DR. ÚRY LÁSZLÓ

COMMODORE 128

I. KÖTET

BASIC és felhasználói kézikönyv

+ JANE

LSI ALKALMAZÁSTECHNIKAI
TANÁCSADÓ SZOLGÁLAT

Dr. Úry László

Commodore 128

I. kötet

BASIC és felhasználói kézikönyv

+ JANE

Lektorok : Bankó Miklós

Borsányi Attila

LSI

LSI ALKALMAZASTECHNIKAI TANACSADÓ SZOLGALAT

BUDAPEST, 1987

Tartalomjegyzék

6. fejezet
Programozói fogások

7. fejezet
Gépi kódú programozás

Függelék

1. fejezet

Bevezetés

1.1 A kézikönyv feladata

A kézikönyv, amit az Olvasó a kezében tart, felépítésében némileg eltér az LSI ATSZ által kiadott Commodore kézikönyvektől. Az eddig megjelent gépkönyvek és egyéb kiadványok teljességre törekedtek; úgy készültek, hogy további könyvek használata nélkül is kielégítő ismeretet adjanak az adott gépről. A C-128-as felhasználói kézikönyv elsősorban a Commodore 128 személyi számítógép C-128-as módjáról, illetve a gépen futtatható CP/M-ről ad részletes tájékoztatást. Teljes egészében kimarad a C-64-es üzemmódról szóló rész, az M8502-es, illetve a Z80A-s mikroprocesszorok, továbbá a C-128-hoz ugyancsak közvetlenül csatlakoztatható VC 1541-es lemezegység működésének ismertetése. Ezeket ugyanis teljes részletességgel megtalálhatjuk az LSI ATSZ gondozásában már megjelent egyéb kiadványokban. Éppen ezért a könyv végén egy tárgymutatót közlünk, amely nem csak ennek a könyvnek az anyagát dolgozza fel, hanem a következő könyvekét is:

Commodore 64 assembly, Erdős Iván;
Commodore 64, dr. Úry László;
Commodore 64 I/O és file-kezelés, Farkas-Bálint;
Commodore C-16 & C-116, dr. Úry László;
Commodore információs kártya, dr. Úry László;
CP/M operációs rendszer, szerkesztette Szenes Katalin;
Z80 assembler, Hofmanné Boskovitz Éva.

A **Commodore információs kártya** valamennyi Commodore személyi számítógépre, így a C-128-ra vonatkozó információt tartalmazza. A kézikönyv megjelenésével egyidőben jelenik meg a C-128 oktatócsomag, amely a következőket tartalmazza:

- Commdore 128 felhasználói kézikönyv
- Commodore 64 felhasználói kézikönyv
- Commodre információs kártya
- Oktatólemez

Az oktatólemez a C-64-es, C-128-as, illetve a CP/M üzemmód használatát bemutató példaprogramokat tartalmaz.

Miután ismertettük, miről nem szól a kézikönyv, szóljunk azért arról is, hogy miről szól! A 2. fejezet a C-128-as mód képernyő szerkesztőjét, a 3. fejezet pedig ugyanennek az üzemmódnak a BASIC interpreterét ismerteti. A BASIC ismertetésben egészen röviden szerepelnek azok az utasítások, amelyek a C-64-en is ugyanebben a formában vannak meg. Elsősorban a BASIC V2.0-hoz képest új utasításokhoz adunk mintapéldákat.

A 4. fejezet a CP/M üzemmódot ismerteti. A CP/M üzemmód a szokványos CP/M rendszerekhez képest igencsak speciális módon valósul meg, erről a 4.1 paragrafusban szólunk részletesen. A használható CP/M parancsokat ezt követően ismertetjük.

Az 5. fejezet a C-128-hoz kifejlesztett új lemezegységeket mutatja be.

A 6. fejezet a grafikus lehetőségeket, a hanggenerálást, illetve a programstruktúrákat foglalja össze. Mindkét esetben először összefoglaljuk, hogy milyen támogatást nyújt a BASIC V7.0, s azután soroljuk fel azokat a lehetőségeket, amelyeket csak gépi kódból, vagy a PEEK/POKE utasításpár segítségével érhetünk el.

A 7. fejezetben a C-128 gépi kódú programozásához szükséges ismereteket foglaljuk össze. A két processzor (M8502, Z80A) működését - mint már utaltunk rá - nem ismertetjük. Részletesen tárgyaljuk a C-128 gépi kódú monitorát, illetve a C-128 memóriaszervezését biztosító MMU chip használatát.

A függelékben a C-128 hibaüzeneteit s a legfontosabb kódtáblázatokat ismertetjük. Ezt követi az előbb már említett tárgymutató.

1.2 A C-128 mikroszámítógép üzembe állítása

A C-128 típusú személyi számítógépet két eltérő formában hozza a Commodore cég forgalomba C-128, illetve C-128D típusjelzéssel. Az első esetben a billentyűzet magával a számítógéppel van egybeépítve, míg a második esetben a billentyűzet külön csatlakozik a lemezegységgel és a tápegységgel egybeépített számítógéphez. A C-128-as változathoz a lemezegységet külön kell megvásárolnunk.

A Commodore 128-as számítógéphez egyidejűleg két monitor is csatlakoztatható, amelyek egyike televíziós készülék is lehet. Ezen a kijelzés 40 oszlopos. A másik egy RGBI monitor, amelyiken a kijelzés 80 oszlopos. A rendszer üzembeállításának a lépései a következők:

1. Győződjünk meg róla, hogy valamennyi egység kikapcsolt állapotban van. (A kapcsolók **OFF** feliratú helyzetben vannak!) Ha C-128-as számítógépet használunk, akkor a transzformátort először dugjuk be a számítógépbe, a villásdugót pedig a 220V, 50Hz-es hálózatba. (Előtte ellenőrizzük a gépek hátán az adatokat. Az USA-ban vásárolt gépek általában 110V, 60Hz-es feszültséget igényelnek.)

2. Csatlakoztassuk a használni kívánt monitorokat és perifériákat az összekötő kábelek segítségével. A C-128 és a televíziós készülék összeköttetésére szolgáló kábel kivételével ezeket a perifériákkal együtt kell beszerezni.

3. Ha a 40 oszlopos képernyőhöz televíziós készüléket használunk, akkor a készüléket a C-128 frekvenciájára kell hangolni. Ha már a készüléket pl. a C-64-re vagy a C-16-ra ráhangoltuk, akkor az jó lesz a C-128-ra is. Célszerű valamelyik csatornát egyszer s mindenkorra a számítógépre lekötni.

4. Ha kiegészítő berendezéseket is csatlakoztatunk a C-128-hoz, akkor ezek csatlakoztatását a gép bekapcsolása előtt kell elvégezni. Ez az esetleg használt botkormányokra is vonatkozik!

5. Ha a C-128 bővítőjébe, vagy a felhasználói kapuba kártyát illesztünk, azt mindenképpen a bekapcsolása előtt kell megtennünk! Vigyázzunk, hogy a kártyákat megfelelő oldalával felfelé dugjuk be a gépbe!!

6. Bekapcsolás sorrendje: Monitorok (televíziós készülék)
Egyéb kiegészítő berendezések
(lemezegység, nyomtató)
C-128-as gép

Ha csak egyetlen monitort csatlakoztattunk a renszerhez, akkor C-128 bekapcsolása előtt meg kell győződnünk a <40/80 DISPLAY> feliratú kapcsoló állapotáról. Ha ez lenyomott helyzetben van, akkor a rendszer a 80 oszlopos monitoron jelentkezik be, ha nem, akkor a 40 oszloposon (ez lehet a televíziós készülék is).

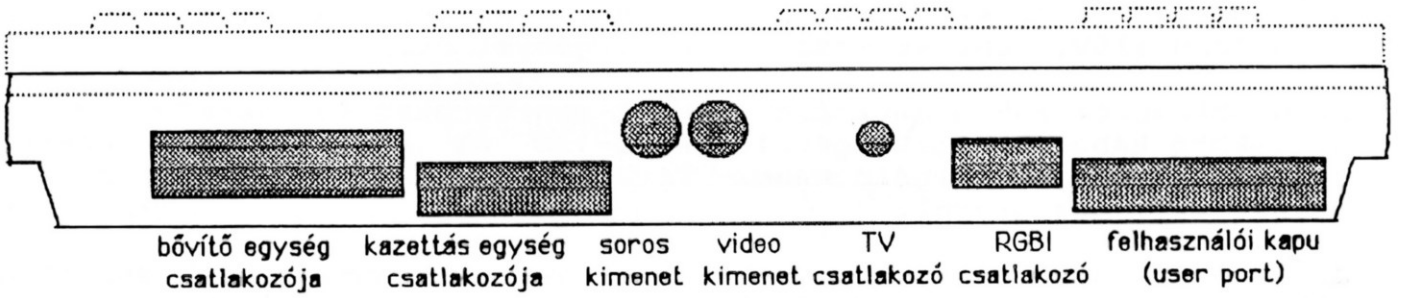
Előfordulhat, hogy automatikus indítású lemezt használunk. Ekkor a lemezt a lemezegység bekapcsolása után, de a számítógép bekapcsolása előtt helyezzük be a meghajtóba!

