	Vdc
Működési hőmérséklet	T_A	0-tól +70-ig	Celsius
Tárolási hőmérséklet	T_{STG}	-55-től +150-ig	Celsius

A mikroprocesszor ugyan tartalmaz bizonyos védelmet a nagy statikus feszültségek vagy elektromos terek ellen, de alkalmazásakor ügyelni kell, hogy elkerüljük a maximális értékeknéi nagyobb feszültségeket.

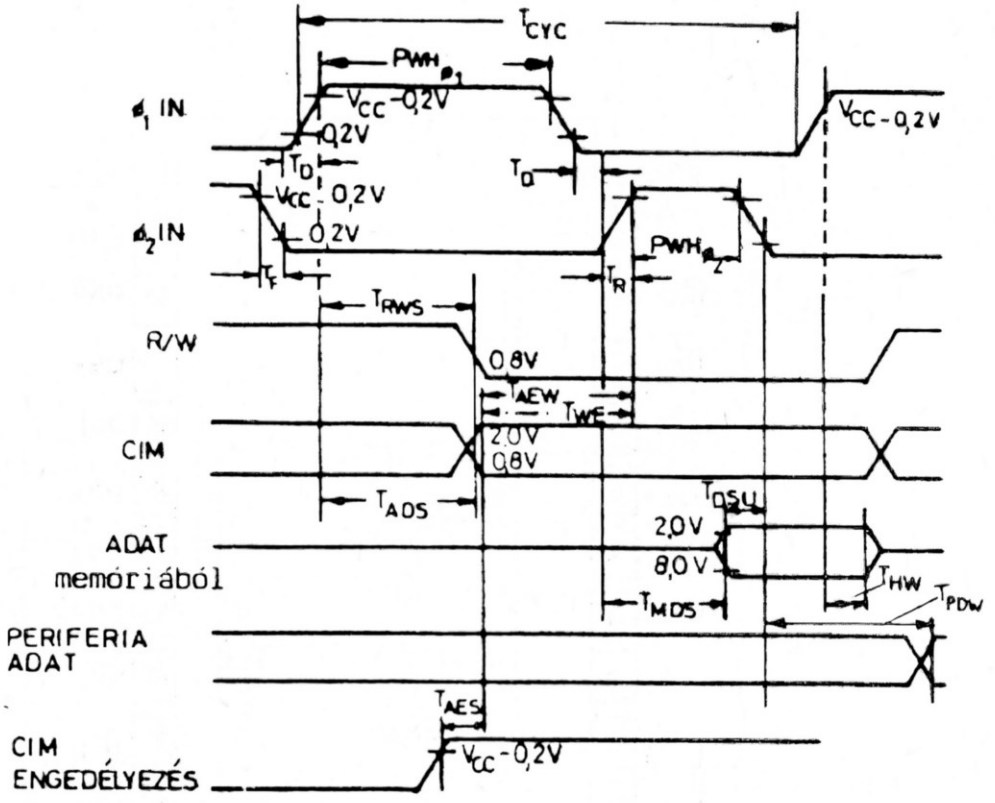
A 6510 processzor blokkdiagramja



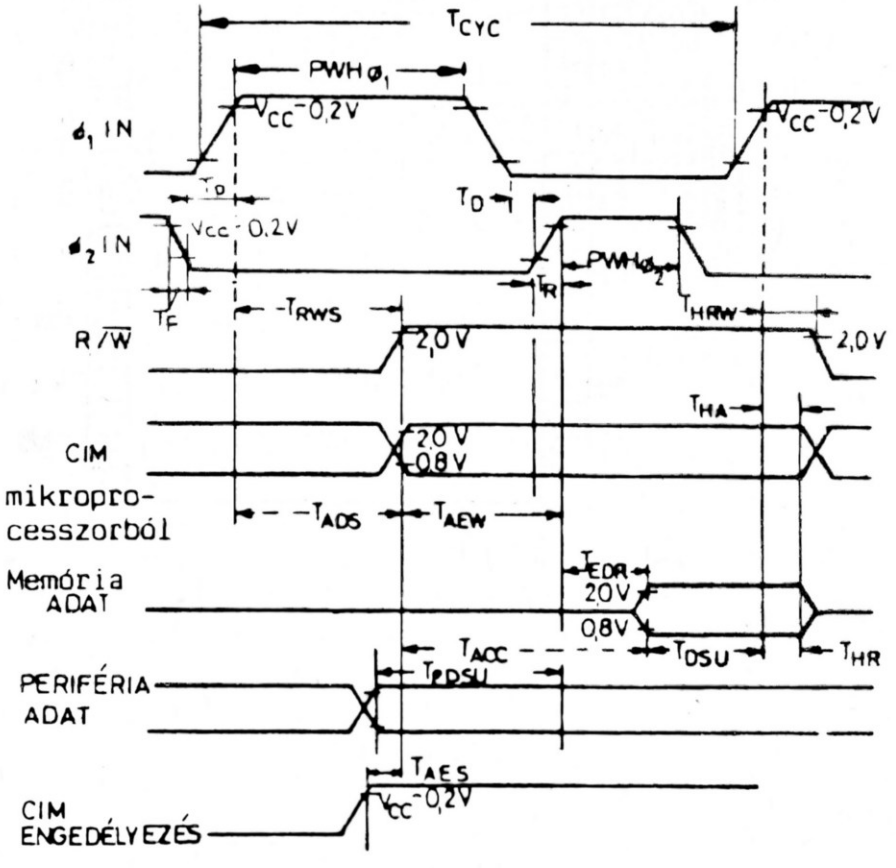
A 6510 processzor lábkiosztása



6510 írás diagramja



6510 olvasás diagramja



A 6510 vonalainak leírása

Órajelek (Φ1, Φ2)

A 6510-nek két Vcc feszültségszintű, ellenfázisú órajelre van szüksége.

Címbusz (A0-A15)

Ezek a TTL-szintű kimenetek egy TTL terhelés és 130pF meghajtására képesek.

Adatbusz (D0-D7)

8 láb szolgál az adatbusz céljaira. Ez egy kétirányú busz, melyen keresztül az adatforgalom a mikroprocesszor és a perifériák között lezajlik. A kimenetek háromállapotú pufferek, melyek egy TTL-terhelés és 130pF meghajtására képesek.

Reset (RES)

Ezen a bemeneten keresztül lehet törölni vagy indítani a mikroprocesszort egy aktív '0' szintű jellel. Míg ezen a lábon '0' van, a mikroprocesszor írása vagy olvasása le van tiltva. Mihelyt ezen a bemeneten egy felfutó él megjelenik, a mikroprocesszor rögtön megkezdi a reset-sorozatot. Hat óraciklusnyi rendszer-inicializálás után a maszkolható megszakítások letiltódnak, amit egy, a ($FFFC)-re való indirekt ugrás követ.

6510 írás/olvasás időzítése

Jel	1 MHz			2 MHZ			
	MIN	TIP	MAX	MIN	TIP	MAX	
TRWS	-	100	300	-	100	150	ns
TADS	-	100	300	-	100	150	ns
TACC	-	-	575	-	-	300	ns
TDSU	100	-	-	60			ns
THR	-	-					ns
THW	10	30	-	10	30		ns
TMDS	-	150	200	-	75	100	ns
THA	10	30	-	10	30		ns
THRW	10	30	-	10	30		ns
TAEW	180	-	-				ns
TEDR	-	-	395				ns
TDSU	300	-	-				ns
TWE	130	-	-				ns
TPDW	-	-	1				us
TPDSU	300	-	-				ns
TAES			60			60	ns

6510 Elektromos jellemzők

Leírás	Jel	MIN	TIP	MAX	
Bemenő vonal - magas	VIH	VCC-0.2	-	VCC + 1.0V	VDC
(Φ1,Φ2in)					
Bemenő vonal - magas		VSS + 2.0	-	-	VDC
(RES,P0-P7,IRQ,DATA)					
Bemenő vonal - alacsony	VIL	VSS-0.3	-	VSS + 0.2	VDC
1, 2in ,P0-P7,IRQ,DATA		-	-	VSS + 0.8	VDC
Bemenő átvezetési áram					
(Vin = 0-tól 5.25V,					
VCC = 5.25V)	IIN	-	-	2.5	uA
Logika, 1, 2in		-	-	100	uA
Három állapotú bemenet					
(Vin = 0.4-től 2.4V,					
VCC = 5.25V)	ITSI	-	-	10	uA
Adatvonalak					
Kimenő vonal - magas					
(IOH = 100uA, VCC = 4.75V	VOH	VSS + 2.4	-	-	VDC
DATA,A0-A15,R/W,P0-P7					
Kimenő vonal - alacsony					
IOL = 1.6mA,VCC = 4.75V	VOL	-	-	VSS + 0.4	VDC
DATA, A0-A15, R/W,P0-P7					
Áramfelvétel	ICC	-	125		mA
Kapacitások					
Vin = 0,TA = 25C,f = 1MHz	C				pF
Logika P0-P7	Cin	-	-	10	
DATA		-	-	15	
A0-A15, R/W	Cout	-	-	12	
Φ1	CΦ1	-	30	50	
Φ2	CΦ2	-	50	80	

6510 órajelei

Leírás	Jel	1 MHz			2 MHz			
		MIN	TIP	MAX	MIN	TIP	MAX	
Ciklusidő	TCYC	1000	-	-	500	-	-	ns
Az órajel szélessége Φ1	PWH01	430	-	-	215	-	-	ns
Φ2	PWH02	470	-	-	235	-	-	ns
Átbillenési idők	TF,TR	-	-	25	-	-	15	ns
Az órajelek közti késleltetés	TD	0	-	-	0	-	-	ns

Miután a tápfeszültség elérte a 4,75 V-os feszültségi szintet, a RESET vonalnak még legalább két óraciklusig alacsonynak kell lennie. Ez alatt a R/W vonal érvényessé válik. Ha a RESET vonalon a két óraciklus után egy felfutó él jelenik meg, a mikroprocesszor a fenti RESET-sorozatot hajtja végre.

Megszakítási kérés (IRQ)

Ez a TTL-szintű bemenet kéri a mikroprocesszortól a megszakító ciklus megkezdését. A mikroprocesszor befejezi a kérelem felismerése előtt megkezdett utasítás végrehajtását. Ezután megvizsgálja az állapotregiszterben levő megszakítás engedélyező bitet. Ha ez alacsony, a mikroprocesszor elkezd egy megszakító ciklust. A programszámláló és állapotregiszter tartalmát elmenti a verembe, és letiltja további megszakítások engedélyezését. A ciklus végén a $FFFE és $FFFF címekről betölti az ott található címet, és arra egy feltétel nélküli vezérlésátadást hajt végre.

Cím engedélyezés (AEC)

A címbusz csak akkor érvényes, ha magas. Ha alacsony, akkor nagy impedanciájú állapotban van. Ez a tulajdonsága könnyű DMA-t és multiprocesszoros kezelést tesz lehetővé.

I/O kapu (P0-P7)

Nyolc láb szolgál a mikroprocesszor és a perifériális eszközök közti adatcserére. A kapu a $0001, az adat irány regiszter (DDR) a $0000 címen található.

Írás/olvasás (R/W)

Ezt a jelet a mikroprocesszor azért generálja, hogy ellenőrizze az adatbuszon át lezajló adatforgalmat. Ez a vonal mindig magas, kivéve, ha a mikroprocesszor a memóriába vagy egy perifériális eszközbe ír.

F.14 A 6526 (CIA) chip specifikációja

A 6526 komplex interface adapter (CIA) egy, a 65-ös sorozattal buszkompatibilis, perifériális interface chip rugalmas időzítési és I/O lehetőségekkel.

Jellemzői:

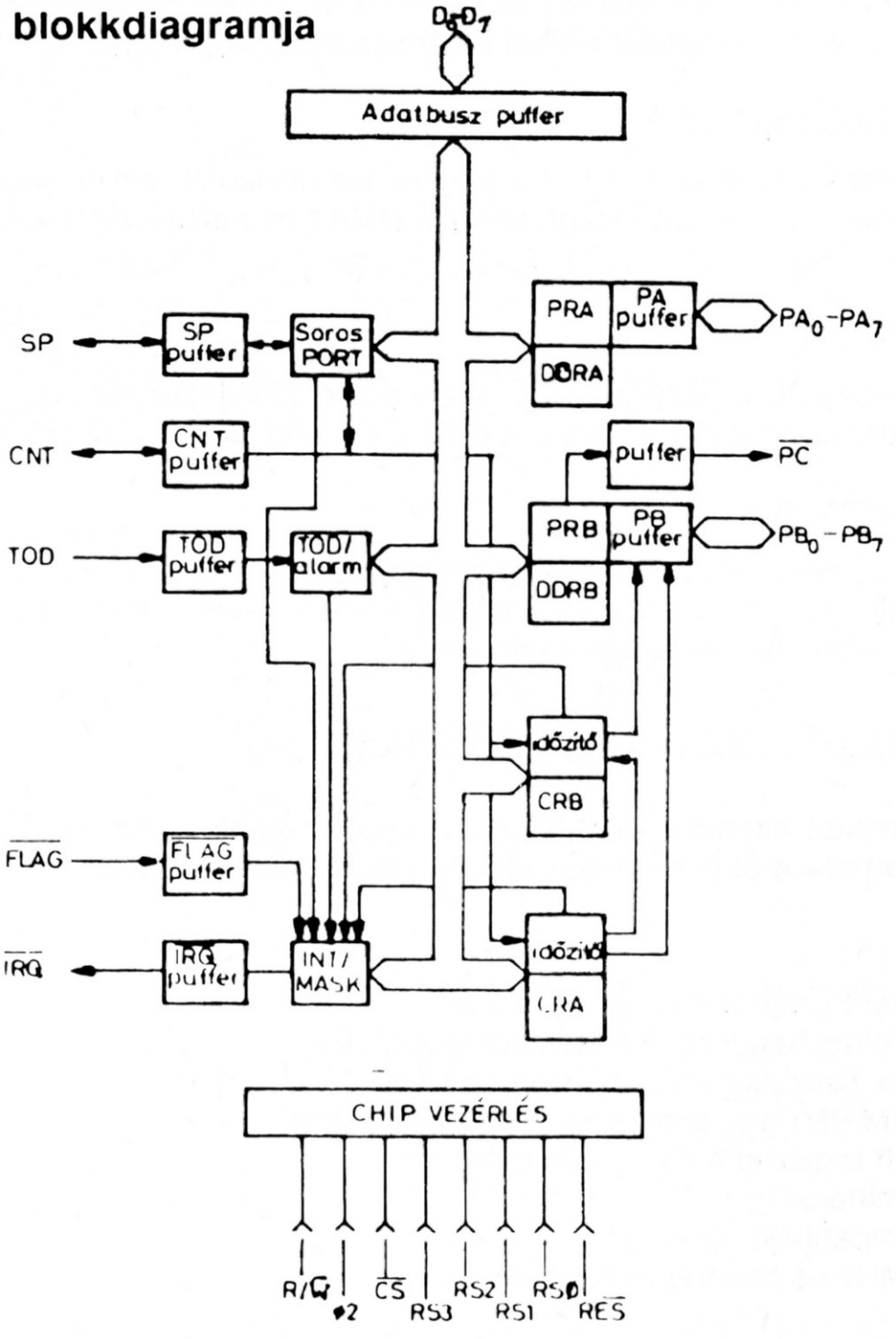
- 16 egyenként programozható I/O-vonal
- 8 vagy 16 bites handshaking (olvasásra vagy írásra)
- 2 független, összeköthető 16 bites belső óra
- 24 órás (AM/PM) óra, programozható ALARM jelzéssel
- 8 bites shift regiszter a soros I/O-porthoz
- 2 TTL terhelhetőség
- CMOS kompatibilis I/O-vonalak
- 1 vagy 2 MHz-es órajellel működtethető.