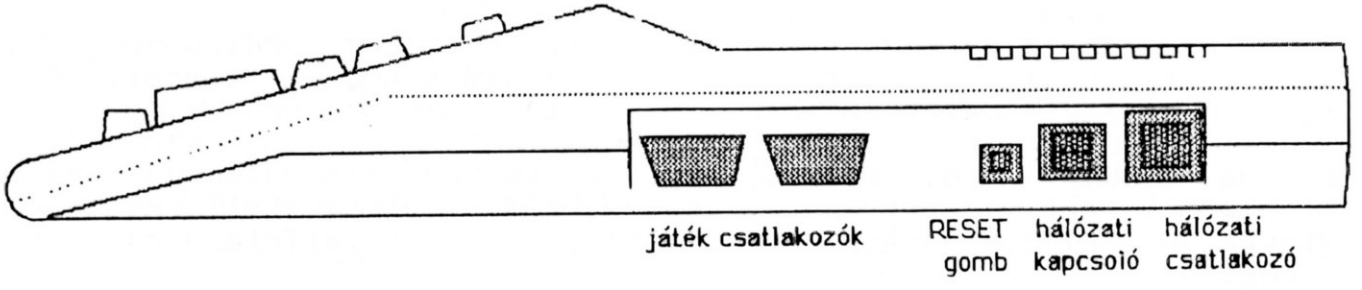
Általában is vigyázzunk, hogy a lemezegységben a lemezegység ki/be kapcsolásakor ne legyen lemez, mert az tönkremehet. A számítógép ki/be kapcsolása a lemezben fizikai kárt nem okoz. Ha azonban megnyitott, de le nem zárt állományok voltak a lemezen a gép kikapcsolásakor, azok logikailag elérhetetlenné válnak. Ez az eset pl. áramkimaradás esetén. Ahol az áramellátás ingadozása ezt megköveteli, használjunk folytonos áramforrást.

Bizonyos esetekben (pl. valamelyik lemezegység átszámozásakor) a fenti bekapcsolási sorrend nem valósítható meg.

7. Kikapcsolás sorrendje: C-128-as gép
Egyéb kiegészítő berendezések
Monitorok

A Commodore-128 személyi számítógép csatlakozói:





1.3 A C-128 számítógép üzemmódjai

A C-128-as számítógép öt eltérő üzemmódban képes dolgozni. Ezek a következők:

- C-64-es üzemmód;
- C-128-as üzemmód 40 oszlopos kijelzés;
- C-128-as üzemmód 80 oszlopos kijelzés;
- CP/M üzemmód 40 oszlopos kijelzés;
- CP/M üzemmód 80 oszlopos kijelzés.

C-64-es üzemmód
C-64-es üzemmódban a gép C-64-es számítógépként üzemel. Hogy a Commodore 128-as számítógép C-64-es üzemmódja valójában mennyire kompatibilis, arról a függelékben részletesen szólunk.

C-64-es üzemmódba háromféleképpen kerülhet a Commodore 128:

a/ Bekapcsoláskor lenyomva tartjuk a <C=> gombot.
b/ C-128-as üzemmódban kiadjuk a **GO64** BASIC parancsot és a rendszer visszakérdezésére (ARE YOU SURE?) Yes-szel (és a <RETURN> megnyomásával) válaszolunk.
c/ A bővítőbe egy C-64-es kártyát (cartridge-t) helyezünk. Megjegyezzük, hogy a C-64-es és C-128-as bővítő kártyák hardverben eltérnek.

Az a/ és a b/ esetben a számítógép a Commodore 64 ismert logojával jelentkezik be:

****** Commodore 64 BASIC V2 ******

64K RAM SYSTEM 38911 BASIC BYTES FREE

A c/ esetben a használt bővítő kártyától függ, hogy mi jelenik meg először a képernyőn. (Néhány extrém esettől eltekintve ugyanaz, mintha egy Commodore 64-et használtunk volna. Lásd a függeléket!)

C-64-es üzemmódból a többi üzemmódba csak a RESET gomb megnyomásával lehet áttérni. Ez inicializálja a C-128-at.

C-128-as üzemmód
A számítógép bekapcsolásakor ebbe az üzemmódba kerül a rendszer. A képernyőn a következő felirat jelenik meg:

COMMODORE BASIC V7.0 122365 BYTE FREE
(C)1985 COMMODORE ELECTRONICS, LTD.
(C)1977 MICROSOFT CORP.
ALL RIGHTS RESERVED

C-128-as üzemmódban változtatható a 40 és a 80 oszlopos kijelzés. A funkcióbillentyűk közt található egy **<40/80 DISPLAY>** feliratú billentyű. Ha bekapcsolás előtt ezt lenyomjuk, akkor 80 oszlopos, különben 40 oszlopos kijelzést kapunk. A rendszer nem ellenőrzi folyamatosan a billentyű állapotát. Kivétel egy <STOP-RESTORE> billentyűzés, amikor a képernyő szerkesztő inicializálásának részeként a gép újraolvassa a billentyűt. Bármikor megváltoztathatjuk a kijelzést az <ESC-X> billentyűzéssel. Ennek hatását természetesen csak akkor érzékeljük, ha két monitort csatlakoztattunk a rendszerhez.

A C-128-as üzemmódból valamennyi üzemmódba át tudunk térni. A GO64 BASIC parancs kiadásával a rendszer C-64-es üzemmódba kerül. Ha a lemezegységbe behelyezzük a CP/M rendszerlemezt és kiadjuk a BOOT parancsot a gép betölti és elindítja a CP/M rendszert.

CP/M üzemmód
CP/M üzemmódban a Z80-as mikroprocesszort és a CP/M+ V3.0 operációs rendszert használhatjuk. Ennek következtében a Commodore 64-en megismert és megszokott hardver-szoftver környezettől merőben eltérőbe kerül a számítógép kezelője, s lehetővé válik a CP/M-re készített programok felhasználása. Ezek egyike például a világon legelterjedtebbnek tekinthető (mikrogépes) adatbáziskezelő rendszer, a dBASE is.

A CP/M programok 24 sor x 80 oszlopos képernyőre készültek. Ezért a 40 oszlopos kijelzés esetén is a sorok 80 karakter szélesek, s speciális billentyűk segítségével lehet a képernyőt, mint egy ablakot mozgatni a 80 oszlopos logikai képernyőn. Mind a 40, mind a 80 oszlopos kijelzés esetén a 25. sort a CP/M rendszerüzenetekre használja.

CP/M üzemmódba kétféleképpen lehet belépni:

a/ A gép bekapcsolása előtt behelyezzük a lemezegységbe a CP/M rendszerlemezt.
b/ C-128-as üzemmódban kiadjuk a **BOOT** BASIC parancsot.

A 40 és 80 oszlopos kijelzés a <40/80 DISPLAY> billentyű helyzetétől függ. A CP/M használata közben a 40 és 80 oszlopos kijelzés csak a DEVICE parancs segítségével változtatható meg.

A rendszer az alábbi üzenettel jelentkezik be:

```
CP/M3.0 On the Commodore 128
        xx column display

A>
```

xx értéke 40 vagy 80, a használt kijelzési módtól függően.

Ha a rendszerlemez tartalmaz egy PROFILE.SUB nevű file-t, akkor az abban tárolt CP/M parancsok végrehajtása is azonnal megkezdődik.

1.4 Bővítési lehetőségek

A C-128 már alapkiépítésben is a nagyobb teljesítményű személyi számítógépek közé tartozik. Mégis a gépet úgy tervezték meg, hogy nemcsak háttértárakat, hanem közvetlen bővítéseket is illeszthetünk hozzá.

Legfontosabbak a **memória bővítések**. Ezek a bővítőegység csatlakozójába helyezhető kártyákon kerülnek forgalomba. A kapható bővítések nagysága 512K. Ez a memóriaterület közvetlenül nem érhető el a BASIC-ból. Speciális parancsok segítségével magunknak kell mozgatni a memóriaterületeket. A memóriabővítő kártyát a számítógép **bekapcsolása előtt** kell a helyére illesztenünk. Ügyeljünk arra, hogy a kártya megfelelő oldala legyen felfelé! Helytelen csatlakoztatás esetén a kártya is, a gép is tönkremehet.

Ugyancsak a bővítőegységhez csatlakoztatandó kártyákon **kész programo**kat is vásárolhatunk. Ezek ugyanúgy épülnek fel, mint a memória bővítő kártyák, azzal a különbséggel, hogy a kártyán nem RAM, hanem ROM található. Ez a ROM tartalmazza a beégetett programot.

Két eltérő lehetőség van a C-128-as bővítőkártyák használatára. Az első esetben a rendszer belső bővítésnek tekinti a programot s annak megfelelően is kezeli. Ez azt jelenti, hogy a BASIC továbbra is elérhető, de egy speciális paranccsal (amit általában az <F1> lenyomásával adhatunk ki), a bővítmény program is használható. Két új paranccsal bővül automatikusan a BASIC: OFF, ami a bővítmény kikapcsolására szolgál és a QUIT, amivel a BASIC-be lehet visszatérni. Az eljárás hasonló ahhoz, ahogy a Commodore 4+ kezeli a beépített alkalmazói szoftvert.

Külsőnek tekintett bővítmény esetén a BASIC kikapcsolódik és csak a bővítmény program fut. A fentiek természetesen csak a C-128-as hardverű bővítményekre vonatkoznak. Ha egy C-64-es bővítő kártyát illesztünk a géphez, akkor az automatikusan C-64 üzemmódba tér át.