Elektromos jellemzők

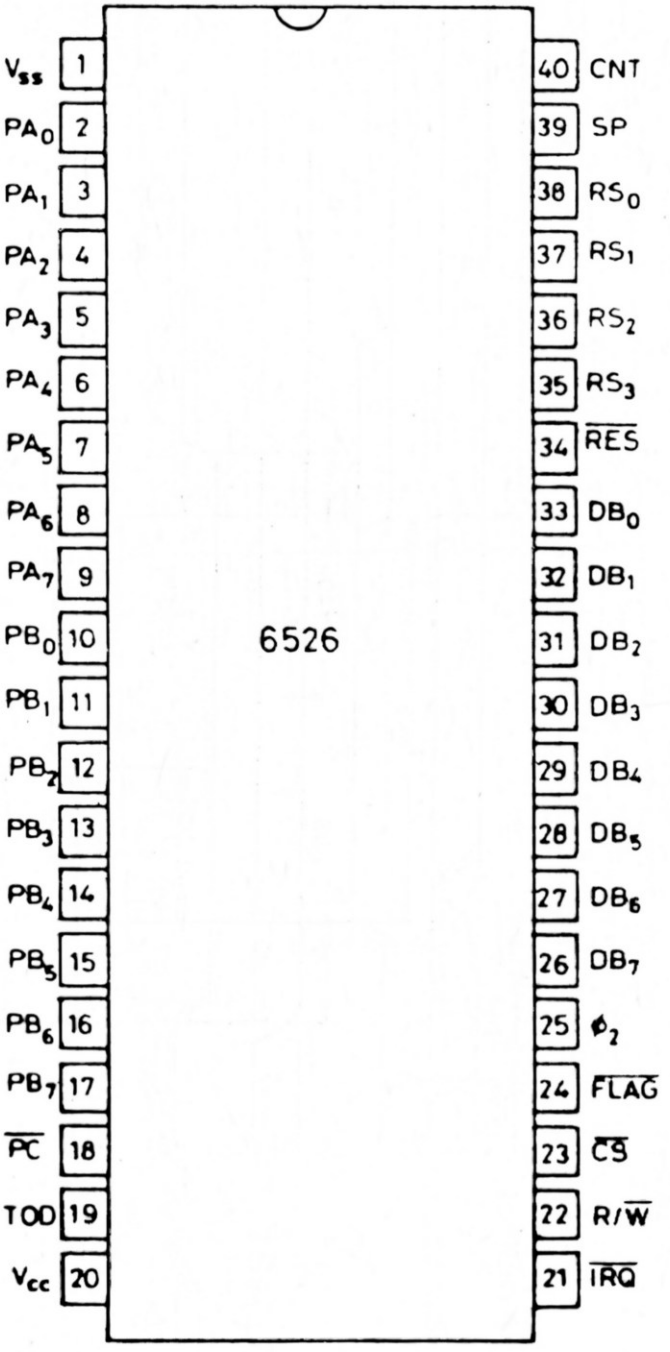
Paraméter	Jel	Érték	Mértékegység
Tápfeszültség	V	-0,3 - +7,0	V
Input/Output feszültség	V	-0,3 - +7,0	V
Működési hőmérséklet	T	0 - +70	C
Tárolási hőmérséklet	T	-55 - +150	C

Valamennyi bemenet tartalmaz védőáramkört a statikus feltöltődések és feszültségek okozta károk megakadályozására. Ügyelni kell azonban, hogy ne alkalmazzunk feleslegesen a megengedettnél nagyobb feszültségeket.

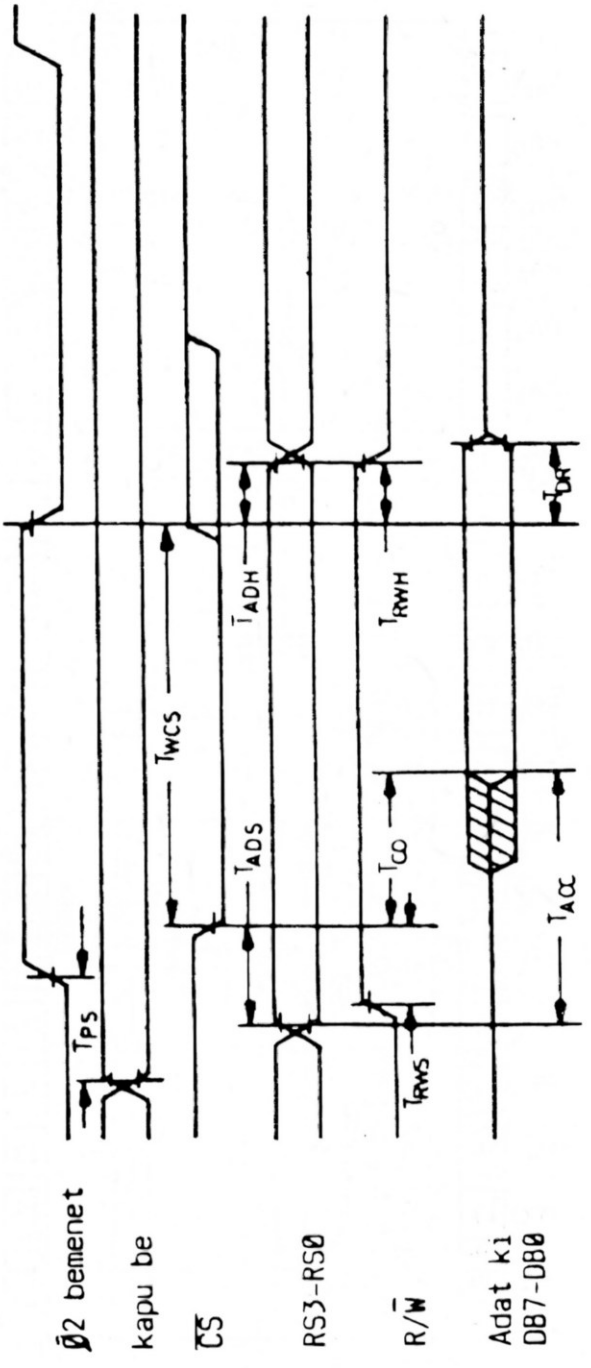
A 6526 chip blokkdiagramja



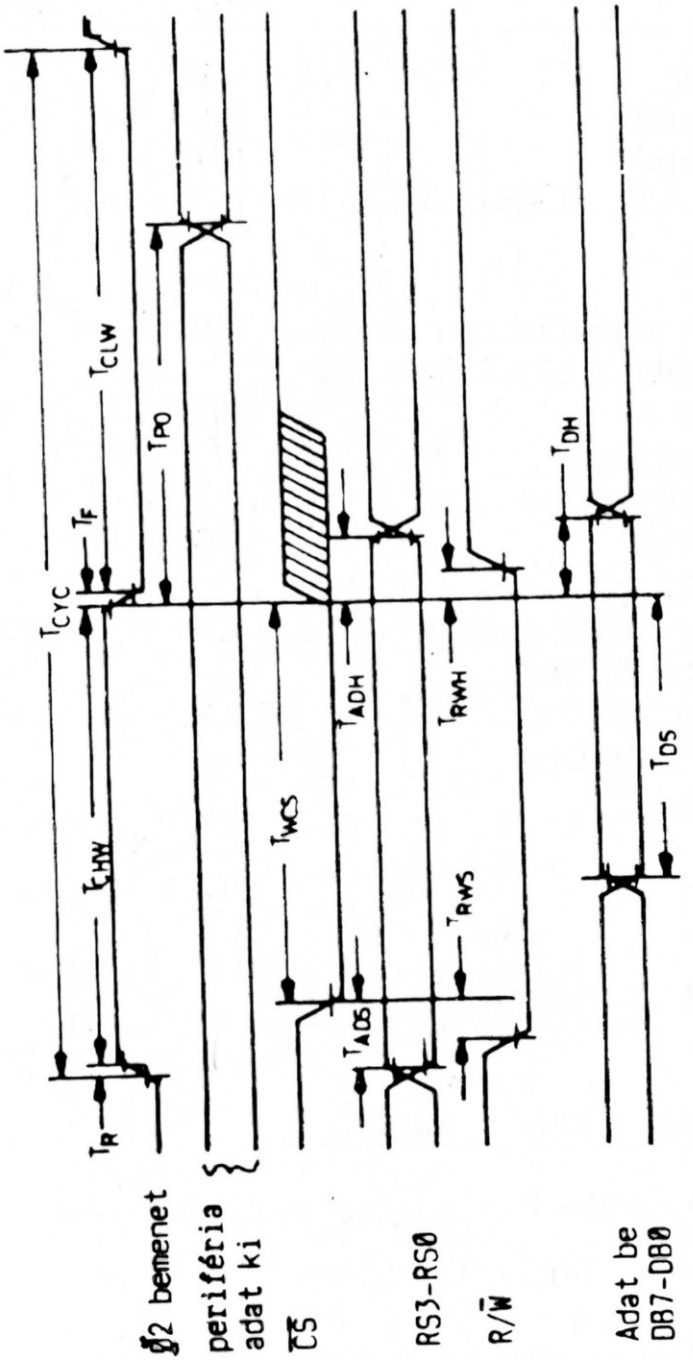
A 6526 chip lábkiosztása



6526 olvasás időzítés



6526 írás időzítés



6526 Elektromos jellemzők

Leírás	Jel	MIN	TIP	Max	
Bemenő vonal - magas	VIH	+2.4	-	VCC	V
Bemenő vonal - alacsony	VIL	-0.3	-	-	V
Bemenő átvezetési áram (Vin = VSS + 5V) TOD, R/W, $\overline{FLAG}$, $\overline{RES}$, RS0-RS3, $\overline{CS}$	IIN	-	1.0	2.5	A
Felhúzó ellenállás	RP1	3.1	5.0	-	k
Kimenő áram a magas impedanciájú állapotban VIN = 4V-tól 2.4V;DB0-DB7,SP,CNT,IRQ	ITSI	-	+1.0	+10.0	A
Kimenő vonal - magas VCC = MIN, ILOAD < -200 a PA0-PA7,PC,PB0-PB7,DB0-DB7	VOH	+2.4	-	VCC	V
Kimenő vonal - alacsony VCC = MIN, ILOAD < 3.2 mA	VOL	-	-	+0.40	V
Kimenő áram VOH > 2.4 V (Sourcing) PA0-PA7,PB0-PB7,PC,DB0-DB7	IOH	-200	-1000	-	A
Kimenő áram (sinking) VOL < 4V PA0-PA7,PC,PB0-PB7,DB0-DB7	IOL	3.2	-	-	mA
Bemenő kapacitás	CIN	-	7	10	pF
Kimenő kapacitás	COUNT	-	7	10	pF
Áramfelvétel	ICC	-	70	100	mA

6526 Írás/olvasás időzítése

Jel	1 MHz		2 MHz		
	MIN	MAX	MIN	MAX	
Írás					
TPD	-	1000	-	500	ns
TWCS	420	-	200	-	ns
TADS	0	-	0	-	ns
TADH	10	-	5	-	ns
TRWS	0	-	0	-	ns
TRWH	0	-	0	-	ns
TDS	150	-	75	-	ns
TDH	0	-	0	-	ns

Jel	1 MHz		2 MHz		
	MIN	MAX	MIN	MAX	
Olvasás					
TPS	300	-	150	-	ns
TWCS/2/	420	-	20	-	ns
TADS	0	-	0	-	ns
TADH	10	-	5	-	ns
TRWS	0	-	0	-	ns
TRWH	0	-	0	-	ns
TACC	-	550	-	275	ns
TCO/3/	-	320	-	150	ns
TDR	50	-	25	-	ns

6526 órajelek

Leírás	Jel	1 MHz		2 MHz		
		MIN	MAX	MIN	MAX	
Ciklusidő	TCYC	1000	20000	500	20000	ns
Átbillenési idő	TR,TF	-	25	-	25	ns
órajel szélessége (magas)	TCHW	410	10000	255	10000	ns
órajel szélessége (alacsony)	TCLW	420	10000	200	10000	ns

Irás/olvasás bemenet (R/$\overline{W}$)

A R/$\overline{W}$ jelet általában a mikroprocesszor adja ki, és a 6526 adatforgalmának az irányát vezérli. A R/$\overline{W}$ magasra állítása olvasást (adat-olvasás a 6526-tól), alacsonyra állítása (adat-írás a 6526-ba) írást jelent.

Címvonalak (RS3-RS0)

Ezek választják ki, hogy melyik regisztert írja/olvassa a mikroprocesszor.

Szinkronizálás

Az adat-átvitel 'handshaking' üzemmódját a $\overline{PC}$ kimenet és a $\overline{FLAG}$ bemenet felhasználásával lehet létrehozni. A $\overline{PC}$ egy ciklusnyi időre alacsony lesz a B kapu olvasását vagy írását követően. Ez a jel használható 'adat kész' vagy 'adat fogadva' jelként is. A 16 bites adatátvitel 'handshaking'-je is lehetséges (felhasználva mind a két kaput), ha először mindig az A kaput olvassuk ki, illetve az A kapuba írunk be. A $\overline{FLAG}$ egy lefutó élre érzékeny bemenet, mely használható egy másik 6526 $\overline{PC}$ kimenetének fogadására vagy általános célú magszakító bemenetként. A $\overline{FLAG}$ bemeneten bármely lefutó él magasra állítja a $\overline{FLAG}$ megszakító bitet.

Adatbusz ki-/bemenetek (DB7-DB0)

Nyolc lábon keresztül történik a 6526 és a rendszer adatbusz közötti adatátvitel. Ezek a lábak nagy impedanciájú állapotban vannak, ha a $\overline{CS}$ vonal nem alacsony valamint az R/$\overline{W}$ és a I2 pedig nem magasak (olvasás). Olvasás alatt az adatbusz-pufferek engedélyezve vannak, hogy a kiválasztott regiszterből a rendszer adatbuszra továbbítsák az adatokat.

Megszakítás-kérés ($\overline{IRQ}$)

Az $\overline{IRQ}$ egy nyitott kollektoros kimenet, mely általában a processzor megszakítási bemenetére csatlakozik. Egy külső ellenállás tartja a jelet magasan, lehetővé téve ezáltal több $\overline{IRQ}$ kimenet összekapcsolását. Az $\overline{IRQ}$ kimenet általában nagy impedanciájú állapotban van, és alacsonyra állítása jelenti a megszakítás kérést.

Reset vonal ($\overline{RES}$)

A $\overline{RES}$ lábon egy alacsony jel törli az összes belső regisztert. A kapu bitjei bemenetként állítódnak be és a kapuk regiszterei nullázódnak (bár a kapu olvasásakor csupa "1"-est kapunk). Az időzítés vezérlő regiszterek nullázódnak és az időzítő zárak magasra állítódnak. Valamennyi további regiszter nullázódik.

F.15 A 6566/67 (VIC II) chip specifikációja

A 7. fejezetben részletesen ismertettük a VIC-II chip működését. Itt csak a ki-/bemeneti vonalakat, s néhány, eddig még nem említett funkcióját írjuk le.

Dinamikus RAM frissítés

A 6566/67 chipbe egy dinamikus RAM frissítés vezérlő van beépítve. Öt 8 bites sor címet frissít minden raszter sorban. Ez a sebesség biztosítja a maximálisan 2.02 ms-os késleltetést egyetlen sor-cím két frissítése között 128 címes frissítési séma esetén. (256 címes frissítési séma esetén a maximális késleltetés 3.66 ms.) Ez a frissítés egyáltalán nem vehető észre a rendszer szempontjából, mivel a frissítés a rendszer óra 1. fázisa alatt lép fel. A 6567 generálja mind a $\overline{RAS}$, mind a $\overline{CAS}$ jelet, amelyek általában közvetlenül a dinamikus memóriához kapcsolódnak. A $\overline{CAS}$ és $\overline{RAS}$ jeleket minden 2. fázisra és minden video adat kéréskor kiadja, így nem szükséges további külső órajel.

Rendszer interface

A 6566/67 chip speciális módon kapcsolódik a rendszer adatbuszhoz. A 65XX rendszernek a rendszerbuszra csak a ciklus 2. fázisa alatt (órajel magas) van szüksége. A 6566/67 chip előnyösen használja ki ezt a tulajdonságot, mivel rendesen az 1. fázis alatt (órajel alacsony) fordul a rendszer memóriához.

Ezért a karakter alakjának olvasása és a memória frissítése egyáltalán nem vehető észre a processzor szempontjából, és nem csökkentik a processzor teljesítőképességét. A video chip szolgáltatja ilyenkor az interface vezérlő jeleket, amelyek a busz kezeléséhez szükségesek. A VIC-II szolgáltatja továbbá az AEC jelet (cím engedélyezés vezérlése), mely letiltja a processzor buszának meghajtó áramköreit, megengedi, hogy a VIC-II hozzáférhessen a címbuszhoz. Az AEC aktív "0" szintű jel, így lehetőség van a 65XX család AEC bemeneteinek közvetlen csatlakoztatására. Az AEC jel rendesen az 1. fázis alatt aktiválódik, így nem befolyásolja a processzor működését. A busz "felosztása" következtében minden memória hozzáfordulást 1/2 ciklus alatt be kell fejezni.