Külön szólunk a rendszerhez csatlakoztatható monitorokról. A kisebb Commodore gépek esetén ez nem volt probléma: egy jobb minőségű televíziós készülék is megtette. A C-128 azonban egyszerre két monitort is tud kezelni. Az egyiken 40, a másikon 80 oszlopos a kijelzés. A 40 oszlopos kijelzést a Commodore 64-en megismert VIC II segítségével valósítja meg a rendszer. A 80 oszlopos kijelzéshez egy új video chip került a gépbe. Ennek segítségével egy RGBI monitort lehet vezérelni. A Commodore erre a célra az 1901 és 1902 márkajelű monitorjait ajánlja. Mindkettő kombinált: egy kapcsoló segítségével 40 illetve 80 oszlopos kijelzésre állítható át. Monitorok nélkül csak a 40 oszlopos kijelzésű üzemmódot használhatjuk. A gépet a hozzá adott antennakábellel kell a televízióhoz csatlakoztatni.

A C-128 rendszeres használatához mindenképp célszerű valamilyen háttértárat beszerezni. Felhívjuk a figyelmet arra, hogy a C-128-hoz csatlakoztatható kazettás egységnek éppen úgy **DATASETTE** a neve, mint a C-16-hoz csatlakoztathatóé, de típusszáma eltér: **1530**. A C-16-hoz csatlakoztatható botkormányok a C-128-hoz nem jók (és persze fordítva sem).

A Commodore soros kimenettel rendelkező lemezegységei (VC1541, 4040, 1551, 1570, 1571) közvetlenül csatlakoztathatók a C-128-hoz, s ugyanez áll az ilyen típusu nyomtatókra (MPS801, 802, 803, SHEIKOSHA stb) is.

A CP/M üzemmódban lehetőség nyílik nem Commodore típusú nyomtatók használatára. A SETUP program segítségével a felhasználói kapura csatlakoztatott, CENTRONICS protokol szerint kommunikáló nyomtatók is közvetlenül üzemeltethetők. Nevezetesen valamennyi EPSON FX és RX típusú nyomtató használható!

1.5 Hogyan használjuk a könyvet?

A kézikönyv három jól elkülöníthető részre oszlik. Az elsőhöz a C-128-as üzemmóddal foglalkozó fejezetek, a másodikhoz a CP/M-ről szóló 4. fejezet, a harmadikhoz pedig a C-64-es üzemmódról szóló részek tartoznak. A gépet azonnal használni tudják azok, akik ismernek már valamilyen Commodore gépet (C-64, C-16, C-4+, VC-20), vagy

járatosak a CP/M-ben. Akik nincsenek ilyen előismeretek birtokában, azoknak azt ajánljuk, hogy először a C-128-as üzemmód használátát sajátítsák el. A C-128 - mint az az üzemmódokból is sejthető - három különböző számítógép, amelyet két, lényegesen eltérő processzor valósít meg. Ezek közül a C-64-es és a CP/M-re vonatkozó ismeretek egymagukban is pár ezer oldalt tesznek ki. éppen ezért ez a könyv **elsősorban a C-128-as üzemmódról** szól. A másik két üzemmódról is adunk azonban annyi információt, hogy használni lehessen őket. Akik azonban a részletekre is kíváncsiak, használják a könyv végén levő tárgymutatót, s keressék ki az ajánlott irodalomból a szükséges ismereteket.

Pár szót a jelölésekről. A Commodore szakirodalom a hexadecimális számjegyek jelzésére a $ jelet használja, míg a CP/M-es rendszerek ugyanerre a szám végén álló H betűt használják:

    $A0 és A0H egyaránt 160.

Abban az esetben, ha a **memória egy címét kerek zárójelek közé tesszük,** akkor arra utalunk, hogy az adot memória két egymást követő byte-ja tartalmazza az igazi címet. Például ($7A) az a memória cím, amelynek alsó byte-ja a $7A (122), felső byte-ja a $7B (123) memóriacímen található. A mutató értéke tehát:

```
MU=PEEK(122)+PEEK(123)*256
```

Gyakran lesz szükségünk bizonyos billentyűk jelölésére. Ilyen esetben a billentyűk nevét csúcsos zárójelek közé írjuk:

    <STOP>
    <ASCII/DIN>
    <ENTER>

Ha a váltókat is használni kell, azt a csúcsos zárójelen belül jelöljük. Ilyenkor **az első billentyű lenyomva tartása közben kell a második billentyűt lenyomni:**

    <STOP-RESTORE>
    <CTRL-M>
    <C=-A>

A CONTROL billentyű feliratát CTRL-lel rövidítjük! Hasonlóan az utasítások alakjának megadásakor az egyes részeket is csúcsos zárójelek közé tesszük, jelezve mely részeket lehet változtatni:

    GOTO <sorszám>
    ON <feltétel> **GOTO** <sorszámlista>
    INPUT "<szöveg>";<változó lista>

A mondottak szerint a vastagon szedett részek a nyelv alapszavai és jelei, amiket változtatás nélkül kell beírni, míg a többi rész változhat, esetleg el is maradhat. Az alábbi példákban még alá is húztuk a változtatható részeket:

    GOTO 123
    GOTO 2000
    INPUT "Kezdoertek:";X0

2. fejezet

A C-128-as üzemmód képernyő szerkesztője

2.1 A képernyő szerkesztő működése

Kijelzési módok

Ez a fejezet a C-128 személyi számítógép képernyő szerkesztőjét ismerteti. A C-64-es módban pontosan ugyanúgy dolgozhatunk, mintha a C-64-et magát használnánk. (Lásd C-64 2.fejezet 16-26. oldalak!) A CP/M üzemmódban sorszerkesztőt kell használni, erről bővebben a 4. fejezetben szólunk.

A C-128-as üzemmód egy **teljes képernyős szerkesztő**vel rendelkezik. Ez azt jelenti, hogy parancsok bevitelére a képernyő tetszőleges részét használhatjuk, innen az elnevezés. A képernyő szerkesztőt mind a 40, mind a 80 oszlopos képernyő esetén használhatjuk. Vannak azonban olyan lehetőségek, amelyekkel csak a 80 oszlopos képernyő esetén élhetünk. Ilyen pl. a karakterek villogtatása, vagy az aláhúzás. Van néhány - egyébként lényegtelen - funkció, ami a két képernyő használata esetén eltér. Ilyen például a <C=-SHIFT> billentyűzés. Ezeket leszámítva a két képernyőn egyformán szerkeszthetünk.

Az egyik képernyőről a másikra egyszerűen térhetünk át a <40/80 DISPLAY> billentyű átváltásával, majd a <STOP-RESTORE> leütésével. Ha az 1901/1902 monitorokat használjuk, akkor a monitort kézzel kell átkapcsolnunk. Ha két monitort használunk, a hatás természetesen azonnal jelentkezik.

A kijelzést még lényegesen befolyásolja az <**ASCII/DIN**> billentyű állapota. Lenyomott helyzetben a billentyűzet és a karaktergenerátor is a német szabvány szerint működik. Például a <Z> billentyűből <Y> lesz, a <[> billentyűből pedig <Ö>. A DIN kijelzés esetén a kijelzett karakterek sokkal vékonyabbak lesznek.

A billentyűzet felosztása

A C-128 billentyűzetén összesen 92 billentyű található. Ezek közül 88 a hardver szempontjából teljesen egyenértékű, csak az interpreter különbözteti meg őket. A baloldali <SHIFT> billentyű fölött egy **<SHIFT LOCK>** feliratú billentyű található. Ennek lenyomása blokkolja a (baloldali) <SHIFT> billentyűt, azaz ezt követően bármely billentyűt is nyomunk meg, az olyan, mintha a <SHIFT>-et is lenyomva tartottuk volna. A <SHIFT LOCK> újbóli megnyomása feloldja a zárt.

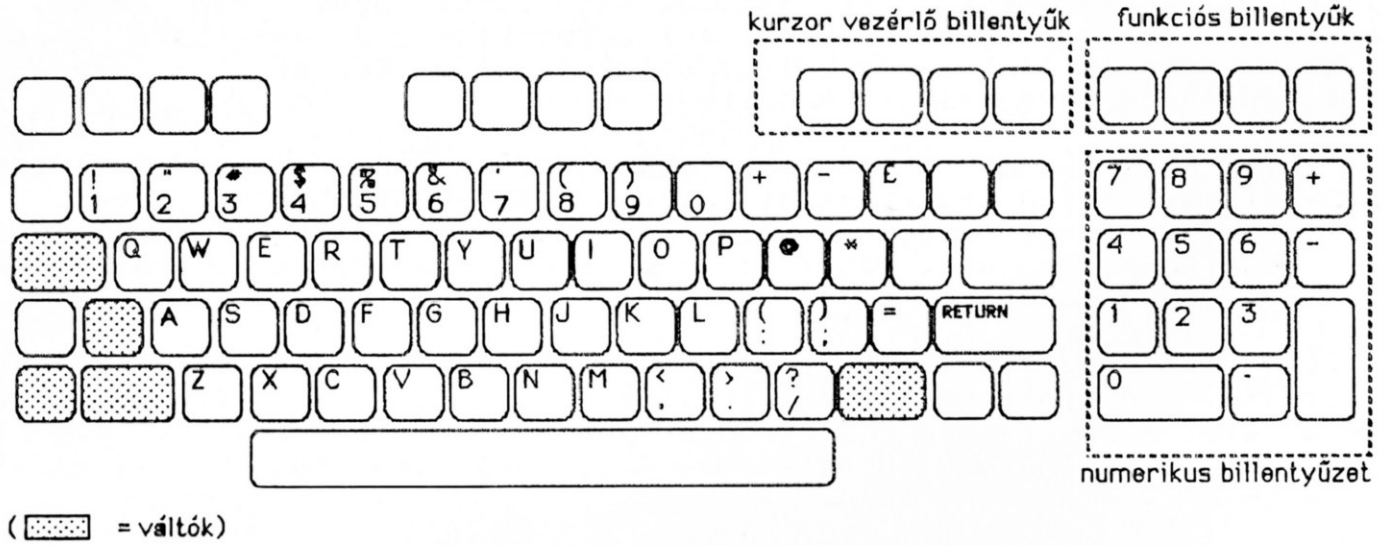
További speciális billentyűk: **<ASCII/DIN>, <40/80 DISPLAY>, <RESTORE>.** Az első kettő hatását már ismertettük. A <RESTORE> egy kondenzátoron keresztül egy nem maszkolható megszakítást generál, éppen úgy, mint a Commodore 64 esetén. Hatása csak a <STOP> billentyűvel együtt jelenik meg.

A billentyűzeten az első sorban, jól elkülönülten találjuk a funkciós és speciális billentyűket. Ezeket még az is megkülönbözteti, hogy **szürke színűek.** A billentyűzet jobb szélén egy tömbben a numerikus billentyűzet 14 billentyűjét (USA szabvány szerint rendezve, alul a 0-val) látjuk. A többi billentyű egyetlen tömbben helyezkedik el, ugyancsak az USA szabvány szerint. Alul egyetlen hosszú billentyű található, ez a **szóköz** (space) billentyű. Ennek a résznek a jobb, illetve bal oldalán helyezkednek el további, speciális hatású **vezérlő** billentyűk.

A négy úgynevezett **funkció** billentyű jobbra fent elkülönítve található meg (**F1/F2,F3/F4,F5/F6,** illetve **F7/F8** jelöléssel). A numerikus billentyűzetet és a funkcióbillentyűktől eltérő speciális (szürke) billentyűket leszámítva éppen egy C-64 billentyűzetet kapunk, bár a billentyűk elhelyezése kicsit eltér.

Az <F>, a <J> és a numerikus billentyűzet <5> feliratú billentyűinek közepén kis kiemelkedés látható. Ez a 'vakon' írók helyzetét könnyíti meg: a billentyűzetre nézés nélkül is ellenőrizhetik, hogy ujjaik jó helyen vannak-e.

A Commodore 128 billentyűzete



Valamennyi billentyű ismétli önmagát. Ez azt jelenti, hogy ha lenyomva tartjuk a billentyűt, akkor a neki megfelelő karakter folyamatosan a képernyőre kerül. Ez eltér a Commodore 64 géptől, amelyiken csak a kurzor vezérlő billentyűk és a <szóköz> ismétli önmagát.

Az információ (szövegek, betűk, számok) begépelésére - írógépen használt terminológiával élve - négy **váltó** is szolgál. Váltók használata nélkül az a karakter kerül a képernyőre, amelyik a billentyűn látható; vagy amennyiben két jel van a billentyűn, akkor azok közül az alsó. A DIN kijelzés használatakor a billentyűn vékonyabb betűtípussal felírt karakterek számítanak.

A <SHIFT> váltót használva (ami azt jelenti, hogy a <SHIFT> lenyomva tartása közben lenyomunk egy billentyűt!) a következők történnek.

Olyan billentyűk esetén, amelyekre csak két (vastag) jel van ráírva, a siftelés (emelés) a felső jelet eredményezi, a neki megfelelő karakter kerül a képernyőre. Ha a billentyűn egyetlen jel látható, akkor a siftelés a billentyű gombján **elől látható jobb oldali** karakter (grafikus jel) begépelését eredményezi. A váltók használatát bizonyos esetekben feltüntetjük, pl. <CTRL-RVSON>. Olyankor azonban amikor ez világos, elhagyjuk. Az előző példát <RVSON>-nak is írjuk.

A **<C=>** váltó használatával a billentyűk gombján **elől látható baloldali** karaktereket lehet begépelni.

A rendszer még egy harmadik, **<ALT>** feliratú váltót is tartalmaz. Alapértelmezésben ez nem változtatja meg a bevitt jelet, de segítségével lehetőség nyílik egy negyedik billentyűzet definiálására. (Lásd a függeléket!).

A <CTRL> és <C=> váltóknak még néhány speciális feladata is van, amelyekről később még szólunk.

A C-128 a képernyőn két, egyenként 256 jelből álló karakterkészletet tud megjeleníteni. Az egyiket '**nagybetűk/grafikus jelek**' karakterkészletnek hívjuk. Ha ezt használjuk, a betűket csak nagybetűs alakjukban tudjuk a képernyőre írni. A másikat '**kisbetűk/nagybetűk**' karakterkészletnek hívjuk, mert ebben az esetben a betűk kisbetűkként íródnak a képernyőre, míg a <SHIFT> váltót használva nagybetűket kapunk (ez az üzemmód felel meg az **írógépnek**).

Egyik karakterkészletről a másikra a **<SHIFT>** és a **<C=>** váltók **egyidejű** megnyomásával térhetünk át. A karakterkészlet váltó használata azonban letiltható, illetve újra engedélyezhető.

A karakterkészlet váltás másképpen valósul meg a 40, illetve a 80 oszlopos kijelzés esetén. A 40 oszlopos kijelzés esetén a képernyőre kiírt karakterek maguk is megváltoznak. Kisbetűből nagybetű, nagybetűből grafikus jel lesz, illetve fordítva. 80 oszlopos kijelzés esetén nem történik semmi. Ha azonban új karaktereket kezdünk begépelni, akkor a változás már érzékelhető. A 80 oszlopos képernyőn – szemben a 40 oszlopossal – **mindkét karakterkészlet egyszerre** megjeleníthető. (Ennek oka, hogy a videochip hardvere képzi az inverz karaktereket, míg a VIC II esetén azok alakját külön kell definiálni.)

A váltók hatását az egyes karakterkészletek esetén a következőkben foglalhatjuk össze:

nagy betűk/grafikus jelek		**kisbetűk/nagybetűk**
V = <V> = <SHIFT-V> = <C=-V>		v = <V> V = <SHIFT-V> = <C=-V>
3 = <3> # = <SHIFT-3>		3 = <3> # = <SHIFT-3>

Terminátor billentyűk

A legtöbb billentyű lenyomásának nincs közvetlen hatása a számítógép működésére. Hatásukra csak további karakterek íródnak a képernyőre, de számítás, adatátvitel, programsor írása valójában nem történik. Vannak azonban olyan billentyűk, amelyeknek lenyomásával kikerülünk a szerkesztő felügyelete alól, és a gép azonnal elkezd egy konkrét parancsot végrehajtani. Az ilyen billentyűket összefoglaló néven **terminátor** billentyűknek hívjuk. A terminátor billentyűk a következők:

<RETURN>	szerkesztés vége
<STOP-RESTORE>	BASIC meleg indítás
<RESET>	C-128 inicializálás
<RUN>	lemezről program indítása
<ASCII/DIN>	ASCII/DIN karakterkészlet váltás
<C=-CTRL>	karakterkészlet váltás

Az utolsó két terminátor billentyű lemyomása mindössze a kijelzés módját változtatja meg, a többieknek azonban 'komoly' funkciója van.

Talán a legfontosabb terminátor billentyű a **<RETURN>**. Hatására a szerkesztő átadja feldolgozásra a BASIC interpreternek azt a logikai sort, amelyben a <RETURN> megnyomásának pillanatában a kurzor volt. Ha ez számjeggyel kezdődött, akkor programsor lesz, és bekerül a memóriába a többi programsor közé. Ha nem, akkor parancsnak tekinti az interpreter, és megkísérli végrehajtani.

A numerikus billentyűzeten található <ENTER> billentyűnek a hatása pontosan megegyezik a <RETURN>-évei.

A <RETURN>-höz nagyon hasonló a funkciója a **<SHIFT-RETURN>** billentyűnek. Hatására a sor írása abbamarad, és a kurzor a következő sor elejére kerül. A beírt sor feldolgozása nem kezdődik meg, **továbbra is szerkesztő módban** dolgozik a gép.

Terminátor billentyű a **<RUN>** is. Megnyomása ekvivalens a

```
DLOAD "*"
RUN
```

parancssorozat kiadásával. (Az utasítások hatásának részletes leírása a 3.3 paragrafusban található meg.) Ennek megfelelően a <RUN> billentyű megnyomása után az interpeter a 8-as lemezegység Ø-ás meghajtójában levő első programot betölti a memóriába, majd elindítja. Ha tévedésből nyomtuk meg, akkor a <STOP> billentyű lenyomásával tudjuk a program töltését megszakítani. **Ne keverjük össze** a <RUN> billentyűt és a RUN parancsot. Az előbbi hatására a fentebb vázoltak történnek, és egyetlen billentyű megnyomását jelenti, míg a RUN parancsot betűnként kell begépelnünk, és utána még a <RETURN> billentyűt is meg kell nyomnunk!

A **<SHIFT-RETURN>** hatására a C-128 újra indítja a BASIC-et az ún. melegindítási pontról. A BASIC program és változók nem törlődnek, viszont a rendszer inicializálja a képernyő szerkesztőt. Speciálisan törlődik a képernyő is.