Mivel a video chip egy 1 MHz-es órajelet szolgáltat, (melyet a rendszer 2. fázisaként kell használni), a memória ciklus 500 ns hosszúságú, és magába foglalja a cím dekódolását, az adat elérést és a chipbe való adat beolvasást. Bizonyos 6566/6567 műveletekhez több idő kell, mint ami az 1. fázis alatti olvasásakor rendelkezésre áll. Speciálisan ilyen a karakter mutatók beolvasása a képernyő memóriából és a MOB-adatok olvasása. Ennélfogva a processzort le kell tiltani és folytatni kell az adat beolvasását az órajel 2. fázisa alatt is. Ezt a BA jel (busz felhasználható) segítségével hajtja végre. A BA vonal normálisan magas, de az 1. fázis alatt a video chip egy "0"-val jelzi, hogy szüksége van a 2. fázisra is az adat eléréséhez. A BA vonal "0"

szintjének megjelenése után a processzornak három 2. fázis idő áll rendelkezésére, hogy befejezzen valamely folyamatban levő memóriaműveletet. A BA "0" szintje alatt a negyedik 2. fázisra az AEC jel "0"-ban marad a 2. fázis alatt, mivel a video chip adatot olvas. A BA vonal általában a 65XX processzor RDY bementére van kötve. A karakter mutatók beolvasására minden nyolcadik raszter sor után (a látható területen) kerül sor, és a sor beolvasás negyven egymást követő 2. fázis idejéig tart. A MOB-adatok beolvasásához négy memória hozzáférés szükséges a következőképpen:

Fázis	Adat	Feltétel
1	MOB mutatók	Minden raszter
2	1. MOB byte	A MOB megjelenítése alatt valamennyi raszter
1	2. MOB byte	A MOB megjelenítés alatt valamennyi raszter
2	3. MOB byte	A MOB megjelenítés alatt valamennyi raszter

A MOB mutatót minden raszter sor végén az 1. fázis alatt olvassa be. Ha szükséges, további kiegészítő ciklusokat használ a MOB adatok beolvasására. ilyenkor ismét a 6566/6567 chip szolgáltatja az összes szükséges busz vezérlő jelet.

Memória interface

A video interface chip két típusa, a 6566 és a 6567 a kimenő cím konfigurációban különbözik. A 6566 13 teljesen dekódolt címvonallal rendelkezik, amelyek közvetlenül csatlakoztathatók a rendszer címbuszához. A 6567-nek multiplexelt címvonalai vannak, hogy a 64 Kbyte-os dinamikus RAM memóriához lehessen közvetlenül csatlakoztatni. Az A06-A00 legkisebb helyiértékű címbitek az A06-A00 vonalakon a $\overline{RAS}$ = '0' alatt vannak jelen, míg az A13-A08 legnagyobb helyiértékű bitek az A05-A00 vonalakon a $\overline{CAS}$ = '0' alatt állnak rendelkezésre.

A 6567-nél az A11-A07 lábak statikus cím kimenetek, melyeket közvetlenül a hagyományos 16 Kbyte-nyi (2K*8) ROM memóriához csatolhatunk.

Processzor interface

Eltekintve a fentiekben leírt speciális memória hozzáfordulásoktól, a 6566/6567 regiszterei bármely más periféria chiphez hasonlóan érhetők el. A következő processzor interface jelek biztosítják ezt:

Adatbusz (DB7-DB0)

A nyolc adatbusz láb egy kétirányú kaput alkot, melyeket a $\overline{CS}$, $\overline{RW}$ és a 0. fázis jelek vezérelnek. Az adatbusz csak akkor válik hozzáférhetővé, ha az AEC és a 0. fázis jel "1"-ben és a CS jel "0"-ban áll.

Chip kiválasztás ($\overline{CS}$)

A vonalat alacsonyra állítva, a címbusz és a R/W vonal beállításával hozzáférhetővé válik a chip bármelyik regisztere. A $\overline{CS}$ alacsonyra állításával egyidejűleg az AEC és a 0. fázis jeleket magasra kell állítani, különben $\overline{CS}$ nem érvényes.

Írás/olvasás (R/$\overline{W}$)

Az írást/olvasást engedélyező vonal, a $\overline{CS}$ jellel kiegészítve, határozza meg az adatbuszon az adatforgalom irányát. Ha az R/$\overline{W}$ vonal magas, akkor adatolvasás történik a kiválasztott regiszterből. Ha az R/$\overline{W}$ vonal alacsony, akkor az adatbuszon megjelenő adat betöltődik a kiválasztott regiszterbe.

Címbusz (A05-A00)

Az A5-A0 legkisebb helyiértékű címlábak kétirányúak. A video toknak a processzor által történő írásakor vagy olvasásakor ezek a címlábak bemenetek. A cím bemeneteken levő adatok a regiszterek kiválasztását határozzák meg.

Kimenő órajel (PH0)

Ez a 0. fázis óra egy 1 MHz-es órajelet ad ki, mely a 65xx processzor 0. fázis bemenetére kapcsolódik. Valamennyi rendszerbusz tevékenység ehhez az órajelhez van szinkronizálva. Az óra frekvenciát a 8 MHz-es video bemenő órajel nyolccal való osztásával kapjuk.

Megszakítás kérelem ($\overline{IRQ}$)

Az $\overline{IRQ}$ vonal akkor alacsony, ha egy engedélyezett megszakító forrás megszakítást kér. Az $\overline{IRQ}$ kimenet open drain kimenet, így külső felhúzó ellenállás szükséges.

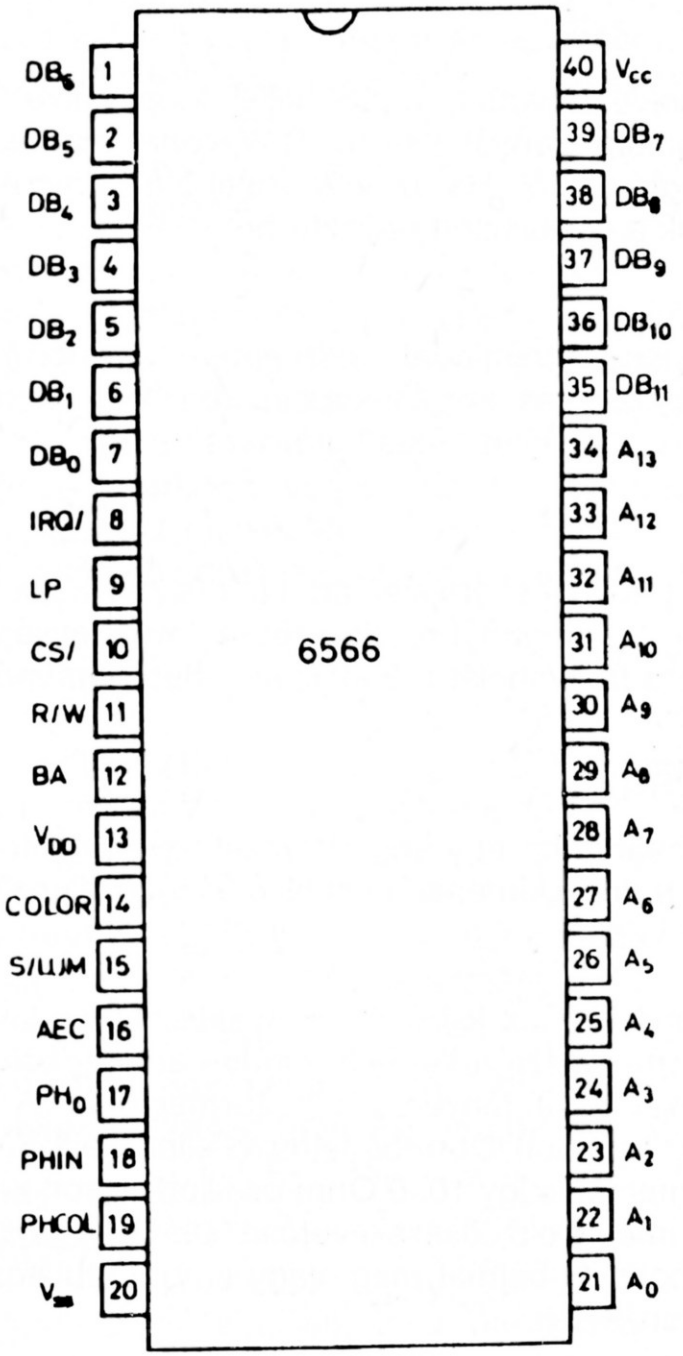
Video Interface

A 6566/6567 video kimenő jele két jelből áll, melyeket chipen kívül kell összekeverni. A SYNC/LUM kimenet tartalmazza a képnek minden adatát, beleértve a vízszintes és függőleges szinkronjeleket és a fényerősség információkat. A SYNC/LUM kimenet open drain kimenet, így egy 500 Ohm-os felhúzó ellenállást kell rákötni. A COLOR kimenet open source kimenet és így 1000 Ohm-os ellenálláson keresztül le kell kötni a földhöz. Ezen két jel megfelelő összekeverése után az eredményül kapott jel közvetlenül egy video monitort hajthat meg, vagy egy modulátoron át hagyományos színes televíziókhoz használható.

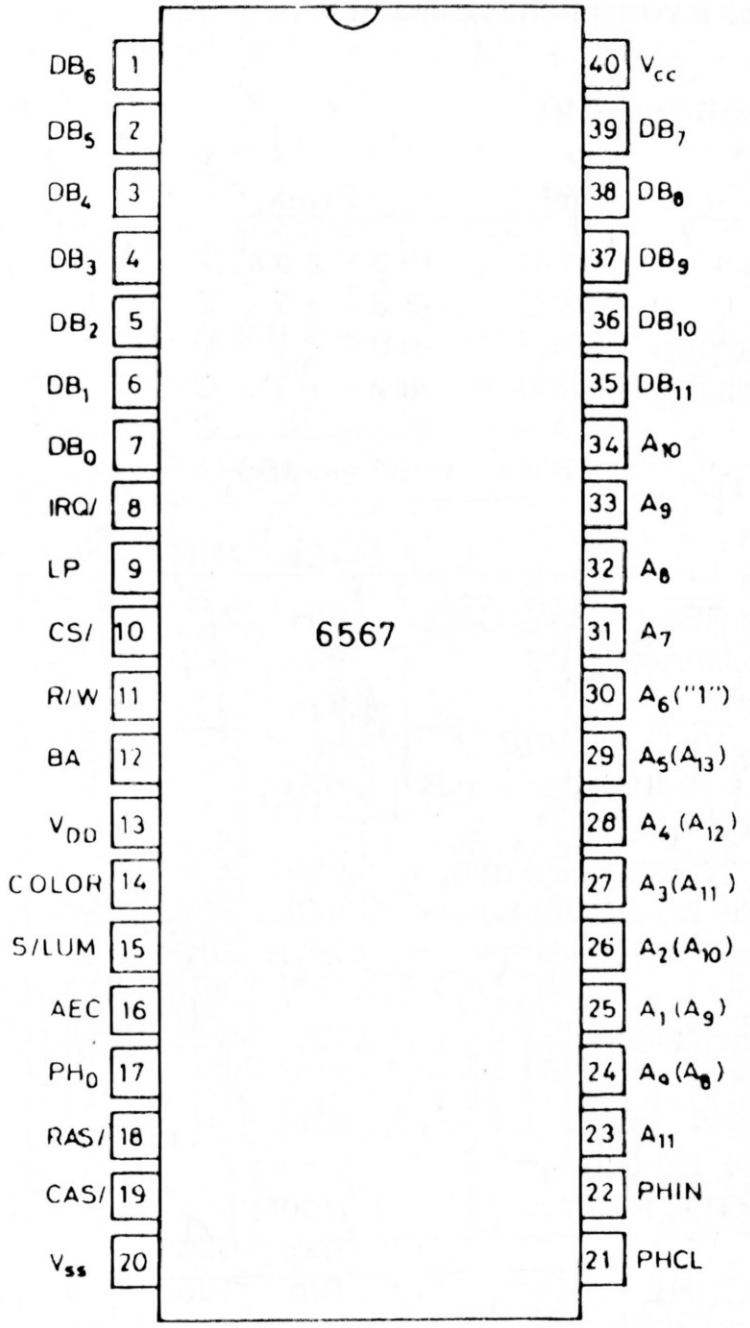
A 6566/6567 busz tevékenységeinek összefoglalása

AEC	PHO	$\overline{CS}$	R/$\overline{W}$	Tevékenység
0	0	X	X	1. fázisbeli olvasás, frissítés
0	1	X	X	2. fázisbeli olvasás (processzor tiltva)
1	0	X	X	Nincs tevékenység
1	1	0	0	Írás a kiválasztott regiszterbe
1	1	0	1	Olvasás a kiválasztott regiszterből
1	1	1	X	Nincs tevékenység

A 6566 chip lábkiosztása



A 6567 chip lábkiosztása



F.16 A 6581 (SID) chip specifikációja

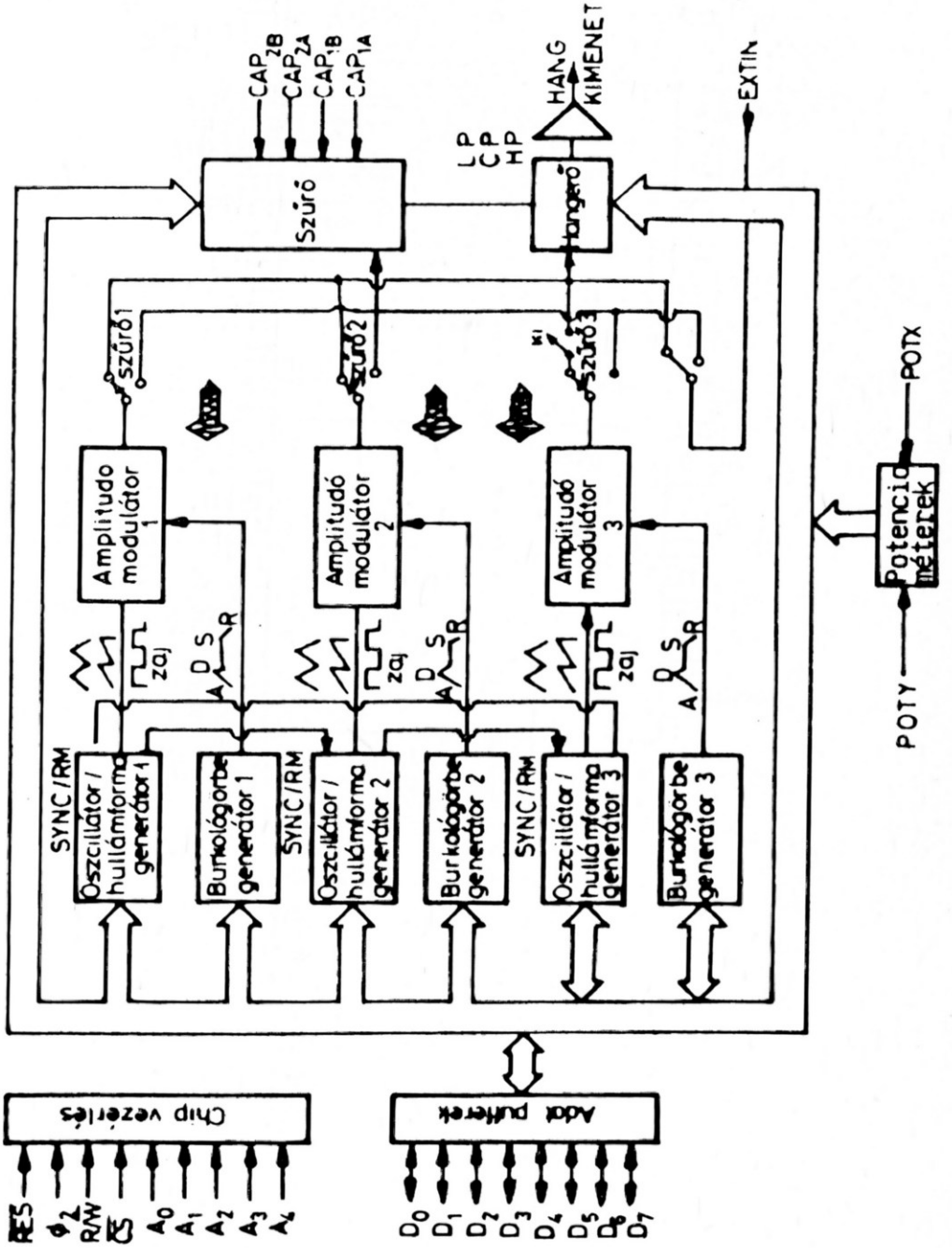
A SID működését a 8. fejezetben részletesen leírtuk. Itt csak a chip lábkiosztását, az időzítési diagramokat és a vonalak használatát ismertetjük.