Nem tekinthető igazi terminátor billentyűnek, de itt szólunk a **<STOP>** billentyűről. Hatása csak akkor van, ha egy BASIC program futása közben nyomjuk meg. Ekkor **megállítja** a futó programot. A program ezek

után a CONT paranccsal tovább indítható. (Lásd a 3. fejezetben a STOP parancsot!) Ha szerkesztő üzemmódban nyomjuk le, semmilyen hatása sincs.

Valójában terminátor billentyűnek tekinthetők az <ASCII/DIN> és <40/80 DISPLAY> billentyűk, továbbá a <STOP-RESTORE> billentyűzés, amiről már szóltunk.

Az utolsó terminátor 'billentyű' valójában két billentyű egymás utáni lenyomását jelenti. Az első mindig az <ESC> feliratú '**escape**' billentyű. Ez után bizonyos billentyűk megnyomása speciális vezérlő funkciót tölt be. (Részletes ismertetésükre a 2.2 fejezetben kerül sor.)

Hasonlóan itt szólunk a **RESET nyomógombról**, ami a készülék jobb oldalán található. Benyomása a készülék ki-, majd újbóli bekapcsolásával egyenértékű. Használata tehát törli a gépben esetleg tárolt programot! A memória fizikai törlésére nem kerül sor, csupán a program elejét és végét jelző mutatók törlődnek. Ha pl. a memóriában gépi kódú program volt, az változatlanul megmarad.

Vezérlő karakterek

Bizonyos billentyűk megnyomásának nem az a hatása, hogy valamely karakter megjelenik a képernyőn. Ilyenek például a kurzor mozgató billentyűk. Ezeket a karaktereket összefoglaló néven **vezérlő karakter**eknek hívjuk. A következőkben ezekről lesz szó.

A kurzor mozgatása

A kurzor billentyűk teszik lehetővé, hogy a kurzort - a képernyőn látható információ megváltoztatása nélkül - tetszőleges helyre pozicionáljuk. Tizenegy olyan billentyű van, amelyeket ebbe a kategóriába sorolunk.

A **<CLEAR>** billentyű lenyomása **törli a képernyőt**, és a kurzort a bal felső sarokba viszi vissza. (Ez az ún. **home** pozició.) A **<HOME>** lenyomása nem törli a képernyőt, csak a 'home' pozicióba viszi a kurzort.

A **<CRSR JOBBRA>**, a **<CRSR BALRA>**, a **<CRSR FEL>** és a **<CRSR LE>** billentyűk a rajtuk levő nyilak irányának megfelelően mozgatják a kurzort, a képernyő tartalmának megváltoztatása nélkül. Abban az esetben, ha a képernyő alsó sorában nyomjuk meg a <CRSR LE> billentyűt, a szerkesztő a teljes képernyőt feljebb tolja és így alul egy üres sor keletkezik. A képernyő első sorában a <CRSR FEL>-nek nincs hasonló hatása, egyszerűen hatástalan. Ha a képernyő jobb vagy bal szélén lépünk át a <CRSR JOBBRA> vagy a <CRSR BALRA> billentyűvel, akkor a képernyő másik szélén, egy sorral lejjebb illetve feljebb lépünk vissza a másik oldalon.
A Commodore 64-től eltérően minden iránynak külön is megfelel egy szürke billentyű (rajta az irányt mutató nyíl). Ezen billentyűknek ugyanaz a hatásuk mint a jobboldali <SHIFT> gomb mellett levőknek, csak sokkal egyszerűbben használhatók (nem kell shiftelni). Ezekre a **<FEL>, <LE>, <BALRA>, <JOBBRA>** jelölésekkel hivatkozunk. (Vigyázat: a CP/M üzemmódban a kétféle kurzor vezérlő billentyűnek más és más szerepe van!)

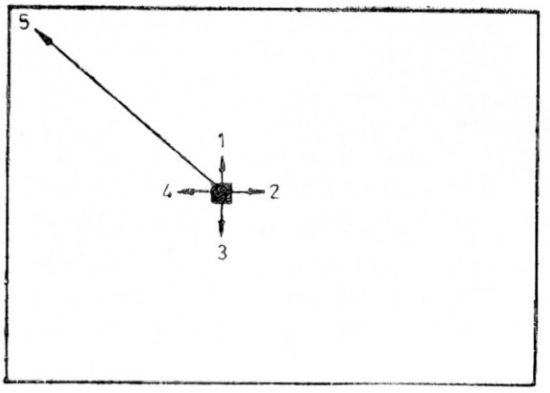
1 = <CRSR FEL> <FEL>

2 = <CRSR JOBBRA> <JOBBRA>

3 = <CRSR LE> <LE>

4 = <CRSR BALRA> <BALRA>

5 = <HOME>



Idézőjel üzemmód

Bizonyos feltételek mellett a képernyő-szerkesztő a vezérlő billentyűket szabályszerű üzemmódjuktól eltérően kezeli. Ezek egyike az ún. 'idézőjel' üzemmód.

Idézőjel üzemmódban, azaz megnyitott idézőjel után, a 'közönséges' karakterek szabályszerűen kiíródnak a képernyőre, **a vezérlő karakterek azonban nem hajtódnak végre.** Helyettük inverz karakterek íródnak ki, amelyek valójában a lenyomott vezérlő karaktereket **reprezentálják.** Ez lehetővé teszi, hogy a programokban a sztringek közé kurzor- és színvezérlő karaktereket tegyünk. Ennek köszönhető, hogy amikor az idézőjelben levő szöveg kiíródik a képernyőre, akkor a sztring részeként automatikusan végrehajtódnak a megfelelő vezérlő funkciók. Példaként gépeljük be a következő sort:

```
PRINT "<CLR><LE><LE><CTRL-BLK>A" <RETURN>
```

Amíg nem nyomjuk meg a <RETURN> billentyűt, nem történik semmi; a sztringbe egyszerűen grafikus jelek kerülnek. A <RETURN> megnyomása után az interpreter törli a képernyőt, és a képernyő második sorába egy fekete A betűt ír.

Idézőjel üzemmódban pl. a <CTRL-RVSON> sem hajtódik végre, hatására egy inverz R jelenik meg az idézőjelen belül. Az ezután beírt karakterek a sztring kiírása során inverz formában jelennek majd meg. Az inverz kiírás hatásának megszüntetésére a <CTRL-RVSOFF> billentyűt kell leütnünk. (Ennek hatására is egy inverz grafikus jel kerül a sztringbe.) Például:

```
10 PRINT "<CTRL-RVSON>KUTYAFULE<CTRL-RVSOFF>";
20 PRINT "KUTYAFULE"
```

A fenti rövid kis program kétszer írja ki a "KUTYAFULE" sztringet, először inverz formában, utána normálisan.

Idézőjel üzemmódban azok a karakterek is inverz alakú grafikus jelként íródnak a sztringbe, amelyeknek amúgy nincs is hatásuk. Például a <CTRL-A> hatására egy inverz alakú 'a' íródik ki.

A <DEL> billentyű (lásd lejjebb) az egyetlen vezérlő billentyű, amelyre az "idézőjel" üzemmód nincs hatással. így, ha egy hibát vétünk idézőjel üzemmódban, a <CRSR BALRA> billentyűt nem használhatjuk visszaléptetésre. E helyett a megkezdett sort a <SHIFT-RETURN> billentyű lenyomásával be kell fejeznünk.

A C-128-as üzemmódban az interpreter a $F4(244) címet használja annak rögzítésére, hogy idézőjel üzemmódban vagyunk-e, vagy sem. Ennek átírásával törölhetjük, vagy beállíthatjuk az idézőjel módot.

Karakterek törlése/beszúrása

A szerkesztő lehetőséget nyújt arra is, hogy egy-egy sorba karaktereket szúrhassunk be, vagy törölhessünk. Az **<INST>** billentyű lenyomása a kurzor alatt, illetve attól jobbra levő karaktereket egy hellyel jobbra tolja. így a kurzor alatt egy szóköz keletkezik, ahova új karaktert írhatunk be. A szerkesztő megjegyzi, hogy az <INST> billentyűt használtuk, és egész addig, míg ezeket a helyeket be nem töltöttük ún. 'inzert' vagy **beszúrási üzemmód**ban dolgozik. Az <INST> hatását a következő ábrán szemléltethetjük:

```
ABCDEFGHIKLMNO      <INST>
ABCDEFGHI KLMNO     <J>
ABCDEFGHIJKLMNO
          ↑
a kurzor helye
```

A C-128-as üzemmódban az interpreter a beszúrási üzemmód jelzésére a $F5(245) címet használja. Beszúrási üzemmód esetén ennek értéke $80, különben $00.

Beszúrási üzemmódban a kurzor és a színvezérlő karakterek ismét inverz karakterként íródnak ki (úgy mint idézőjel üzemmódban). Az egyetlen különbség az <INST> és <DEL> beszúró, illetve törlő billentyű hatásában van. A <DEL> ahelyett, hogy szabályosan működne, most inverz 'T'-ként íródik ki. Az <INST> billentyű, amely inverz karaktert hoz létre idézőjel üzemmódban, most szabályosan szóközt szúr be.

Ezt a jelenséget felhasználhatjuk olyan PRINT utasítás létrehozására, amelyik törlést (<DEL>) tartalmaz. (Ez idézőjel üzemmódban lehetetlen.) Az inzert üzemmódot a <RETURN>, <SHIFT-RETURN>, <RUN> billentyűk lenyomásával, vagy valamennyi hely betöltésével lehet megszüntetni.