6581 Elektromos jellemzők

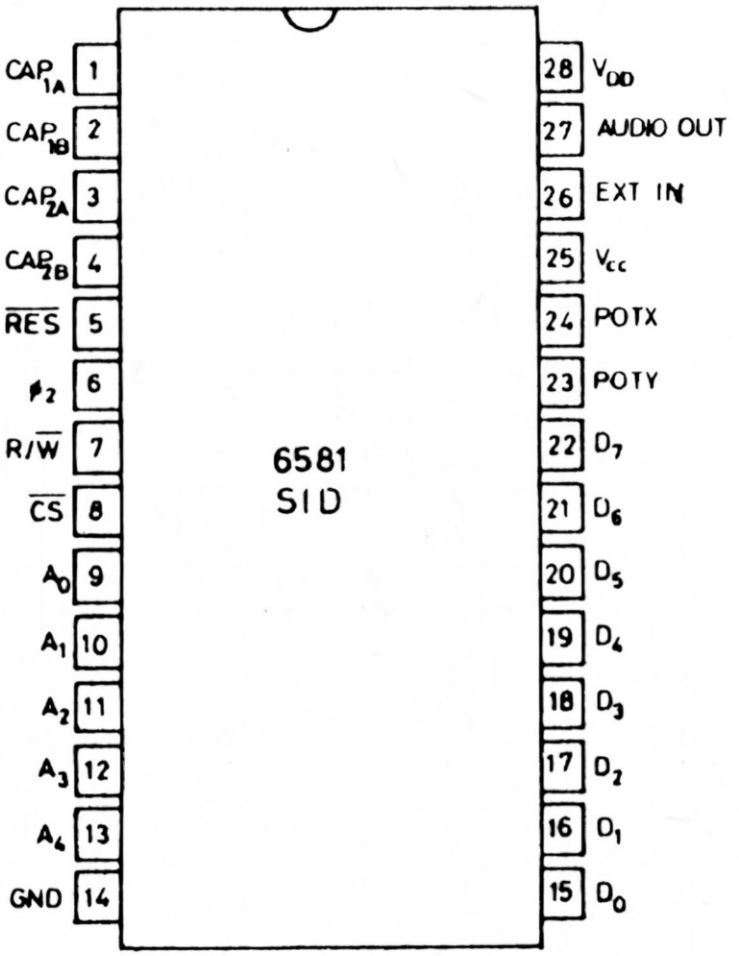
Leírás	Jel	Érték
Tápfeszültség	VDD	-0.3 - + 17 V
Tápfeszültség	VCC	-0.3 - + 7 V
Bemenő feszültség (analóg)	Vina	-0.3 - + 7 V
Bemenő feszültség (digital.)	Vind	-0.3 - + 7 C
Működési hőmérséklet	TA	0 - + 70 C
Tárolási hőmérséklet	TSTG	- 55 - +150 C

Leírás	JEL	MIN	TIP	MAX	
Bemenő vonal - magas $\overline{RES}$,Φ2, R/$\overline{W}$, $\overline{CS}$	VIH	2	-	VCC	VDC
alacsony A0-A4,D0-D7	VIL	-0.3	-	0.8	VDC
Bemenő átvezetési áram	I	-	-	2.5	A
($\overline{RES}$,Φ2,R/$\overline{W}$,$\overline{CS}$,A0-A4) Vin = 0-6 VDC	in				
Három állapotú vonalak (D0-D7,Vcc = max.)	ITSI	-	-	10	A
Bemenő áram Vin = 0.4-2.4 VEC			-		
Kimenő vonal - magas D0-D7, Vcc = min.	VOH	2.4	-	VCC-0.7	VDC
Kimenő vonal - alacsony D0-D7,Vcc = max.	VOL	GND	-	0.4	VDC
Kimenő átvezetési áram (Sourcing)	IOH	200	-	-	A
D0-D7 VOH = 2.4 VDC					
Kimenő átvezetési áram (Sinking)	IOL	3.2	-	-	mA
D0-D7, VOL = 0.4 VDC					
Bemenő kapacitás	Cin	-	-	10	pF
($\overline{RES}$, 2, R/$\overline{W}$,$\overline{CS}$, A0-A4, D0-D7)					
Potméter feszültség, POTX, POTY	Vpot	-	VCC/2	-	VDC
Potméter áram (sink)	Ipot	500	-	-	A
Bemenő ellenállás (EXT IN)	Rin	100	150	-	k
Audio bemenő feszültség (EXT IN)	Vin	5.7	6	6.3	VDC
			0.5	3	VAC
Audio kimenő feszültség					
(AUDIO OUT, 1 k , load = volume = max.	Vout	5.7	6	6.3	VDC
1 oszcillátor		0.4	0.5	0.6	VAC
3 oszcillátor		1.0	1.5	2.0	VAC
Áramfelvétel (VDD)	IDD	-	20	25	mA
Áramfelvétel (VCC)	ICC	-	70	100	mA
Teljesítmény	PD	-	600	1000	mW

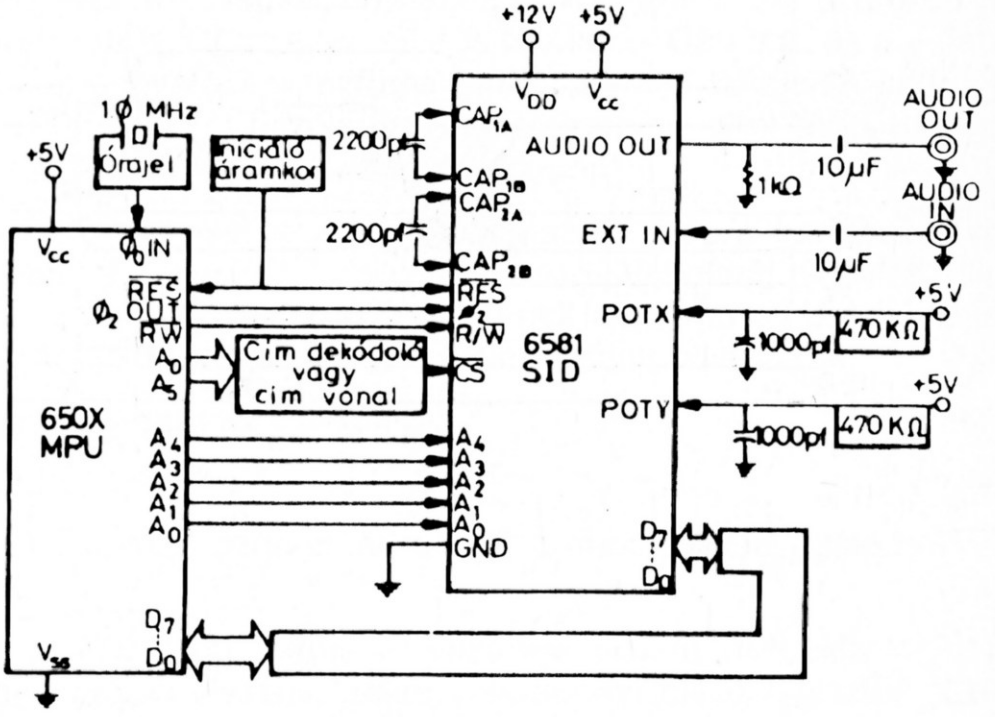
A 6581 chip blokkdiagramja



A 6581 chip lábkiosztása



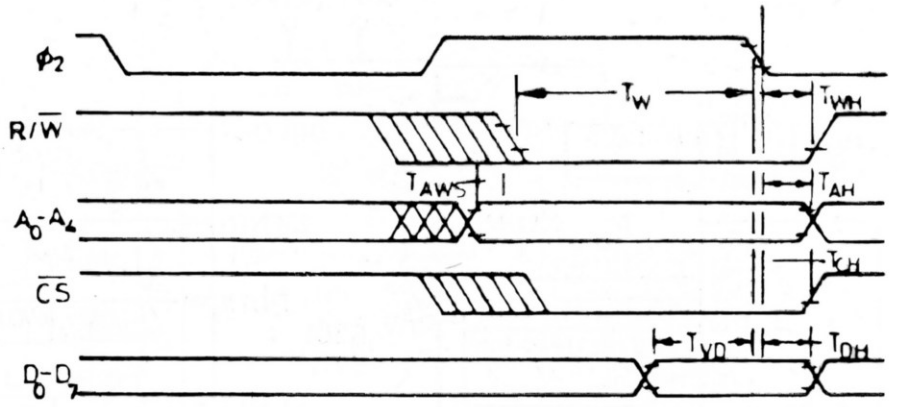
A 6581 egy tipikus alkalmazása

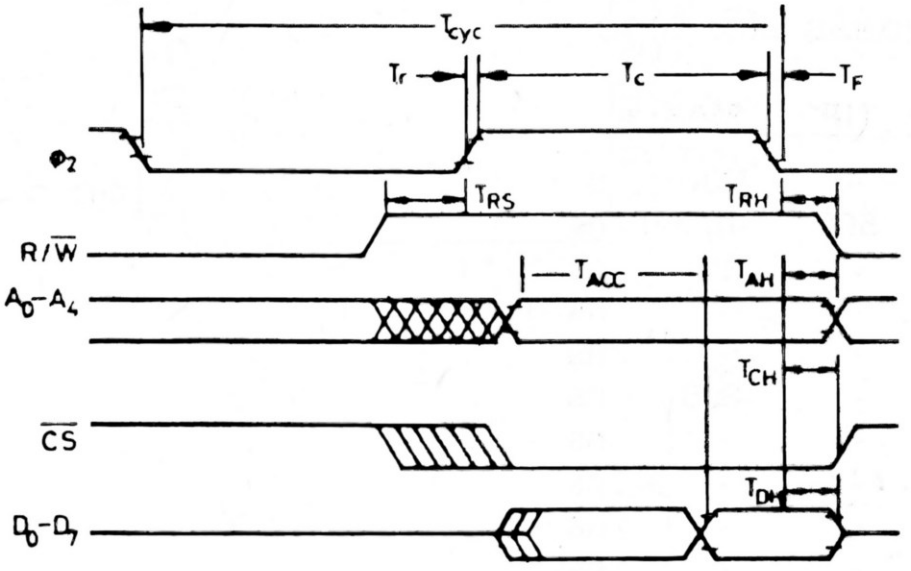


6581 Írás/olvasás időzítése

Jel	MIN	TIP	MAX	
TCYC	1	-	20	u
TC	450	500	10000	ns
TR,TF	-	-	25	ns
TRS	0	-	-	ns
TRH	0	-	-	ns
TACC	-	-	300	ns
TAH	10	-	-	ns
TCH	0	-	-	ns
TDH	20	-	-	ns
TW	300	-	-	ns
TWH	0	-	-	ns
TAWS	0	-	-	ns
TAH	10	-	-	ns
TCH	0	-	-	ns
TVD	80	-	-	ns
TDH	10	-	-	ns

A 6581 chip írás/olvasás diagramja





Kapacitások (CAP1A, CAP1B, CAP2A, CAP2B)

Ezeken a lábakon keresztül csatlakoztathatjuk a programozható szűrőhöz szükséges két integráló kondenzátort. C1-et az 1. és a 2. láb közé, C2-t a 3. és a 4. láb közé kell bekötni. Mindkét kondenzátornak azonos értékűnek kell lennie. A szűrő C1 = C2 = 2200 pF értékű. Komplex, többszólamú rendszerekben, ahol több SID követi egymást, ajánlatos összeillő kondenzátorokat választani.

A szűrő működési tartományát speciális alkalmazásokhoz a kondenzátorok értékének megválasztásával áthelyezhetjük. Például olcsó játékoknál nincs mindig szükség a magas frekvenciákra. Ilyenkor C1-nek és C2-nek nagyobb értéket is választhatunk, hogy a szűrő vezérlését inkább a mély frekvenciák fölött biztosítsuk.

A szűrő maximális levágási frekvenciája:

FCmax = 2.6E-5/A

ahol C a kapacitás értéke. A szűrési tartomány a maximális levágási frekvencia alatti 9 oktávra terjed ki.

RESET (RES) Ez a TTL-szintű bemenet vezérli a SID törlését. Ha legalább tíz Φ2 óraciklus ideig alacsonyba visszük, akkor valamennyi belső regiszter nullázódik, és az audio kimenet törlődik. Ezt a lábat általában a mikroprocesszor reset vonalához vagy egy tápfeszültség bekapcsolásra kapcsoló áramkörhöz csatlakoztatjuk.

Φ2 órajel

Ez a láb a SID TTL-szintű órajel bemenete. Valamennyi oszcillátor-frekvencia és burkoló görbe sebesség ehhez az órajelhez viszonyul. A Φ2 szabályozza a SID és a mikroprocesszor közötti adatforgalmat is. Az adatforgalom csak akkor jöhet létre, ha a Φ2 magas. Lényegében a Φ2 egy felmenő élre érzékeny kiválasztó jelként működik az adatforgalmat illetően. Ezt a lábat általában a rendszer órajelhez csatlakoztathatjuk, melynek a névleges működési frekvenciája 1 MHz.

R/W (7.láb)

Ez a TTL-szintű bemenet vezérli a SID és a mikroprocesszor közötti adatátvitel irányát. Ha a tok ki lett választva, akkor egy '1' szintű jel ezen a lábon lehetővé teszi, hogy a mikroprocesszor adatokat olvasson ki a kiválasztott SID-regiszterből, míg egy '0' szintű jelre a mikroprocesszor adatokat írhat a kiválasztott SID-regiszterbe. Ezt a lábat általában a rendszer R/W vonalhoz csatlakoztatjuk.

$\overline{CS}$ (8.láb)

Ez egy TTL-szintű "0"-ra aktív, chip kiválasztó bemenet, amely a SID és a mikroprocesszor közötti adatátvitelt vezérli. Minden adatátvitelnél a $\overline{CS}$ lábnak "0"-ban kell lennie. A kiválasztott SID-regiszterből csak akkor tudunk adatot kiolvasni, ha $\overline{CS}=0$, $\Phi 2=1$ és $R/\overline{W}=1$. A kiválasztott SID-regiszterbe csak akkor tudunk adatokat beírni, ha $\overline{CS}=0$, $\Phi 2=1$ és $R/\overline{W}=0$.
Ezt a lábat általában egy cím dekódoló áramkörhöz kapcsoljuk, és így lehetőség nyílik, hogy a SID-et a rendszer memóriájaként címezzük.

A0-A4 (9-13.lábak)

Ezen TTL-szintű bemenetek segítségével választhatunk ki a lehetséges 29 SID-regiszterből egyet. Ugyan 32 regisztert címezhetnénk meg ily módon, de a fennmaradó 3 regiszterhely nincs felhasználva. Erre a 3 regiszterhelyre történő írást nem veszi figyelembe a chip, míg a 3 regiszter olvasásakor érvénytelen adatok kerülnek a buszra. Ezek a lábak általában a mikroprocesszor megfelelő címvonalaihoz kapcsolódnak, így a SID-regiszterek a memóriákkal azonos módon címezhetők.

GND (14.láb)

A legtökéletesebb megoldás érdekében a SID GND (föld) lábát le kell választanunk a többi digitális áramkör GND (föld) lábairól. Így minimalizálhatjuk az audio kimeneten a digitális zajt.

D0-D7 (15.-22.lábak)

Ezeken a kétirányú vonalakon keresztül lehet a SID és mikroprocesszor közötti adatforgalmat lebonyolítani. Bemenetként TTLkompatibilisek és kimenetként 2 TTL terhelés meghajtására képesek. Az adatpufferek általában a nagy impedanciájú harmadik állapotban vannak. Íráskor az adatpufferek input állapotban lesznek és a mikroprocesszor által kiadott adatok ezeken a vonalakon keresztül kerülnek a SID-be. Olvasáskor az adatpufferek irányítása megváltozik és a SID adatai ezeken a vonalakon át a mikroprocesszorba jutnak. Ezeket a lábakat rendszerint a mikroprocesszor megfelelő adatvonalaival kötjük össze.

POT X, POT Y (24., 23.láb)

Ezek a lábak A/D átalakítók bemenetei, melyek a külső potenciométerek állását digitalizálják. A konverziós folyamat a POT láb és a föld közé kötött kondenzátor időállandójától függ, melyet a POT láb és a +5V közé kötött potenciométeren keresztül töltünk fel. A komponensek értékeit a következőképpen határozhatjuk meg:

RC = 4.7E-4

ahol R a potenciométer maximális ellenállás értéke és C a kondenzátor kapacitása.

Minél nagyobb a kapacitás, annál kisebb a potenciométer állítási lehetősége. Az ajánlott értékek: R = 470 kOhm és C = 1000 pF. Jegyezzük meg, hogy önálló potenciométer és kondenzátor kell minden egyes POT lábhoz.

Vcc, Vdd (25. és 28. lábak)

Mint a GND lábnál, itt is különválasztott +5VDC (+12VDC) vonalat kell a tápegységtől a SID Vcc (Vdd) lábához vezetni, hogy minimalizáljuk a zajt. Emellett egy zavarszűrő kondenzátort kell a lábhoz közel elhelyezni.

EXT IN (26.láb)

Ezen az analóg bemeneten keresztül lehet külső audio jeleket a SID audio kimenetével keverni vagy azokat a szűrőn átvezetni. Tipikus külső források: az ének-, gitár- és orgonahang. A láb bemenő impedanciája 100 kOhm nagyságrendű.

Az ehhez a lábhoz csatolt jelek egyenfeszültségű komponense 6V legyen és hullámossága ne haladja meg a 3V p-p-t. Azért, hogy megakadályozzuk a DC-szintek közötti különbségekből adódó interferenciát, a külső jeleket AC-csatolással egy 1-10 μF kapacitású kondenzátoron keresztül kell az EXT IN bemenethez kapcsolni. Mivel az egyenes audio ágnak (FILTEX = 0) egységnyi az erősítése, az EXT IN bemeneteket felhasználhatjuk több SID chip kimeneteinek keverésére, daisy-chainben (soros láncban) összekötve őket. Az ilymódon összeláncolható chipek számát a végső audio kimeneten megengedhető zaj és torzítás nagysága határozza meg. Jegyezzük meg, hogy a kimenő hangerő szabályozás nemcsak a SID három hangösszetevőjét, hanem a külső bemenő jelet is befolyásolja.

AUDIO OUT (27.láb)

Ez az open source puffer a SID végső audio kimenetét adja, mely magába foglalja a SID három hangösszetevőjét, a szűrő kimenő jelét és a külső bemenő jelét. A kimenő jel szintjét a hangerő szabályozás állítja be, egyenfeszültségű komponense 6V, hullámossága 2V p-p-ig terjed. A megfelelő működés érdekében a kimenet és a föld közé egy ellenállást kell kötni. Szabványos kimeneti impedancia esetén az ellenállás ajánlott értéke 1 kOhm.

Mivel a SID kimenete 6V DC-szint körül mozog, azt AC-csatolással egy 1-10μF kapacitású kondenzátoron keresztül kell az ezt követő audio erősítőhöz kapcsolni.

F.17 Nyomtatók vezérlő karakterei

GP-100VC grafikus nyomtató

Gyártó: Seikosha Co.,Ltd.

Nyomtatási mód:	6*7-es pontmátrix
Karakter méret:	magasság: 2.82mm
	szélesség: 2.11mm
Karakter sűrűség:	10 karakter/inch
Nyomtatási sebesség:	30 karakter/másodperc
Sorszélesség:	maximum 80 oszlop
Sorsűrűség:	karakteres mód: 5 sor/inch
	grafikus mód: 7.5 sor/inch
Karakterkészlet:	teljes C-64
Illesztés:	soros busz
Hardver szám:	4 vagy 5 (kapcsolóval állítható)
Nyomtatási módok:	karakteres mód
	grafikus mód

A GP-100VC a C-64-hez talán leggyakrabban használt nyomtató. A Commodore kompatibilis nyomtatók közé tartozik azzal az egyetlen megszorítással, hogy programozható karaktert nem használhatunk. A grafikus, illetve karakteres üzemmódban mind a soremelés nagysága, mind a nyomtatás sebessége eltér. A grafikus és karakteres üzemmód egy soron belül váltakozva használható.

A nyomtató hátán található kapcsolóval állíthatjuk be a periféria számot: 4, vagy 5 lehet. A kapcsolónak van egy 'T' állása is. Ha erre állítjuk be, akkor bekapcsolás után teszt üzemmódban kezd el dolgozni, és kikapcsolásig újra és újra kiírja a teljes karakterkészletet.

A nyomtató megnyitása

A nyomtató a 0, illetve 7 megnyitási módokat értelmezi. Az első esetben a grafikák/nagy betűk, a második esetben a kis betűk/nagy betűk karakterkészlettel kezd el dolgozni. A megnyitási módtól **függetlenül**, nyomtatás közben is lehet a karakterkészletet váltani. Ennek megfelelően egyetlen sorban mindkét karakterkészlet karaktereit kiírathatjuk, ami a képernyőn lehetetlen.

Vezérlő karakterek

NL CHR$(10) kocsivissza soremelés

CR CHR$(13) kocsivissza soremelés

A nyomtatás egészen addig nem történik meg, míg meg nem telik a puffer, vagy egy CHR$(10) vagy CHR$(13) karaktert ki nem nyomtatunk. A puffer megtelésekor a puffer egy sornyi része kiíródik. A pufferben levő többi adat azonban nem vész el.

SO CHR$(14) dupla széles karaktermód

A karakter kiírása után a nyomtató dupla széles karaktereket nyomtat, egészen egy CHR$(15), CHR$(13) vagy CHR$(10) kiírásáig.

CD CHR$(17) kis betűk/nagy betűk karakterkészlet kiválasztása

CU CHR$(145) grafikák/nagy betűk karakterkészlet kiválasztása

RON CHR$(18) inverz karakterek kiírása

ROF CHR$(146) normál (nem inverz) karakterek nyomtatása

BS CHR$(8) grafikus üzemmód

A CHR$(8) karakter kinyomtatása után következő byte-okat a nyomtató egy raszteroszlop adatainak tekinti, s az annak megfelelő alakzatot nyomtatja ki. A nyomtatónak 7 tűje van, ennek megfelelően az egyes tűknek a következő bitek felelnek meg:

byte	érték	alak
0	1	* *
1	2	* *
2	4	*
3	8	***
4	16	*
5	32	* *
6	64	* *

A 7. bitnek mindig magasnak kell lennie. Ennek megfelelően a fenti alakzat kinyomtatásához - grafikus üzemmódban - a következő három byte-ot kell kiíratni:

1. oszlop: 1 + 2 + 8 + 32 + 64 + 128 = 235
2. oszlop: 4 + 8 + 16 + 128 = 156
3. oszlop: 1 + 2 + 8 + 32 + 64 + 128 = 235

A fenti három raszteroszlopot így a következő utasítással írathatjuk ki:

```
PRINT#4,CHR$(8),CHR$(235)+CHR$(156)+CHR$(235);
```

SUB CHR$(26) grafikus adat ismétlése

Lehetőség van grafikus üzemmódban egy raszteroszlop többszöri ismétlésére. Ennek alakja a következő:

```
PRINT#4,...CHR$(8);...CHR$(26);CHR$(DB);CHR$(GRF);...
```

ahol GRF az ismétlendő grafikus adat, DB pedig az ismétlések száma. Egy grafikus adatot maximum 255-ször lehet megismételni.

SI CHR$(15) megszünteti a grafikus üzemmódot.

POS CHR$(16) tabulálás

A nyomtatást tetszőleges raszteroszlopban elkezdhetjük. (A képernyőn csak karakterhelyre pozicionálhatunk.) A tabulálás alakja:

```
PRINT#4,...CHR$(27);CHR$(16);CHR$(HP);CHR$(LP);...
```

A tabulálás helyét a P = HP*256 + LP képlet segítségével számítja ki a nyomtató. Ennek értéke nem haladhatja meg a 479-et. Igy HP vagy 0, vagy 1 lehet.

MPS801 mátrix nyomtató

Gyártó: Commodore Business Machines, Inc.

Nyomtatási mód:	6*7-es pontmátrix
Karakter méret:	magasság: 2.82mm
	szélesség: 2.11mm
Karakter sűrűség:	10 karakter/inch
Nyomtatási sebesség:	50 karakter/masodperc
Sorszélesség: maximum	80 oszlop
Sorsűrűség:	karakteres mód: 5 sor/inch
	grafikus mód: 7.5 sor/inch
Karakterkészlet:	teljes C-64
Illesztés:	soros busz
Hardver szám:	4 vagy 5 (kapcsolóval állitható)
Nyomtatási módok:	karakteres mód
	grafikus mód

Az MPS801 a Sheikosha GP-100VC-hez igen sokban hasonlít. A Commodore kompatibilis nyomtatók közé tartozik. A grafikus, illetve karakteres üzemmódban mind a soremelés nagysága, mind a nyomtatás sebessége eltér. A grafikus és karakteres üzemmód egy soron belül váltakozva használható.

Jelenleg szinte kizárólag az MPS801-es nyomtatóval ekvivalens MPS803-as nyomtató vásárolható.

A nyomtató hátán található kapcsolóval állíthatjuk be a periféria számot; ez vagy 4, vagy 5. A kapcsolónak van egy 'T' állása is. Ha erre állítjuk be, akkor bekapcsolás után teszt üzemmódban kezd el dolgozni, és kikapcsolásig újra és újra kiírja a teljes karakterkészletet.

A nyomtató megnyitása

A nyomtató a következő megnyitási módokat értelmezi:

SA = 0: nagy betűk/grafikák karakterkészlettel kezd el dolgozni
SA = 7: kis betűk/nagy betűk karakterkészlettel kezd el dolgozni

Megjegyezzük, hogy a megnyitási módtól **függetlenül**, nyomtatás közben is lehet a karakterkészletet, vagy a nyomtatási módot váltani. Ennek megfelelően egyetlen sorban mindkét karakterkészlet karaktereit kiírathatjuk, ami a képernyőn lehetetlen.

Vezérlő karakterek

NL CHR$(10) kocsivissza soremelés

CR CHR$(13) kocsivissza soremelés

A nyomtatás egészen addig nem történik meg, míg meg nem telik a puffer, vagy egy CHR$(10) vagy CHR$(13) karaktert ki nem nyomtatunk. A puffer megtelésekor a puffer egy sornyi része kiíródik. A pufferben levő többi adat azonban nem vész el.

SO CHR$(14) dupla széles karaktermód

A karakter kiírása után a nyomtató dupla széles karaktereket nyomtat, egészen egy CHR$(15), CHR$(13) vagy CHR$(10) kiírásáig.

CD CHR$(17) kis betűk/nagy betűk karakterkészlet kiválasztása

CU CHR$(145) grafikák/nagy betűk karakterkészlet kiválasztása

RON CHR$(18) inverz karakterek kiírása

ROF CHR$(146) normál (nem inverz) karakterek nyomtatása

BS CHR$(8) grafikus üzemmód

A CHR$(8) karakter kinyomtatása után következő byte-okat a nyomtató egy raszteroszlop adatainak tekinti, s az annak megfelelő alakzatot nyomtatja ki. A nyomtatónak 7 tűje van, ennek megfelelően az egyes tűknek a következő bitek felelnek meg:

byte	érték	alak
0	1	* *
1	2	* *
2	4	*
3	8	***
4	16	*
5	32	* *
6	64	* *

A 7. bitnek mindig magasnak kell lennie. Ennek megfelelően a fenti alakzat kinyomtatásához – grafikus üzemmódban – a következő három byte-ot kell kiíratni:

1. oszlop: 1 + 2 + 8 + 32 + 64 + 128 = 235
2. oszlop: 4 + 8 + 16 + 128 = 156
3. oszlop: 1 + 2 + 8 + 32 + 64 + 128 = 235

A fenti három raszteroszlopot így a következő utasítással írathatjuk ki:

```
PRINT#4,CHR$(8),CHR$(235)+CHR$(156)+CHR$(235);
```

SUB CHR$(26) grafikus adat ismétlése

Lehetőség van grafikus üzemmódban egy raszteroszlop többszöri ismétlésére. Ennek alakja a következő:

```
PRINT#4,...CHR$(8);...CHR$(26);CHR$(DB);CHR$(GRF);...
```

ahol GRF az ismétlendő grafikus adat, DB pedig az ismétlések száma. Egy grafikus adatot maximum 255-ször lehet megismételni.

SI CHR$(15) megszünteti a grafikus üzemmódot.

POS CHR$(16) tabulálás

A nyomtatást tetszőleges raszteroszlopban elkezdhetjük. (A képernyőn csak karakterhelyre pozícionálhatunk.) A tabulálás alakja:

```
PRINT#4,...CHR$(27);CHR$(16);CHR$(HP);CHR$(LP);...
```

A tabulálás helyét a P = HP*256 + LP képlet segítségével számitja ki a nyomtató. Ennek értéke nem haladhatja meg a 479-et. Igy HP vagy 0, vagy 1 lehet.

MPS802/1526 kétirányú mátrix nyomtató

Gyártó: Commodore Business Machines, Inc.

Nyomtatási mód:	8*8-as pontmátrix
Karakter méret:	magasság: 2.388mm
	szélesség: 2.032mm
Karakter sűrűség:	10 karakter/inch
Nyomtatási sebesség:	50 karakter/masodperc
Sorszélesség:	maximum 80 oszlop
Sorsűrűség:	programozható
Karakterkészlet:	teljes C-64
Illesztés:	soros busz
Hardver szám:	4 vagy 5 (kapcsolóval állítható)
Nyomtatási módok:	karakteres mód

Az MPS802 a neve és a száma ellenére **nem** kompatibilis felülről az MPS801-es nyomtatóval. Két lényeges különbség, hogy az MPS802-es nyomtatónak **csak karakteres** üzemmódja van. Másrészről lehetővé teszi számok **formázott kiírását**. A Commodore kompatibilis nyomtatók közé tartozik.

A nyomtató hátán található kapcsolóval állíthatjuk be a periféria számot; ez vagy 4, vagy 5. Ha bekapcsolás közben benyomva tartjuk a 'Paper ADVANCE' feliratú papírtovábbító gombot, akkor teszt üzemmódban kezd el dolgozni, és kikapcsolásig újra és újra kiírja a teljes karakterkészletet.

A nyomtató megnyitása

A nyomtató a következő megnyitási módokat értelmezi:

SA = 0: nagy betűk/grafikák karakterkészlettel kezd el nyomtatni.

SA = 1: az ezzel a móddal megnyitott csatornán nyomtatott adatokat megformázva nyomtatja ki.

SA = 2: az 1-es megnyitási módhoz tartozó csatornán kiírt adatok formátumát határozza meg. Az ebbe a csatornaba kiírt egyes karakterek jelentése a következő:

9 = egy számjegyet jelent. Ha nincs erre a pozicióra eső számjegy, akkor szóköz íródik ki.
Z = egy számjegyet jelent. Ha nincs, egy 0 íródik ki a helyére.
$ = a szám elé egy $ jelet kell írni. Ha valamennyi számjegyet $-al definiáljuk, akkor a $ jel és a szám közti esetleges szóközök nem íródnak ki.
. = a tizedespont helyét definiálja
S = ha egy numerikus mező előtt használjuk, akkor a pozitív előjel '+'-ként íródik ki.

- = egy numerikus mező végén azt jelenti, hogy a negatív előjelet a szám végére kell kiírni.
A = egy alfanumerikus karaktert jelent. Az alfanumerikus jeleket balra igazítva, a vezető szóközöket elhagyva, jobbról esetleg szóközökkel feltöltve írja ki. Annyi A-t kell egy mezőben megadni, ahány karaktert akarunk kiíratni. A be nem férő karakterek elvesznek.

Lehetőség van a formátummezőben olyan karakterek elhelyezésére, amelyek változatlan formában kiíródnak. Ezek elé egy <RVS ON> karaktert kell kinyomtatni.

SA = 4 engedélyezi a hibaüzenetek kiírását. Ezek a hibaüzenetek a következők:

PE:L - az egy oldalra kiírható sorok számának a (13,128) intervallumba kell esnie. Ettől eltérő értéket adtunk meg az SA=3 móddal megnyitott csatornán. Az előző sor/lap érték marad érvényben.
PE:C - egy nem megengedett megnyitási módot használtunk. A parancs figyelmen kívül marad.
PE:M - Az adatformátum nem megfelelő. A nyomtató egy numerikus adat helyett mást kapott. A hibaüzenet után az első ilyen karakter is kiírásra kerül.
PE:E - Exponens hiba. A számadatok egy + vagy - előjellel bevezetett két jegyből álló exponenst tartalmazhatnak csak.
PE:F - Rossz formátum. Az SA=2 móddal megnyitott csatornán egy nem megengedett karaktert küldtünk el, vagy a formátum szintaxisa nem megfelelő.
PE:T - Terminátor hiba. Különböző megnyitási módú csatornákat csak egy CHR$(13) vagy CHR$(10) után lehet cserélni.

SA= 6: sorok számának megadása. 216 lépés van inchenként. Azt kell megadni, hogy egy sor (beleértve a hozzá tartozó soremelést is) hány lépésből álljon. Így például 27 lépés 8 sor/inchnek felel meg, míg 32 lépés 6 sor/inchnek. Az ezzel a megnyitási móddal megnyitott csatornán elküldött byte definiálja a lépésszámot, s az határozza meg, hogy mennyi sor íródjék ki inchenként.