Példa <DEL> karakterek használatára sztringben (így is kell billentyűzni!):

```
10 PRINT "HELLO" <DEL> <INST> <INST> <DEL> <DEL> P" <RETURN>
```

Ha ezt a példa-programot a RUN parancs kiadásával lefuttatjuk, a HELP szó fog kiíródni, mert az LO betűk törlődnek, mielőtt a P kiíródna. A sztringben levő törlő karakter hatása a LIST és PRINT utasítás esetén

egyaránt érződik. Ezzel a módszerrel el lehet 'rejteni' a sor egy részét vagy a program egy teljes sorát. Az ilyen sorok újraszerkesztése természetesen igen nehézkes.

Az eddig említetteken kívül is van még néhány olyan vezérlő karakter, amely speciális módon helyezhető csak el egy sztringkonstansban. Ahhoz, hogy ezeket beírhassuk, részükre üres helyeket kell hagyni a sztringben, majd lenyomni a <SHIFT-RETURN> billentyűt, és utána újból szerkeszteni a sort. A javítás megkezdése előtt nyomjuk le a <CTRL-RVSON> billentyűt. A sztringbe most már gépelhetünk inverz karaktereket is. (Az idézőjelet természetesen nem nyithatjuk meg újra. A szerkesztő nem kerül idéző jel üzemmódba, ha a kurzorral átlépünk egy idézőjelen. Ugyanígy, ha a megnyitott idézőjelet a <DEL> billentyűvel töröljük, a szerkesztő továbbra is idéző jel üzemben dolgozik! Az inverz alakú vezérlő karakterek a következők:

Funkció	**Billentyűzés**
<RETURN>	M
<SHIFT-RETURN>	<SHIFT> M
kisbetűk/nagybetűk	N
nagybetűk/grafikák	<SHIFT> N

A fenntartott helyre gépeljük be a megfelelő inverz karaktert, majd nyomjuk meg a <RETURN> billentyűt. A sztring kiírásakor a megfelelő vezérlő funkció hajtódik végre.

Írjuk be a következő sorokat:

```
10 REM " FOPROGRAM<SHIFT-RETURN>
<FEL>,<JOBBRA> ez utóbbit nyolcszor!
<CTRL-RVSON>,<SHIFT-M><RETURN>
```

A LIST 10 parancs a programsort a következő alakban listázza:

```
10 REM "
FOPROGRAM
```

Logikai sor

Mint már említettük, s mint ahogy ez azonnal látszik, a képernyőn egy sorba 40 vagy 80 karaktert lehet írni. Ezt a sort **fizikai** sornak hívjuk, mert a képernyő valódi méretét jelenti. A C-128 szövegszerkesztőjét úgy tervezték meg, hogy képes **több, egymás alatti** sort egyetlen **logikai sor**nak tekinteni. Két fizikai sorból akkor lesz egyetlen logikai sor, ha a képernyő jobb szélén - a kurzor vezérlő billentyű kivételével - valamilyen billentyűvel átírunk. Ilyen módon pl. - figyelembe véve a puffer hosszát is - a BASIC programsorok hossza maximum 160 karakter lehet. 40 oszlopos képernyőn 4, a 80 oszlopos képernyőn két sornyi lehet egy BASIC parancs vagy program sor.

Az interpreter a $035E(862) címen kezdődő négy byte-ot használja a sorok összekapcsolásának jelzésére. A négy byte bitjei balról-jobbra a képernyő sorainak felelnek meg. Ha valamelyik bit magas, a szóbanforgó

sor az alatta levővel egyetlen logikai sort alkot.

A képernyőn nem látszik, hogy mely egymás alatti sorok lettek logikai sorrá összekapcsolva. Ebből néha kellemetlenségek adódnak.

Az interpreter a logikai sor fizikai sorait a következő esetekben kezeli **egynek:**

a/ karakterek törlése/beszúrása;
b/ a képernyő fel/le történő görgetése;
c/ a sor feldolgozásra való átadása.

Az a/ esetben, ha a logikai sor első fizikai sorában törlünk egy karaktert, akkor a második fizikai sor is egy hellyel balra tolódik. Hasonlóan, ha beszúrunk egy karaktert, a második sor is egy hellyel jobbra tolódik.

Bizonyos esetekben (például ha a képernyő utolsó során túl írunk) a képernyő egy sorral felfelé vagy lefelé tolódik el. Sor alatt itt mindig **logikai** sor értendő. Ha a képernyőt felfelé kell eltolni, és az első két sor egyetlen logikai sort alkot, akkor fizikailag a képernyő két sorral tolódik el felfelé.

Az interpreter a c/-ben jelzettnek megfelelően a teljes logikai sort feldolgozza. írjuk be például a következőket:

A<SHIFT-RETURN><BALRA><Szóköz>?2+2

A kurzort tartalmazó sorban látszólag egy szintaktikusan helyes utasitás: ?2+2 áll. Ha most megnyomjuk a <RETURN> billentyűt, akkor mégis ?SYNTAX ERROR hibajelzést kapunk, mert az interpreter a sort **A?2+2**-ként értelmezi. A <BALRA> <Szóköz> billentyűzéssel ugyanis a két sor egyetlen logikai sorrá állt össze.

Színvezérlés

A C-128 számítógép 16 féle szint képes előállítani. A 16 féle szín a 40 és 80 oszlopos képernyő esetén némileg eltér, gyakorlatilag azonban azonosnak tekinthető. A képernyőre kerülő karakterek szinének beállitására a <CTRL> vagy a <C=> váltók és a színbillentyűk szolgálnak. A színbillentyűk azonosak az 1-8 számbillentyűkkel. A <CTRL> és <C=> váltók használatakor kapott szín az oldalukra van ráírva. A felső szín a <CTRL>, az alsó a <C=> váltónak felel meg. így például a <CTRL-1> billentyűzés hatására a kurzor fekete színű lesz. Az ezt követően beírt valamennyi karakter fekete színnel jelenik meg a képernyőn.

A kurzor színe a COLOR 1,<színkód> BASIC utasitással is beállitható.

Inverz karakterek

Normális esetben a képernyőre kerülő betűk pontjai lesznek a háttértől eltérő színűek. Lehetőség van azonban arra is, hogy a karakterhely maradék pontjai legyenek a betű színével megvilágítva, míg maga **a betű háttérszínű** maradjon. Úgy is mondhatnánk, hogy a betű negatívját írjuk ki. A <CTRL-RVSON> billentyű lenyomását követően valamennyi karakter inverz alakban íródik ki. Normál üzemmódra a <CTRL-RVSOFF> billentyű

leütésével térhetünk vissza. A fenti két billentyű leütésekor a képernyőn - látszólag - semmi sem történik. Ha azonban egy további karaktert leütünk, a hatás azonnal jelentkezik. Próbáljuk ki a

<CTRL RVSON> A <CTRL RVSOFF> A

beírását! Az inverz kijelzési mód használatát valamennyi terminátor billentyű (kivéve az ESC és a <SHIFT-C=> megszünteti.
Az inverz írásmód jelzésére a C-128 a $F3(243) címet használja. Ha inverz kijelzési módot használunk, ennek értéke $80, különben $00.

Karakterek villogtatása

80 oszlopos kijelzés esetén lehetőség van a képernyőre kerülő karakterek villogtatására. Az ilyen feliratok kiugranak a többiek közül, azonnal magukra vonják a figyelmet. Ha villogó karaktert akarunk a képernyőre vinni, akkor a <CTRL-O> billentyűzést kell használnunk. Ha már nem akarunk további villogó karaktereket a képernyőre vinni, akkor az <ESC-O> billentyűzést kell végrehajtanunk. Ez természetesen nem szünteti meg a villogást, de az új karakterek már normálisan íródnak a képernyőre. írjuk be például a következőt:

<CTRL-O>A<ESC-O>A

A képernyőre kerülő két 'A' betű közül az első villog, a második nem. (Csak 80 oszlopos képernyő esetén!)

További vezérlő karakterek

Mint már utaltunk rá, a CTRL váltó segítségével további vezérlő funkciók hajthatók végre. Az ezeknek a funkcióknak megfelelő karakterek idézőjel üzemmódban egy szövegkonstansba beírhatók, s az adott funkciók csak a sztring kiírásakor hajtódnak végre.

<CTRL-B>: aláhúzás bekapcsolása
80 oszlopos képernyő esetén lehetőség van a karakterek aláhúzására. A billentyűzés ezt az üzemmódot kapcsolja be. A kikapcsolása <ESC-O>-val!

<CTRL-G>: hangjelzés adása
A billentyűzés után rövid hangjelzés, 'beep' hallható. Ez a hatás az <ESC-H>-val letiltható. <ESC-G> újra engedélyezi.

<CTRL-I>: tabulálás
Hatása azonos a <TAB> billentyűvel. A kurzor a legközelebbi tabulálási pontra ugrik.

<CTRL-J>: soremelés
Hatása azonos a <LINE FEED> billentyű lenyomásával. A kurzor egy sort lefelé lép.

<CTRL-K>: karakterkészlet váltás letiltása
A <C=-SHIFT> billentyű lenyomására nem változik meg a karakterkészlet.