SA= 7: kis betűk/nagy betűk karakterkészlettel kezd el dolgozni

SA= 9: letiltja a hibaüzenetek nyomtatását

SA=10: inicializálja a nyomtatót

Megjegyezzük, hogy a megnyitási módtól **függetlenül**, nyomtatás közben is lehet a karakterkészletet, vagy a nyomtatási módot váltani. Ennek megfelelően egyetlen sorban mindkét karakterkészlet karaktereit kiírathatjuk, ami a képernyőn lehetetlen.

Vezérlő karakterek

NL CHR$(10) soremelés

CR CHR$(13) kocsivissza soremelés

A nyomtatás egészen addig nem történik meg, míg meg nem telik a puffer, vagy egy CHR$(10) vagy CHR$(13) karaktert ki nem nyomtatunk. A puffer megtelésekor a puffer egy sornyi része kiíródik. A pufferben levő többi adat azonban nem vész el.

CRN CHR$(141) kocsivissza soremelés nélkül

SO CHR$(14) dupla széles karaktermód

A karakter kiírása után a nyomtató dupla széles karaktereket nyomtat, egészen egy CHR$(15), CHR$(13) vagy CHR$(10) kiírásáig.

CD CHR$(17) kis betűk/nagy betűk karakterkészlet kiválasztása

CU CHR$(145) grafikák/nagy betűk karakterkészlet kiválasztása

RON CHR$(18) inverz karakterek kiírása

ROF CHR$(146) normál (nem inverz) karakterek nyomtatása

PCH CHR$(254) programozható karakter kiírása

Lehetőség van egyetlen karakter alakjának a definiálására. Ezt a karaktert a fenti kóddal írathatjuk ki. Bekapcsoláskor egy szóközt nyomtat. A karakter alakját az 5-ös megnyitási móddal megnyitott csatornán kell elküldeni. Az ezen a csatornán kinyomtatott byte-ok a karakter egy-egy oszlopának felelnek meg. A programozható karakter alakjának megadásához ezen a csatornán 8 byte-ot kell elküldeni. Az egyes bitek egy-egy raszterpontnak felelnek meg, méghozzá fordított sorrendben. A 7. bit a legfelső, a 0. a legalsó raszterpontnak felel meg. A bit magasra állítása jelenti a nyomtatást.

PON CHR$(147) lapozás bekapcsolása

A lapozás bekapcsolásakor a 12 inches papírra a nyomtató csak 66 sort ír. A lap elején és végén 3-3 üres sort ír. A sorszám az SA = 3 móddal megnyitott csatorna segítségével változtatható. **Lapdobást** a CHR$(12) kiírásával érhetünk el.

POF CHR$(19) lapozás kikapcsolása

A sorok írása folyamatos. Ha előzőleg bekapcsoltuk a lapozást, akkor ez még egy lapemelést is eredményez.

F.18 Lemezformátumok

Ez a függelék a 1541, a 1570/71 és a 1581 típusú lemezegységek formátumát írja le, azaz azt, hogy az egyes katalógusbejegyzéseknek mi a szerepe, s hogyan is épülnek fel az egyes file-ok.

A lemez leglényegesebb része a BAM, amelyik az egyes blokkok foglaltságát jelzi. Ezek formátuma a 1541/1570/1581 esetén eltér.

BAM 1541 (18. sáv, 0. szektor)

Byte	Jelentés
0	A következő katalógus-blokk sávja (mindig 18)
1	A következő katalógus-blokk szektora (mindig 1)
2	65, ez jelzi a 1541/1570/1571/4040 formátumot
3	nem használt
4	az 1. sávon szabad blokkok száma
5	az 1. sáv 0.-7. szektorainak a foglaltsága
6	az 1. sáv 8.-16. szektorainak a foglaltsága
7	az 1. sáv 17.-23. szektorainak a foglaltsága
.....	
140	a 35. sávon szabad blokkok száma
141	a 35. sáv 0.-7. szektorainak a foglaltsága
142	a 35. sáv 8.-16. szektorainak a foglaltsága
143	a 35. sáv 17.-23. szektorainak a foglaltsága
144-159	a lemez neve inverz szóközökkel feltöltve
160-161	inverz szóköz
162-163	lemez azonosítója
164	inverz szóköz
165-166	2A = a DOS (2) verziójának azonosítója
167-170	inverz szóközök
171-255	nem használt (=CHR$(0))

BAM 1571 (18. sáv, 0. szektor)

Byte	Jelentés
0	A következő katalógus-blokk sávja (mindig 18)
1	A következő katalógus-blokk szektora (mindig 1)
2	65, ez jelzi a 1541/1570/1571/4040 formátumot
3	Lemez oldalainak a száma: $80 = kétoldalas; $00 = egyoldalas
4	az 1. sávon szabad blokkok száma
5	az 1. sáv 0.-7. szektorainak a foglaltsága
6	az 1. sáv 8.-16. szektorainak a foglaltsága
7	az 1. sáv 17.-23. szektorainak a foglaltsága
.....	
140	a 35. sávon szabad blokkok száma
141	a 35. sáv 0.-7. szektorainak a foglaltsága
142	a 35. sáv 8.-16. szektorainak a foglaltsága
143	a 35. sáv 17.-23. szektorainak a foglaltsága
144-159	a lemez neve inverz szóközökkel feltöltve
160-161	inverz szóköz
162-163	lemez azonosítója
164	inverz szóköz
165-166	2A = a DOS (2) verziójának azonosítója
167-170	inverz szóközök
171-220	nem használt (=CHR$(0))
221-237	a 36.-52. sávokon szabad szektorok száma
238	0 (= az 53. sávon szabad szektorok száma
239-244	a 54.-59. sávokon szabad szektorok száma
245-250	a 60.-65. sávokon szabad szektorok száma
251-255	a 66.-70. sávokon szabad szektorok száma

A következő BAM bejegyzés csak a 1571-es lemezegység és kétoldalas lemez használatakor jelenik meg.

BAM 1571 (53. sáv, 0. szektor)

Byte	Jelentés
0	az 1. sáv 0.-7. szektorainak a foglaltsága
1	az 1. sáv 8.-16. szektorainak a foglaltsága
2	az 1. sáv 17.-23. szektorainak a foglaltsága
.....	
102	a 35. sáv 0.-7. szektorainak a foglaltsága
103	a 35. sáv 8.-16. szektorainak a foglaltsága
104	a 35. sáv 17.-23. szektorainak a foglaltsága
105-255	nem használt (CHR$(0))

BAM 1581 (40. sáv, 0. szektor)

Byte	Jelentés
0-1	az első katalógus blokk sáv és szektorszáma
2	lemezegység verziószám
3	$00
4-21	Lemez neve inverz szóközökkel feltöltve
22-23	Lemez azonosítója
24	Inverz szóköz ($A0)
25	DOS verziószám (=3)
26	lemezegység verziószám (C)
27-28	Inverz szóköz ($A0)
29-255	Nem használt (CHR$(0))

BAM 1581 (40. sáv, 1. szektor)

Byte	Jelentés
0-1	a következő BAM blokk sáv és szektorszáma
2	verziószám
3	kiegészítő verziószám
4-5	lemez azonosítója
6	I/O byte 7. bit ellenőrzés ki/be 6. CRC ellenőrzes ki/be
7	autotöltő flag
8-15	fenntartott
16-255	foglaltság jelző az 1.-40. szektorokra

BAM 1581 (40. sáv, 2. szektor)

Byte	Jelentés
0	0
1	$FF
2	verziószám
3	kiegészítő verziószám
4-5	lemez azonosítója
6	I/O byte 7. bit ellenőrzés ki/be 6. CRC ellenőrzes ki/be
7	autotöltő flag
8-15	fenntartott
16-255	foglaltság jelző a 41.-80. szektorokra

Ha a 1581-es lemezegységen olyan partíciót hoztunk létre, ami egyben alkatalógus is, akkor a partíció első sávján a DOS a fentihez hasonló struktúrát hoz létre. A sáv és szektorszámok valódiak, nem a partíció elejéhez viszonyítottak.

Katalógus bejegyzések

Az egyes lemezegységek katalógusbejegyzései formára azonosak. Egyetlen eltérés, hogy hol helyezkedik el a katalógus és hogy hány blokkból állhat.

Katalógus blokk

Byte	Jelentés
0,1	A következő katalógus blokk sáv és szektorszáma
2-31	1. file bejegyzés
34-63	2. file bejegyzés
66-95	3. file bejegyzés
98-127	4. file bejegyzés
130-159	5. file bejegyzés
162-191	6. file bejegyzés
194-223	7. file bejegyzés
226-255	8. file bejegyzés

File bejegyzés felépítése

Byte	Jelentés
0	A file típusa. Ha a file-t lezárták, akkor a típust jelző byte 7. bitje magas (OR $80). Ha a 6. bit is magas (OR $C0), akkor a file még zárolt is és nem lehet a "S0" DOS paranccsal törölni. Az egyes file-ok a következők: 1 = SEQ (szekvenciális) 2 = PRG (program) 3 = USR (felhasználói) 4 = REL (relativ) 5 = CBM (partíció)
1-2	A file első blokkjának sáv, szektor száma
3-18	A file neve inverz szóközökkel kiegészítve
19-20	A super-leíró blokk vagy az első leíró blokk sáv és szektorszáma (REL!)
21	Rekordhossza (REL!)
22-25	Nem használt
26-27	@SAVE és @OPEN esetén a helyettesítő file kezdő sáv és szektorszáma
28-29	A file blokkjainak a száma: alsó, felső byte

A PRG, SEQ, USR és REL típusú file-ok tárolása egyformán történik. Ezt a formátumot hívhatjuk általános formátumnak is.

Általános file formátum

Byte	Jelentés
0,1	A file következő blokkjának sáv és szektorszáma. Ha a file-nak ez az utolsó adatblokkja, akkor a 0. byte egy nulla byte, s az 1. byte a file-hoz még tartozó byte-ok számát mutatja.
2-255	A file byte-jai. Ha ez a file utolsó blokkja, akkor a felesleges byte-ok tartalma esetleges (nem törlődnek pl.)

A relativ file-ok esetén a rekordok mielőbbi megtalálását leíró blokkok segítik. A leíró blokkban összesen 120 adatblokk sáv és szektorszámát tárolja a DOS. A super-leíró blokkok pedig a leíró blokkok elhelyezését adják meg. Super-leíró blokkot csak a 1581-es lemezegységnek van.

Leíró blokk

Byte	Jelentés
0-1	A következő leíró blokk sáv és szektorszáma
2	A leíró blokk sorszáma (0-5)
3	Rekordhossz
4-5	0. leíró blokk sáv- és szektorszáma
6-7	1. leíró blokk sáv- és szektorszáma (0)
8-9	2. leíró blokk sáv- és szektorszáma (0)
10-11	3. leíró blokk sáv- és szektorszáma (0)
12-13	4. leíró blokk sáv- és szektorszáma (0)
14-15	5. leíró blokk sáv- és szektorszáma (0)
16-255	120 adatblokk sáv- és szektorszáma

Super-leíró blokk

Byte	Jelentés
0-1	0. leíró blokk-csoport sáv és szektorszáma
2	$FE
3-4	1. leíró blokk-csoport sáv és szektorszáma
253-254	125. leíró blokk-csoport sáv és szektorszáma

F.19 A C-64-gyel való kompatibilitás

Természetes kérdés: vajon milyen mértében kompatibilis egymással a Commodore 64 és a Commodore 128 C-64-es üzemmódja? A válasz látszólag az, hogy teljes mértékben, hiszen a C-64-es üzemmódban pontosan ugyanazt a ROM-ot használja a gép, mint ami a Commodore 64-ben található. Ennek tanúbizonysága, hogy a Commodore 64-en futó programok változtatás nélkül üzemeltethetők. Nem ez a helyzet a 1570/71-es lemezegységekkel. A gyárilag védett szoftverek egy része nem indul el, csak akkor, ha a 1541-es lemezegységet használjuk. Ugyanez a helyzet a gyorsmásolókkal is. Ennek oka, hogy a 1570/71-es lemezegységek belső felépítése teljesen eltér a 1541-esétől, márpedig mind a védelmi eljárások, mind a gyorsmásolók erre építenek.

A mondottak ellenére a Commodore 64 és a Commodore 128 C-64-es módja egy sor dologban eltér egymástól.

A RESET gomb

A Commodore 64-en nincs RESET gomb. A C-128-on a RESET gomb inicializálja a Commodore 128-at. Lehetőség van tehát arra, hogy a C-64-es módból egyenesen a C-128-ba térjünk vissza. Ez lehetővé teszi a gépi kódú programok megszakítását, s megkönnyítheti a gyári programok védelmének feltörését.

A C-64-es mód inicializálása

A Commodore 64-es gépen, mihelyt a feszültség eléri a 4.75V-ot a processzor végrehajt egy teljes reset ciklust, aminek a végén egy JMP ($FFFC) utasítás kerül végrehajtásra. Amikor a C-128-as gépen áttérünk a C-64-es üzemmódba, a proceszszor már végrehajtott egy ilyen ciklust, s az üzemmód váltás szoftver úton történik meg. Ezt kihasználva módosítani lehet a C-64-es üzemmód reset ciklusát, úgy, hogy az ne egy JMP ($FFFC)-vel kezdődjék. Elérhető például, hogy C-64-es programok is 'boot'-olhatók legyenek. A C-128-as módban betöltött indítóprogram betölti az igazi programot, majd áttér C-64-es üzemmódba, s elindítja a programot.

Lehetőség van C-64-es cartridge szimulálására is. A C-64 induláskor ellenőrzi, hogy a $8004-$8008 címeken a CBM80 szöveg található-e. Ha igen, akkor egy JMP ($8000) ugrást hajt végre a BASIC interpreterbe való belépés helyett. Ha tehát a C-128-as üzemmódba $8000-tőlmegfelelő kódokat és programot helyezünk el, akkor a GO64 parancs végrehajtása után erre a részre kerülhet a vezérlés.

8-bites kapu

A Commodore 64 M6510-es processzorának egy kétirányú hat bites I/O kapuja van, amelyik a memória $0001-es címén helyezkedik el. A $0000-ás cím a kapu adatirány regisztere (1 = kimenet, 0 = bemenet). Ennek segítségével lehet a géphez csatlakoztatott kazettás egységet vezérelni, illetve a három ROM-ot ki/be kapcsolni. (Lásd bővebben C-64 II. kötet 316-317. oldalak!)

A 8510-es processzornak 8 bites I/O kapuja van, s a 6. bitet az interpreter az <ASCII/DIN> kapcsoló állapotának lekérdezésére használja. Ha ezt a bitet bemenetként definiáljuk, akkor lekérdezhetjük a billentyű állapotát. Ha kimenet, akkor programból módosíthatjuk a kijelzett betűk alakját. Ettől függetlenül az <ASCII/DIN> lenyomott állapotában a kijelzett betűk alakja megváltozik

Az alábbi egyszerű programrész BASIC utasítások segítségével állítja át a kijelzett betűk alakját. Először a DIN alaknak megfelelő kijelzést kapunk, majd – bármely billentyű megnyomása után – visszaáll az ASCII kijelzés. Vigyázzunk, hogy a programot az <ASCII/DIN> billentyű felengedett állapotában indítsuk el!

```
10 PRINT"<CLR>ABCDEFGHIJK1234567890"
20 POKE0,PEEK(0) OR 2↑6: REM 6.BIT KIMENET
30 POKE 1,PEEK(1) AND (255-2↑6)
40 GET A$: IF A$="" THEN GOTO 30
50 POKE 1,PEEK(1) OR 2↑6
```

A módosított VICII chip használata

A C-64-es üzemmódban a módosított VICII chipet használjuk. Ez további két regisztert tartalmaz, amivel a használt órajelet és a billentyűzetet lehet lekérdezni.

A C-64-es üzemmódba való belépéskor a rendszer 1MHz-es órajelet használ. Ha ezt a kétszeresére akarjuk növelni, akkor a

```
POKE 53296,1
```

parancsot kell kiadni. Ilyenkor a képernyő teljesen tönkremegy, mert a VICII nem képes ezzel az órajellel szinkronban dolgozni. Ha zavar a képernyő, akkor a VICII chip kikapcsolásával elérhetjük, hogy homogén, egyszínű képernyőt kapjuk. (Lásd C-64 II.kötet 228. oldal!) Az 1MHz-es sebességre a

```
POKE 53296,0
```

parancs kiadásával térhetünk vissza.

A VICII módosításának további következménye, hogy a bővített klaviatúrával dolgozhatunk. Természetesen ezt a lehetőséget csak megfelelő gépi kódú rutinok írásával tudjuk felhasználni, a C-64 ROM-ja erre nincs felkészítve.

Az alábbi gépi kódú rutin leolvassa az 'új' billentyűk állapotát. Ha egy nem C-64-es billentyűt nyomtunk meg, akkor a 928-as címen ($03A0) egy 1-24 közötti kódszámot kapunk. Ha ilyen billentyűt nem nyomtunk meg, akkor a kód 255:

```
033C  78        SEI         ; megszakítások letiltása
033D  A9 00     LDA #$00    ; valamennyi billentyű
033F  8D 2F D0  STA $D02F   ; tesztelése
0342  A9 FF     LDA #$FF    ; jelzés, hogy nincs lenyomva
0344  8D A0 03  STA $03A0
0347  8D 00 DC  STA $DC00   ; a billentyűzet oszlopainak
034A  AD 01 DC  LDA $DC01   ; kiválasztása, majd a sorok be-
034D  C9 FF     CMP #$FF    ; olvasása és ellenőrzése, hogy
034F  D0 07     BNE $0358   ; van-e lenyomott billentyű
0351  A9 FF     LDA #$FF    ; nincs: az új billentyűzetrész
0353  8D 2F D0  STA $D02F   ; passzíválása
0356  58        CLI         ; megszakítások engedélyezése
0357  60        RTS         ; a program vége
0358  A2 03     LDX #$03    ; három billentyűzetsor teszte-
035A  8A        TXA         ; lése melyikben van lenyomott
                            ; billentyű
035B  48        PHA         ; az érték veremben tárolása
035C  BD 9C 03  LDA $039C,X;az x-dik sor kiválasztása
035F  8D 2F D0  STA $D02F
0362  AD 01 DC  LDA $DC01   ; a sorok beolvasása
0365  CD 01 DC  CMP $DC01   ; amikor nincs pergés
0368  D0 F8     BNE $0362   ; ha igen ismétlés
036A  C9 FF     CMP #$FF    ; ha nem ebben a sorban volt a
036C  F0 0B     BEQ $0379   ; lenyomás, akkor ugrás $0379-re
036E  A2 00     LDX #$00    ; hányadik bit alacsony?
0370  A0 08     LDY #$08    ; számláló a ciklushoz
0372  4A        LSR         ; a következő bit kishiftelése
0373  90 10     BCC $0385   ; ellenőrzése, hogy 0-e?
0375  E8        INX         ; nem: bitszám növelése és a
0376  88        BEY         ; ciklusszámláló növelése,
0377  D0 F9     BNE $0372   ; a ciklus folytatása
0379  68        PLA         ; az oszlop mutató visszahozása
037A  AA        TAX         ; x-be töltése
037B  CA        DEX         ; és csökkentése
037C  D0 DC     BNE $035A   ; áttérés a következő oszlopra
037E  A9 FF     LDA #$FF    ; nem volt lenyomott billentyű, ezért
0380  8D 2F D0  STA $D02F   ; új billentyűk passzíválása,
0383  58        CLI         ; megszakítások engedélyezése,
0384  60        RTS         ; visszatérés az alprogramból
0385  68        PLA         ; a tárolt oszlopszám olvasása
0386  AA        TAX         ; x-be írása
0387  CA        DEX         ; és csökkentése
0388  8A        TXA         ; a-ba írása
0389  0A        ASL         ; és háromszor
038A  0A        ASL         ; jobbra shiftelésével
038B  0A        ASL         ; 8-cal való szorzása
038C  8C A1 03  STY $03A1   ; a sorszám tárolása
038F  18        CLC         ; és hozzáadása
0390  6D A1 03  ADC $03A1   ; az oszlopszám 8-szorosához
0393  8D A0 03  STA $03A0   ; a kód tárolása
0396  A9 FF     LDA #$FF    ; az új billentyűk passzíválása
0398  8D 2F D0  STA $D02F
```

```
039B  58        CLI         ; a megszakítások engedélyezése
039C  60        RTS         ; visszatérésaz alprogramból
039D  FE FD FB  .BYTE $FE, $FD, $FB
03A0  00        .END
```

A gépi kódú programot a SYS(828) utasítással hívhatjuk meg. Ezt követően a lenyomott új billentyű kódját a PEEK(928) függvény segítségével kaphatjuk meg. Az alábbi BASIC program bemutatja a rutin használatát, s egyben egy BASIC töltő is a fenti gépi kódú rutinra. A program futás a <STOP-RESTORE> billentyűzéssel állítható meg. Ezt követően a ?KO$(I) kiadásával (I = 1...24) megállapíthatjuk melyik billentyűnek mennyi a kódja.

```
100 DIM KO$(24): FOR I=1 TO 24: READ KO$(I): NEXT I
110 CI=828
120 READ A: IF A=-1 THEN PRINT"<CLR>MOST NYOMD!": GOTO 140
130 POKE CI,A: CI=CI+1: GOTO 120
140 SYS 828
150 :
160 A=PEEK(928)
170 IF A>24 THEN GOTO 140
180 PRINT KO$(A): GOTO 140
190 :
200 DATA 1,7,4,2,TAB,5,8,HELP
210 DATA 3,9,6,ENTER,LF,-,+,ESC
220 :
230 DATA NOSCR,JOBBRA,BALRA,LE,FEL,.,0,ALT
240 DATA 120,169,0,141,47,208,169,255
250 DATA 141,160,3,141,0,220,173,1
260 DATA 220,201,255,208,7,169,255,141
270 DATA 47,208,88,96,162,3,138,72
280 DATA 189,156,3,141,47,208,173,1
290 DATA 220,205,1,220,208,248,201,255
300 DATA 240,11,162,0,160,8,74,144
310 DATA 16,232,136,208,249,104,170,202
320 DATA 208,220,169,255,141,47,208,88
330 DATA 96,104,170,202,138,10,10,10
340 DATA 140,161,3,24,109,161,3,141
350 DATA 160,3,169,255,141,47,208,88
360 DATA 96,254,253,251,-1
```

A 80 oszlopos video chip használata

A $D600 címen a C-64-es üzemmódban is ott találjuk a VDC 80 oszlopos video chip kommunikációs regisztereit. Ez azt jelenti, hogy a C-64-es üzemmódban is 80 oszlopos kijelzést használhatunk! A baj persze ugyanaz, mint az 'új' billentyűk használatakor: a BASIC ezt nem tudja használni. Ha használni akarjuk, valamennyi kezelő rutint nekünk kell megírnunk, méghozzá gépi kódban. Ez előreláthatóan annyi helyet igényel, hogy nem érdemes megtenni.

A 80 oszlopos video chipnek azonban van egy másik felhasználói lehetősége: a 16K-s memóriája!. A C-64-es üzemmódban további 16K-nyi memória áll rendelkezésünkre. Ha a C-64-re eleve úgy írtuk meg a programot, hogy pl. a BASIC vagy a KERNAL mögötti RAM-ot használja, mert kevés hely van, akkor ez most tovább bővül. Ráadásul ennek a lehetőségnek a kihasználásához nem is kell olyan sok és hosszú gépi kódú részeket írni. Az alábbiakban egy olyan rutint adunk, amelyik az operatív tár és a videomemória közti adatcserét végzi. Ehhez adunk egy BASIC töltő programot is, amelyik egyben rögtön be is mutatja ennek használatát.

Az alábbi gépi kódú programrész három rutint tartalmaz, amelyek a $C000, $C003, $C006 belépési pontokon keresztül érhetők el. Meghívásuk előtt három mutatót kell beállítani. Az első a ($FC) mutató a processzor memóriaterületének első mozgatandó byte-jára mutat, míg a ($C009) mutató az első, már nem mozgatandó címet tartalmazza. A VDC memóriájában az első mozgatandó byte-ot a ($fe) mutató tartalmazza.

Az első rutin a memória tartalmát másolja a video chip memóriájába. A második fordítva: a video chip memóriáját írja át a processzor memóriájába. A harmadik felcseréli a két memória területet.

```
C000  4C 0B C0   JMP $C00B    ; mem --> vdcmem
C003  4C 1C C0   JMP $C01C    ; vdcmem --> mem
C006  4C 2D C0   JMP $C02D    ; vdcmem <--> mem
C009  00 00      .BYTE 0,0    ; mutató az utolsó másolandó byte-ra.
                              ; a processzor memóriájának átmásolása
                              ; a VDC memóriájába
                              ; ($fc) mutat az első átmásolandó
                              ; byte-ra,($fe) a VDC memóriájába.
C00B  A6 FE      LDX $FE      ; VDC mutató: alsó byte x-ben
C00D  A0 00      LDY #$00     ; átmásolandó byte beírása az
C00F  B1 FC      LDA ($FC),Y  ; A regiszterbe
C011  A4 FF      LDY $FF      ; VDC-beli mutató: felső byte Y-ban
C013  20 7F C0   JSR $C07F    ; byte beírása a VDC memóriába
C016  20 4B C0   JSR $C04B    ; mutatók növelése
C019  D0 F0      BNE $C00B    ; ellenőrzés: kell-e még másolni?
C01B  60         RTS          ; nem --> visszatérés.
                              ; a VDC memóriájának átmásolása
                              ; a processzor memóriájába
                              ; ($fe) mutat az első átmásolandó
                              ; byte-ra, ($fc) a memóriába.
C01C  A6 FE      LDX $FE      ; VDC mutató: alsó byte X-ben.
C01E  A4 FF      LDY $FF      ; VDC mutató: felső byte Y-ban.
C020  20 64 C0   JSR $C064    ; a VDC memóriából az adat beolvasása
C023  A0 00      LDY #$00     ; az adatot a mutatónak megfelelő
C025  91 FC      STA ($FC),Y  ; helyre írjuk.
C027  20 4B C0   JSR $C04B    ; mutatók növelése, majd ellenőrzés
C02A  D0 F0      BNE $C01C    ; kell-e még másolni.
C02C  60         RTS          ; Nem --> visszatérés
                              ; a processzor memóriájának
                              ; cserélése a VDC memóriájába
                              ; ($fc) mutat az első cserélendő
                              ; byte-ra, ($fe) a VDC memóriájába.
C02D  A6 FE      LDX $FE      ; VDC mutató: alsó byte
C02F  A4 FF      LDY $FF      ; VDC mutató: felső byte
```

```
C031  20 64 C0  JSR $C064    ; az adatbyte beolvasása a VDC memóri-
C034  48 PHA                 ; ájából, s a verembe mentése.
C035  A0 00     LDY #$00     ; a memóriából a byte betöltése
C037  B1 FC     LDA ($FC),Y  ; az akkumulátorba.
C039  A6 FE     LDX $FE      ; VDC mutató: alsó byte
C03B  A4 FF     LDY $FF      ; VDC mutató: felső byte
C03D  20 7F C0  JSR $C07F    ; a byte beírása a VDC memóriájába
C040  A0 00     LDY #$00     ; az akkumulátorba az elmentett byte
C042  68        PLA          ; majd beírása
C043  91 FC     STA ($FC),Y  ; a memóriába.
C045  20 4B C0  JSR $C04B    ; mutatók növelése, majd ellenőrzés,
C048  D0 E3     BNE $C02D    ; kell-e még cserélni.
C04A  60        RTS          ; nem --> visszatérés.
                             ; Mutatók növelése és a ($c009)
                             ; mutató elérésének ellenőrzése.
C04B  E6 FE     INC $FE      ; mutató alsó byte-jának növelése
C04D  D0 02     BNE $C051    ; 0 --> felsőt is kell növelni
C04F  E6 FF     INC $FF      ; felső byte növelése
C051  E6 FC     INC $FC      ; VDC mutató alsó byte-jának növelése
C053  D0 02     BNE $C057    ; 0 --> felsőt is kell
C055  E6 FD     INC $FD      ; felső byte növelése
C057  A5 FD     LDA $FD      ; mutató felső byte
C059  CD 0A C0  CMP $C00A    ; összehasonlítása a végcímmel
C05C  D0 05     BNE $C063    ; nem egyenlő --> kilépés
C05E  A5 FC     LDA $FC      ; ha egyenlő az alsó byte-okat
C060  CD 09 C0  CMP $C009    ; is össze kell hasonlítani
C063  60        RTS          ; visszatérés az alprogramból
                             ; PEEK rutin
                             ; a VDC memóriájának olvasása,
                             ; az eredmény az A regiszterben.
                             ; X a cím alsó, Y a felső byte-ja.
C064  8A        TXA          ; X átírása A-ba,
C065  48        PHA          ; elmentése a verembe.
C066  A2 12     LDX #$12     ; a felső byte regisztere X-ben
C068  98        TYA          ; a cím felső byte-ja A-ban,
C069  20 93 C0  JSR $C093    ; majd beírása a regiszterbe.
C06C  E8        INX          ; X-ben az alsó byte regisztere
C06D  68        PLA          ; a cím alsó byte-ja A-ban,
C06E  20 93 C0  JSR $C093    ; majd a regiszterbe írása
C071  A2 1F     LDX #$1F     ; az adatbyte regisztere X-ben
C073  8E 00 D6  STX $D600    ; beírás a regiszterbe
C076  2C 00 D6  BIT $D600    ; beírta-e a video chip?
C079  10 FB     BPL $C076    ; ha nem, várakozás,
C07B  AD 01 D6  LDA $D601    ; ha igen, az adatbyte olvasása.
C07E  60        RTS          ; visszatérés az alprogramból.
                             ; POKE rutin
                             ; Az A-ban levő byte-ot beírja a VDC
                             ; memória területére. A memória címe
                             ; alsó byte-ja az X-ben, felső az Y-ban.
C07F  48        PHA          ; A byte elmentése a verembe.
C082  A2 12     LDX #$12     ; A VDC memória felső byte-ját tartal-
C084  98        TYA
C085  20 93 C0  JSR $C093    ; mazó regiszter beállítása.
C088  E8        INX          ; X az alsó byte regisztere
C089  68        PLA          ; az alsó byte A-ba hozása,
C08A  20 93 C0  JSR $C093    ; beírása a regiszterbe.
C08D  A2 1F     LDX #$1F     ; az adatbyte regisztere X-ben
```

```
C08F  68        PLA          ; az adatbyte visszahozása,
C090  4C 93 C0  JMP $C093    ; s beírása a regiszterbe
C093  8E 00 D6  STX $D600    ; beírás a kommunikációs regiszterbe
C096  2C 00 D6  BIT $D600    ; majd ellenőrzése, hogy végre lett-e
C099  10 FB     BPL $C096    ; hajtva. Ha nem, akkor várakozás
C09B  8D 01 D6  STA $D601    ; az érték tárolása
C09E  60        RTS          ; visszatérés az alprogramból
```

Az alábbi BASIC program DATA-ban tárolja a fenti gépi kódú rutint, s onnan olvasva a 100-140 ciklus tölti be a helyére. A 140. sorban ellenőrizzük, hogy jó adatokat írtunk-e be. Ha a program itt megszakad, akkor újra kell ellenőrizni a DATA-ba beírt értékeket.