<CTRL-L>: karakterkészlet váltás engedélyezése
Újból engedélyezi a <C=-SHIFT> használatát (a karakterkészlet váltására).

<CTRL-X>: tabulátor ki/be kapcsolása
A kurzor helyén levő tabulátort be, vagy kikapcsolja. Segítségével tetszőlegesen beállíthatjuk a tabulálási pontokat.

<CTRL-[>: ESC
Az ESC sorozatok nem írhatók be idézőjel üzemmódban. Ha mégis azt akarjuk, akkor ESC helyett a <CTRL-[>-t kell billentyűzni. A sztringbe egy grafikus jel kerül, ami az ESC-t reprezentálja.

A fenti funkciók megvalósítására a C-128 interpretere további jelzőket használ. A $0354(852) címtől kezdődően 10 byte szolgál a tabulálási pontok jelzésére. Ha valamelyik bit 1-es, akkor a neki megfelelő helyen egy bekapcsolt tabulátor pont van. A <CTRL-G>, illetve a <C=-SHIFT> engedélyezésére a $F9(249), illetve $F7(247) regiszterek szolgálnak. Ha ezek értéke $80 az, illető funkció végrehajtható, ha $00, akkor nem.

A képernyő felosztása

Első pillantásra is látszik, hogy a képernyő két részre oszlik: egy téglalap alakú, az információt hordozó részre, illetve egy, az azt körülölelő egyszínű keretre. A középső részben 25 sorba, soronként 40, illetve 80 karakter írható. Speciális módon szabályozható azonban az, hogy ennek a területnek melyik - ugyancsak téglalap alakú - részét használja az interpreter. Ezt a részt hívjuk **ablak**nak. Az ablakok használata lehetővé teszi, hogy például a képernyő felső részét a program eredményeinek kiírására, alsó felét listázásra, illetve a program szerkesztésére használjuk. Lehetőség van arra, hogy a képernyő két felén ugyanannak a programnak két különböző részét listázzuk ki. Az 'ablak' definiálására és további szerkesztő műveletek végzésére az <ESC> billentyűt használhatjuk.

2.2 ESC sorozatok

Az <ESC> billentyű megnyomása után bizonyos billentyűkkel további vezérlő funkciókat hajthatunk végre, összesen 27-et. A Commodore 64 és a VC-20 gépek nem rendelkeznek ezekkel a lehetőségekkel, csak a CBM 600-as, illetve 700-as sorozatú gépei, illetve a Commodore C-16, C-116 és 4+ gépek. A továbbiakban ezeket a funkciókat ismertetjük.

<ESC> majd <A>: beszúrási üzemmód bekapcsolása
Alapállapotban a kurzor helyén levő karaktert **írja felül** az a jel, amit éppen begépeltünk. Beszúrási üzemmódban a teljes sor - a kurzor helyén levő karaktert is beleértve - egy hellyel jobbra tolódik, s az éppen benyomott billentyűnek megfelelő karakter a kurzor alatt levő, immár üres helyre szúródik be.

<ESC> majd <B>: az ablak jobb alsó sarkának beállítása
A kurzor aktuális helye lesz az ablak jobb alsó sarka.

<ESC> majd <C>: törli a beszúrási üzemmódot.

<ESC> majd <D>: sor törlése
A billentyűzés hatására törlődik a kurzort tartalmazó sor. A lejjebb levő sorok eggyel feljebb csúsznak. Alul egy üres sor keletkezik.

<ESC> majd <E>: álló kurzor
A kurzor egy nem villogó téglalap lesz.

<ESC> majd <F>: villogó kurzor
Normál üzemmódban a kurzor egy villogó téglalap. A kurzort azonban másképpen is jelezhetjük. A billentyűzés hatására újból egy villogó téglalap lesz a kurzor.

<ESC> majd <G>: megengedi a <CTRL-G>-t
A <CTRL-G> hatására egy rövid hangjelzést kapunk. Ez a funkció az <ESC-H> billentyűzéssel letiltható. A fenti billentyűzés újból engedélyezi a <CTRL-G> billentyűzést.

<ESC> majd <I>: sor beszúrása
A billentyűzés hatására a kurzort tartalmazó sortól kezdve valamennyi sor eggyel lejjebb csúszik. Az utolsó sor elvész.

<ESC> majd <J>: kurzor a sor elejére
A kurzor annak a sornak az első oszlopába kerül, amelyikben éppen áll.

<ESC> majd <K>: kurzor a sor végére
A kurzor annak a sornak az utolsó nem szóköz helyére kerül, amelyikben áll.

<ESC> majd <L>: képernyő görgetés bekapcsolása
Alapállapotban ha a képernyő utolsó során **túl** írunk, akkor az egész képernyő tartalma egy sorral feljebb csúszik. Lehetőség van ennek a módnak a megszüntetésére. Az <ESC-L> billentyűzés újra engedélyezi a képernyő görgetését.

<ESC> majd <M>: képernyő görgetés kikapcsolása
Ha az <ESC-M> billentyűzést követően a képernyő utolsó sora után írunk, a képernyő tartalma nem tolódik egy sorral feljebb, hanem e helyett az írás az **első** sorban folytatódik.

<ESC> majd <N>: normál üzemmód
80 oszlopos kijelzés esetén az inverz módot törli.

<ESC> majd <O>: egyszerre kikapcsolja az inverz karakterírási, aláhúzott, illetve villogtató funkciókat.

<ESC> majd <P>: sor törlése a kurzorig
A billentyűzés törli a sort egészen a kurzorig. A kurzor helyén álló karakter is elvész. A sor többi karaktere a helyén marad.

<ESC> majd <Q>: sor törlése a sor végéig
A kurzor helyén álló karaktertől a sor végéig valamennyi karakter törlődik.

<ESC> majd <R>: inverz kijelzés
A billentyűzés hatására inverz kijelzés módba kerül a rendszer. Csak 80 oszlopos képernyő esetén használható.

<ESC> majd <S>: téglalap kurzor
A vonalkurzor helyett ismét a megszokott téglalap kurzort használhatunk.

<ESC> majd <T>: ablak bal felső sarkának beállitása
Az ablak bal felső sarka a kurzor jelenlegi helyzete lesz.

<ESC> majd <U>: vonal kurzor
A kurzor - a szokástól eltérően - nem egy téglalap, hanem egy vonal lesz,.

<ESC> majd <V>: képernyő felfelé tolása
A képernyő tartalmát egy sorral feljebb tolja. Alul egy üres sor keletkezik.

<ESC> majd <W>: képernyő lefelé tolása
A képernyő tartalmát egy sorral lejjebb tolja. Felül egy üres sor keletkezik.

<ESC> majd <X>: 40/80 váltás
A kijelzési módot megváltoztatja. Ha eddig 40 oszlopos volt, most 80 oszlopos lesz; és fordítva.

<ESC> majd <Y>: tabulátorok kezdeti értéke
A tabulátorok kezdeti értékét állitja be. Ennek megfelelően minden tizedik pozícióra lehet tabulálni a <TAB> billentyű megnyomásával.

<ESC> majd <Z> tabulátorok törlése
Valamennyi tabulálási pontot törli.

<ESC> majd <@> képernyő törlése
A kurzortól törli a teljes képernyőt.

Ha egyszerre több ablakot is szeretnénk használni, akkor ezt a legegyszerűbben úgy érhetjük el, hogy egy rövid programot állandóan a memóriában tartunk, s az <F2> illetve <F3> billentyűhöz hozzárendeljük az egyik, illetve a másik ablak beállítását. Egy ilyen program a következő:

```
10 KEY 1,"RUN99"+CHR$(13)
15 KEY 2,"GOSUB30"+CHR$(13)
20 KEY 3,"GOSUB45"+CHR$(13)
25 END
30 ?CHR$(19)+CHR$(19)+CHR$(27)+"T";
35 FOR I=1 TO 11: ? CHR$(17);: NEXT I
40 ?CHR$(157)+CHR$(27)+"B"+CHR$(19);: RETURN
45 ?CHR$(19)+CHR$(19);
50 FOR I=1 TO 11: ? CHR$(17);: NEXT I
55 ?CHR$(27)+"T";
60 FOR I=1 TO 12: ? CHR$(17);: NEXT I
65 ?CHR$(157)+CHR$(17)+CHR$(27)+"B"+CHR$(19);: RETURN
99 REM
```

A programot töltsük be, majd adjuk ki a RUN parancsot. Ettől kezdve - amíg a program a memóriában van - az <F2> lenyomása után a képernyő felső, az <F3> lenyomása után az alsó felét használhatjuk képernyőnek. Ha programot akarunk írni csak a 100-nál nagyobb sorszámokat használhatjuk. Az <F1> billentyű ekkor is a megfelelő helytől indítja el a programot.

A 30.-40. sorokban az ablak bal felső sarkát a home pozícióra, a jobb alsó sarkát pedig a 12. sor 39. (illetve 79.) karakterhelyére állítjuk be. A 30., illetve a 45. sorban töröljük a beállított ablakot. A 45.-65. sorokban az ablak bal felső sarkát a 13. sor 1., míg jobb alsó sarkát a 25. sor 39. karakterhelyére állítjuk be a megfelelő ESC karakterek kinyomtatásával.