A memória másolást a képernyő memória másolásával próbáljuk ki, mert így azonnal ellenőrizhető a másolás sikere. A 160-210 sorokban a 40 oszlopos képernyőre kiírunk valamit, majd a 240-250 sorokban ezt átmásoljuk a VDC chip memóriájába. Ha a 80 oszlopos monitort bekapcsoltuk, akkor ezt vizulásan is tudjuk ellenőrizni! Ezután töröljük a képernyőt s a sok ? utasítással a kurzort a 9. sorba mozgatjuk. Rövid ideig várakozik a program (290. sor), majd a VDC chip memóriájából visszamásolja az adatokat.

```
100 CI=49152: S=0
110 READ X: IF X=-1 THEN GOTO 140
120 POKE CI,X: CI=CI+1: S=S+X
130 GOTO 100
140 IF S<>22044 THEN PRINT"HIBASAN IRTA BE!": END
150 :
160 PRINT"<CLR>AAAAAAAAAAA": PRINT
170 PRINT" BBBBBBBBB": PRINT
180 PRINT"  CCCCCCC": PRINT
190 PRINT"   DDDDD": PRINT
200 PRINT"    EEE": PRINT
210 PRINT"     F": PRINT
220 :
230 REM MEM-->VDC
240 MUT=1024: VMUT=0: DARAB=999
250 GOSUB 360: SYS 49152
260 :
270 REM VARAKOZAS
280 PRINT"<CLR>": ?: ?: ?: ?: ?: ?: ?: ?: ?
290 FOR I=1 TO 1500: NEXT I
300 :
310 REM VDC-->MEM
320 MUT=1024: VMUT=0: DARAB=999
330 GOSUB 360: SYS 49155
340 END
350 :
360 REM MUT,VMUT,VEGCIM BEALLITASA
370 POKE 252, MUT AND 255
380 POKE 253, INT(MUT/256)
390 POKE 254, VMUT AND 255
400 POKE 255, INT(VMUT/256)
410 VEGCIM=MUT+DARAB+1
420 POKE 49161, VEGCIM AND 255
```

```
430 POKE 49162, INT(VEGCIM/256)
440 RETURN
450 :
460 REM GEPI KODU PROGRAM ADATAI
470 DATA 76,11,192,76,28.192,76,45
480 DATA 192,233,7,166,254,160,0,177
490 DATA 252,164,255,32,127,192,32,75
500 DATA 192,208,240,96,166,254,164,255
510 DATA 32,100,192,160,0,145,252,32
520 DATA 75,192,208,240,96,166,254,164
530 DATA 255,32,100,192,72,160,0,177
540 DATA 252,166,254,164,255,32,127,192
550 DATA 160,0,104,145,252,32,75,192
560 DATA 208,227,96,230,254,208,2,230
570 DATA 255,230,252,208,2,230,253,165
580 DATA 253,205,10,192,208,5,165,252
590 DATA 205,9,192,96,138,72,162,18
600 DATA 152,32,147,192,232,104,32,147
610 DATA 192,162,31,142,0,214,44,0
620 DATA 214,16,251,173,1,214,96,72
630 DATA 138,72,162,18,152,32,147,192
640 DATA 232,104,32,147,192,162,31,104
650 DATA 76,147,192,142,0,214,44,0
660 DATA 214,16,251,141,1,214,96,-1
```

F.20 Összefoglaló táblázatok

F.20.1 A VIC Chip regiszterei

Kezdőcím: 53248 ($D000)

Cím	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Leírás
00 ($00)	M0X7	M0X6	M0X5	M0X4	M0X3	M0X2	M0X1	M0X0	Sprite#0 X koordináta
01 ($01)	M0Y7	M0Y6	M0Y5	M0Y4	M0Y3	M0Y2	M0Y1	M0Y0	Sprite#0 Y koordináta
02 ($02)	M1X7	M1X6	M1X5	M1X4	M1X3	M1X2	M1X1	M1X0	Sprite#1 X koordináta
03 ($03)	M1Y7	M1Y6	M1Y5	M1Y4	M1Y3	M1Y2	M1Y1	M1Y0	Sprite#1 Y koordináta
04 ($04)	M2X7	M2X6	M2X5	M2X4	M2X3	M2X2	M2X1	M2X0	Sprite#2 X koordináta
05 ($05)	M2Y7	M2Y6	M2Y5	M2Y4	M2Y3	M2Y2	M2Y1	M2Y0	Sprite#2 Y koordináta
06 ($06)	M3X7	M3X6	M3X5	M3X4	M3X3	M3X2	M3X1	M3X0	Sprite#3 X koordináta
07 ($07)	M3Y7	M3Y6	M3Y5	M3Y4	M3Y3	M3Y2	M3Y1	M3Y0	Sprite#3 Y koordináta
08 ($08)	M4X7	M4X6	M4X5	M4X4	M4X3	M4X2	M4X1	M4X0	Sprite#4 X koordináta
09 ($09)	M4Y7	M4Y6	M4Y5	M4Y4	M4Y3	M4Y2	M4Y1	M4Y0	Sprite#4 Y koordináta
10 ($0A)	M5X7	M5X6	M5X5	M5X4	M5X3	M5X2	M5X1	M5X0	Sprite#5 X koordináta
11 ($0B)	M5Y7	M5Y6	M5Y5	M5Y4	M5Y3	M5Y2	M5Y1	M5Y0	Sprite#5 Y koordináta
12 ($0C)	M6X7	M6X6	M6X5	M6X4	M6X3	M6X2	M6X1	M6X0	Sprite#6 X koordináta
13 ($0D)	M6Y7	M6Y6	M6Y5	M6Y4	M6Y3	M6Y2	M6Y1	M6Y0	Sprite#6 Y koordináta
14 ($0E)	M7X7	M7X6	M7X5	M7X4	M7X3	M7X2	M7X1	M7X0	Sprite#7 X koordináta
15 ($0F)	M7Y7	M7Y6	M7Y5	M7Y4	M7Y3	M7Y2	M7Y1	M7Y0	Sprite#7 Y koordináta
16 ($10)	M7X8	M6X8	M5X8	M4X8	M3X8	M2X8	M1X8	M0X8	Legnagyobb helyiértékű bit-X koordináta
17 ($11)	RC8	ECM	BMM	DEN	RSEL	Y2	Y1	Y0	Mód/Y irányú görgetés
18 ($12)	RC7	RC6	RC5	RC4	RC3	RC2	RC1	RC0	Raszter regiszter
19 ($13)	LPX8	LPX7	LPX6	LPX5	LPX4	LPX3	LPX2	LPX1	Fényceruza X pozíció
20 ($14)	LPY7	LPY6	LPY5	LPY4	LPY3	LPY2	LPY1	LPY0	Fényceruza Y pozíció
21 ($15)	M7E	M6E	M5E	M4E	M3E	M2E	M1E	M0E	Sprite-ok engedélyezése
22 ($16)	-	-	RES	MCM	CSEL	X2	X1	X0	Mód/X irányú görgetés
23 ($17)	M7YE	M6YE	M5YE	M4YE	M3YE	M2YE	M1YE	M0YE	Y irányú nagyítás(sprite)
24 ($18)	VM13	VM12	VM11	VM10	CB13	CB12	CB11	-	Memória mutatói
25 ($19)	IRQ	-	-	-	ILP	IMMC	IMBC	IRST	Megszakítás regiszter
26 ($1A)	-	-	-	-	ELP	EMMC	EMBC	ERST	Megszakítást engedélyező regiszter
27 ($1B)	M7DP	M6DP	M5DP	M4DP	M3DP	M2DP	M1DP	M0DP	Háttér-prioritás
28 ($1C)	M7MC	M6MC	M5MC	M4MC	M3MC	M2MC	M1MC	M0MC	Sprite többszín üzemmód
29 ($1D)	M7XE	M6XE	M5XE	M4XE	M3XE	M2XE	M1XE	M0XE	X irányú nagyítás(sprite)
30 ($1E)	M7M	M6M	M5M	M4M	M3M	M2M	M1M	M0M	Sprite-sprite ütközés
31 ($1F)	M7D	M6D	M5D	M4D	M3D	M2D	M1D	M0D	Sprite-háttér ütközés
32 ($20)	-	-	-	-	EC3	EC2	EC1	EC0	Keret színe
33 ($21)	-	-	-	-	B0C3	B0C2	B0C1	B0C0	Háttér#0 szín
34 ($22)	-	-	-	-	B1C3	B1C2	B1C1	B1C0	Háttér#1 szín
35 ($23)	-	-	-	-	B2C3	B2C2	B2C1	B2C0	Háttér#2 szín
36 ($24)	-	-	-	-	B3C3	B3C2	B3C1	B3C0	Háttér#3 szín

Cím	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Leírás
37 ($25)	-	-	-	-	MM03	MM02	MM01	MM00	Sprite többszín #0
38 ($26)	-	-	-	-	MM13	MM12	MM11	MM10	Sprite többszín #1
39 ($27)	-	-	-	-	M0C3	M0C2	M0C1	M0C0	Sprite#0 színe
40 ($28)	-	-	-	-	M1C3	M1C2	M1C1	M1C0	Sprite#1 színe
41 ($29)	-	-	-	-	M2C3	M2C2	M2C1	M2C0	Sprite#2 színe
42 ($2A)	-	-	-	-	M3C3	M3C2	M3C1	M3C0	Sprite#3 színe
43 ($2B)	-	-	-	-	M4C3	M4C2	M4C1	M4C0	Sprite#4 színe
44 ($2C)	-	-	-	-	M5C3	M5C2	M5C1	M5C0	Sprite#5 színe
45 ($2D)	-	-	-	-	M6C3	M6C2	M6C1	M6C0	Sprite#6 színe
46 ($2E)	-	-	-	-	M7C3	M7C2	M7C1	M7C0	Sprite#7 színe
47*($2F)	RK	-	-	-	K3	K2	K1	K0	Billentyűzet
48*($3A)	-	-	-	-	-	-	-	2MHZ	1=2MHZ-es órajel

A *-gal megjelölt regiszterek **csak** a C-128-as gépen használhatók!

Regiszterek tartalma

Név	Leírás
RC8	A raszter regiszter legnagyobb helyiértékű bitje
ECM	Bővített háttérszín üzemmód (1 = bekapcsolva)
BMM	Bit térképes üzemmód (1 = bekapcsolva)
DEN	Üres képernyő (0 = üres)
RSEL	24/25 soros képernyő (1 = 25 sor)
Y2-Y0	Y irányú görgetés nagysága (kezdőérték = 3)
RES	0 (mindig!)
MCM	Többszínű üzemmód (1 = bekapcsolva)
CSEL	38/40 oszlopos képernyő (1 = 40 oszlopos)
X2-X0	X irányú görgetés (kezdőérték = 0)
VM13-VM10	Képernyő memória címe a kiválasztott szeletben
CB13-CB11	Karakter memória címe a kiválasztott szeletben
IRQ	Maszkolható megszakítás (1 = megszakítás történt)*
ILP	Fényceruza generálta a megszakítást
IMMC	Sprite-sprite ütközés generálta a megszakítást
IMBC	Sprite-háttér ütközés generálta a megszakítást
IRST	Raszter összehasonlítás (egyenlő megszakítás)

* A megszakító regiszter (25.reg.) többször is olvasható. További megszakításokat a rendszer csak a megszakító regiszterben való visszaírás után generál.

F.20.2 A SID chip regiszterei

Kezdőcím: 54272 ($D400)

Név	Leírás
HP	Felüláteresztő szűrő (1 = bekapcsolva)
BP	Sávszűrő (1 = bekapcsolva)
LP	Alul áteresztő szűrő (1 = bekapcsolva)
30FF	Harmadik hang kikapcsolva (1 = kikapcsolva)
FILTEX	Az audión megjelenő jel szűrése (1 = bekapcsolva)
FILT 1 FILT 2 FILT 3	A megfelelő oszcillátor szűrésének engedélyezése (1 = engedélyezve)
F	Fűrészfog jel
H	Háromszög jel
N	Négyszög jel

Regiszterek tartalma

Cím	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Leírás	
00($00)	F_7	F_6	F_5	F_4	F_3	F_2	F_1	F_0	I. hang	Frekvencia alsó byte
01($01)	F_{15}	F_{14}	F_{13}	F_{12}	F_{11}	F_{10}	F_9	F_8		Frekvencia felső byte
02($02)	PW_7	PW_6	PW_5	PW_4	PW_3	PW_2	PW_1	PW_0		Négyszögjel alsó byte
03($03)	–	–	–	–	PW_{11}	PW_{10}	PW_9	PW_8		Négyszögjel felső 4 bit
04($04)	NOISE	N	F	H	TEST	RING	SYNC	GATE		Kontroll regiszter
05($05)	ATK_3	ATK_2	ATK_1	ATK_0	DCY_3	DCY_2	DCY_1	DCY_0		Felfutás/lecsengés
06($06)	STN_3	STN_2	STN_1	STN_0	RLS_3	RLS_2	RLS_1	RLS_0		Kitartás/elengedés
07($07)	F_7	F_6	F_6	F_4	F_3	F_2	F_1	F_0	II.hang	Frekvencia alsó byte
08($08)	F_{15}	F_{14}	F_{13}	F_{12}	F_{11}	F_{10}	F_9	F_8		Frekvencia felső byte
09($09)	PW_7	PW_6	PW_5	PW_4	PW_3	PW_2	PW_1	PW_0		Négyszögjel alsó byte
10($0A)	–	–	–	–	PW_{11}	PW_{10}	PW_9	PW_8		Négyszögjel felső 4 bit
11($0B)	NOISE	N	F	H	TEST	RING	SYNC	GATE		Kontroll regiszter
12($0C)	ATK_3	ATK_2	ATK_1	ATK_0	DCY_3	DCY_2	DCY_1	DCY_0		Felfutás/lecsengés
13($0D)	STN_3	STN_2	STN_1	STN_0	RLS_3	RLS_2	RLS_1	RLS_0		Kitartás/elengedés
14($0E)	F_7	F_6	F_5	F_4	F_3	F_2	F_1	F_0	III.hang	Frekvencia alsó byte
15($0F)	F_{15}	F_{14}	F_{13}	F_{12}	F_{11}	F_{10}	F_9	F_8		Frekvencia felső byte
16($10)	PW_7	PW_6	PW_5	PW_4	PW_3	PW_2	PW_1	PW_0		Négyszögjel alsó byte
17($ 11)	–	–	–	–	PW_{11}	PW_{10}	PW_9	PW_8		Négyszögjel felső 4 bit
18($ 12)	NOISE	N	F	H	TEST	RING	SYNC	GATE		Kontroll regiszter
19($13)	ATK_3	ATK_2	ATK_1	ATK_0	DCY_3	DCY_2	DCY_1	DCY_0		Felfutás/lecsengés
20($14)	STN_3	STN_2	STN_1	STN_0	RLS_3	RLS_2	RLS_1	RLS_0		Kitartás/elengedés
21($15)	–	–	–	–	–	FC_2	FC_1	FC_0	Egyéb	Szűrő alsó bitek
22($16)	FC_{10}	FC_9	FC_8	FC_7	FC_6	FC_5	FC_4	FC_3		Szűrő felső byte
23($17)	RES_3	RES_2	RES_1	RES_0	FILTEX	$FILT_3$	$FILT_2$	$FILT_1$		Rezonancia/szűrés
24($18)	3OFF	HP	BP	LP	VOL_3	VOL_2	VOL_1	VOL_0		Mód/hangerő
25($19)	PX_7	PX_6	PX_5	PX_4	PX_3	PX_2	PX_1	PX_0		X potméter
26($1A)	PY_7	PY_6	PY_5	PY_4	PY_3	PY_2	PY_1	PY_0		Y potméter
27($ 1B)	O_7	O_6	O_5	O_4	O_3	O_2	O_1	O_0		3. hang oszcillátor (olvasható)
28($1C)	E_7	E_6	E_5	E_4	E_3	E_2	E_1	E_0		3. hang burkológörbe (olvasható)

F.20.3 A CIA chip regiszterei

Kezdőcím: CIA#1 56320($DC00)
CIA#2 56576($DD00)

CIA#1: a kapuk felhasználása

A kapu (Billentyű, botkormány, potméterek, fényceruza)

PA7-PA0	Billentyűzet oszlopkiválasztás
PA7-PA6	Potméter olvasásához a játék I/O kiválasztása 01 = 1.játék I/O 10 = 2.játék I/O
PA4	Botkormány (1) [TŰZ] gombja (1 = lenyomva)
PA3-PA2	Potméter (1) [TŰZ] billentyű
PA3-PA0	Botkormány (1) irány-kapcsolói

B kapu (Billentyűzet, botkormány, potméter)

PB7-PB0	Billentyűzet sorának olvasása
PB7	A B időzítő kimenete
PB6	Az A időzítő kimenete
PB4	Botkormány (2) [TŰZ] gombja (1 = lenyomva)
PB3-PB2	Potméterek (2) [TŰZ] billentyűi
PB3-PB0	Botkormány (2) irány-kapcsolói

A Flag-vonal a kazetta **READ** és a soros busz **SRQ IN** bemeneti vonalaira van kötve.

CIA#2: a kapuk felhasználása

A kapu (Soros busz, RS 232, **VIC** szeletválasztás)

PA7	Soros busz **DATA IN**
PA6	Soros busz **CLK IN**
PA5	Soros busz **DATA OUT**
PA4	Soros busz **CLK OUT**
PA3	Soros busz **ATN OUT**
PA2	RS 232 S_{out} és felhasználói kapu
PA1-PA0	A **VIC** szeletválasztása (kezdőérték = 11)

B kapu (RS 232 és felhasználói kapu)

PB7	RS 232 **DSR** és felhasználói kapu
PB6	RS 232 **CTS** és felhasználói kapu
PB5	felhasználói kapu
PB4	RS 232 **CD** és felhasználói kapu
PB3	RS 232 **RI** és felhasználói kapu
PP2	RS 232 **DTR** és felhasználói kapu
PB1	RS 232 **RTS** és felhasználói kapu
PB0	RS 232 S_{in} és felhasználói kapu

A Flag-vonal az RS 232 **NIM** jelét tárolja

Tárgymutató

A tárgymutató az itt feltüntetett és az LSI ATSz. gondozásában megjelent kiadványok alapján készült és arra utal, hogy az adott címszóról melyik szakirodalom hányadik oldalán találunk bővebb információt.

A könyv címe	Jelölése
C – 64 BASIC és felhasználói kézikönyv I.-II.	C-64
C – 64 Assembly	CASM
C – 64 I/O és file-kezelés	CIO