Programból a <40/80 DISPLAY> billentyű nem használható. Erre az ESC-X billentyűzés szolgál. A következő BASIC programrész hatására például a 40 és a 80 oszlopos képernyőre íródik az a szöveg, hogy 'En 40 oszlopos vagyok', illetve, hogy 'Be, en 80 oszlopos vagyok:

```
10 ?"En 40 oszlopos vagyok"
20 ?CHR$(27);"X";
30 ?"Be, en 80 oszlopos vagyok"
```

(Vigyázat: RUN parancs kiadásakor 40 oszlopos üzemmódban kell lenni!)

A PRINT utasításban szereplő vezérlő karaktereket a most vázolt módszernél sok esetben egyszerűbben lehet CHR$(B) alakban előállítani. Az alábbi táblázat összefoglalja a felhasználható vezérlő karaktereket:

Funkció	B értéke
<ESC>	27
<RETURN>	13
<SHIFT-RETURN>	141
kisbetűk/nagybetűk	14
nagybetűk/grafikák	142
<SHIFT-C=> letiltása	11
<SHIFT-C=> engedélyezése	12
Home	19
CLR	147
Kurzor le	17
Kurzor fel	145
Kurzor jobbra	29
Kurzor balra	157
Törlés	20
Beszúrás	148
Aláhúzás bekapcsolása	2
Aláhúzás kikapcsolása	130
Hangjelzés	7
Tabulálás	9
Soremelés	10
Tabulátor pont ki/be	25
Inverz karakterek	18
Normál karakterek	146
Villogtatás	15
Villogtatás kikapcsolása	143

Funkció	B értéke
F1	133
F3	134
F5	135
F7	136
F2	137
F4	138
F6	139
F8	140
Fehér	5
Piros	28
Zöld	30
Kék	31
Narancssárga	129
Fekete	144
Barna	149
Sárgászöld	150
Rózsaszín	151
Kékeszöld	152
Világoskék	153
Sötétkék	154
Világoszöld	155
Bíbor	156
Sárga	158
Encián	159

2.3 Eltérések a C-64 és C-16 képernyő szerkesztőjétől

A Commodore 128 személyi számítógép 128-as módjának szövegszerkesztője **'felülről kompatibilis'** a Commodore 64 és Commodore 16 képernyő szerkesztőjével. A leglényegesebb eltérés, hogy a Commodore 128-on két képernyőt használhatunk.

További különbség, hogy milyen hosszú programsorok vihetők be a gépbe. A Commodore 64 esetében maximum két fizikai sor alkothatott egy logikai sort, a C-16 esetében akárhány. A sor hosszának azonban korlátot szabott az input puffer mérete. Ez a VC-20-tól öröklötten, 88 karakter volt. A C-128 esetében az input puffer hossza 160, így ez szab határt a beírható sor hosszának.

Az ablakokat a C-16-on ugyanúgy lehetett használni, mint a C-128-on. A Commodore 64-en nem lehet ablakokat használni.

A C-16-on is villogtathatók a karakterek, de máképpen, mint a C-128-on. Ott a <CTRL-FLASHON> és a <CTRL-FLASHOFF> karakterek szolgálnak a villogás ki/be kapcsolására. Ezek ASCII kódja eltér a C-128-on használtaktól.

Mind a C-16, mind a C-128 lehetővé teszi, hogy a funkció billentyűkre tetszőleges szöveget definiáljunk. A funkció billentyű megnyomása ekvivalens a szöveg begépelésével. Vigyázzunk, a két gépen eltérőek az egyes billentyűkhöz rendelt szövegek. A C-64-en ez a lehetőség egyáltalán nincs meg.

Az <ESC-R> billentyűzésnek egészen más a jelentése C-16-on, mint C-128-on. A C-64-en egyáltalán nincs lehetőség ESC sorozatok használatára.

A C-16 esetében lehetőség van a színek intenzitásának beállítására. Ez a C-64 és C-128 esetén egyáltalán nincs meg.

A fenti eltérések is mutatják, hogy a felülről kompatibilis jelző idézőjelbe tétele egyáltalán nem indokolatlan. Az eltérések nem is okoznak problémát ha pl. programot gépelünk be. Ha azonban egy olyan C-16-on megírt programot futtatunk a C-128-on, amelyik kihasználja a C-16 képernyő szerkesztőjének sajátosságait, akkor működési zavarok támadhatnak.

3. fejezet

C-128 BASIC

3.1 BASIC: összefoglalás

3.1.1 A C-128 BASIC felépítése

A C-128 személyi számítógép a **BASIC V7.0**-ás jelű változatát tartalmazza. Ez a változat '**felülről kompatibilis**' a BASIC V2.0-ás és V3.5 változataival, amelyikkel a Commodore 64, illetve Commodore 16 dolgozik. Az új változat lényeges bővítéseket tartalmaz. A leglényegesebbek a grafikus és zenei utasítások, amelyek a C-64-en egyáltalán nincsenek meg. A lemezegység kezelését is további BASIC utasítások segítik. A BASIC V7.0 tartalmazza a BASIC V4.0-ás változat lemezparancsait is. A BASIC V4.0-ás programok mégsem futtathatók közvetlenül, mert a V7.0-ás változát másképpen tokenizálja a lemezegységre vonatkozó parancsokat. Végeredményben a C-128 BASIC-je a BASIC V2.0-ás változathoz képest mintegy 100 új utasítást tartalmaz.

A C-128 személyi számítógép programozásához az alábbi - a gépbe beépített - programkomponensek állnak rendelkezésre:

a/ képernyő szerkesztő;
b/ BASIC interpreter;
c/ a perifériák kezelését végző monitor;
d/ gépi kódú monitor.

A képernyő szerkesztőről részletesen a 2. fejezetben szóltunk. Segítségével a képernyő bármelyik sorát átadhatjuk feldolgozásra az interpreternek. Az interpreter azt a sort dolgozza fel, ahol a kurzor állt, amikor a <RETURN>-t megnyomtuk.

A d/ alatt említett gépi kódú monitorról részletesen szólunk a 7. fejezetben.

A BASIC interpreter két részből áll: egyrészt a programok szerkesztését biztosító **programszerkesztő**ből, másrészt a BASIC **futtató rendszer**ből. Ebben a paragrafusban a BASIC programozási nyelvet ismerők számára foglaljuk össze a C-128 BASIC interpreter legfontosabb jellemzőit.

A programokat a

RUN <sorszám>
RUN <programnév>
GOTO <sorszám>
GOSUB <sorszám>
BOOT
BOOT <programnév> parancsokkal, vagy a **<RUN>** billentyű lenyomásával indíthatjuk el. A GOTO, illetve GOSUB parancsok nem törlik a változók értékét, a többiek igen. A BOOT <programnév> alak lehetővé teszi

bináris programok betöltését és elindítását, hasonlóan a BASIC programokhoz. Lehetőség van BASIC programokat **nyomkövetési üzemmódban** elindítani. Ebben az esetben a végrehajtott utasítások sorszámát az interpreter folyamatosan kiírja a képernyőre. A nyomkövetést a **TRON** utasítással kapcsolhatjuk be, illetve a **TROFF** utasítással kapcsolhatjuk ki. A fenti két utasítást a programban is bárhol használhatjuk.

Lehetőség van hibakezelő rutin beépítésére. A hibakezelő rutin belépési pontját a

TRAP {<sorszám>}

utasítás segítségével adjuk meg. A hibakezelő rutinból a **RESUME** utasítással térhetünk vissza. A hibát okozó sor számát az **EL**, a hiba kódját az **ER** fenntartott változók tartalmazzák. A hiba szövegét az **ERR$** tömb elemeinek a segítségével írathatjuk ki. N maximális értke 41 lehet. Ha nem használunk hibakezelő rutint, és a program valamilyen hiba miatt megszakad, a **HELP** parancs kiadása (vagy a <HELP> billentyű lenyomása) után az interpreter kiírja a hibát okozó sort. A sornak a hibát okozó, feldolgozhatatlan része inverz, illetve aláhúzott alakban jelenik meg.

A futó program a **STOP** vagy az **END** parancs végrehajtásakor áll meg, a program változói **nem törlődnek.** Mindkét esetben a program a **CONT** parancs kiadásával folytatható.

A programszerkesztő hagyományos. A BASIC sorok tetszőleges sorrendben beírhatók, a törlést egy üres sor beírásával érhetjük el. Az **AUTO** parancs automatikus sorszámozást ad. Egy meglévő sor újraírása a szóban forgó sor módosítását eredményezi. A program szövegét a **LIST** paranccsal listázhatjuk ki. A teljes programot a **NEW** paranccsal törölhetjük. Egynél több programsort a **DELETE** parancs segítségével törölhetünk. A **RENUMBER** paranccsal a programot újrasorszámozhatjuk. A parancs a megfelelő GOTO, GOSUB stb. hivatkozásokat is módosítja.

A **KEY** utasítás segítségével a funkcióbillentyűkhöz szövegeket rendelhetünk. A funkcióbillentyű lenyomása a hozzárendelt szövegrész beírásával ekvivalens.

A kész programokat a **SAVE, DSAVE** utasítás segitségével menthetjük kazettára, illetve lemezre. A **BSAVE** utasítás segítségével a memória egy adott részét is elmenthetjük. A tárolt programot a **LOAD**, illetve **DLOAD** utasítás segítségével tölthetjük be a memóriába. A **BLOAD** utasítás segítségével a PRG típusú file-t a memória tetszőleges részébe betölthetünk.

Magának a C-128 BASIC-nek két érdekessége van. Az egyik, hogy a perifériákra vonatkozó utasítások valamennyi perifériára ugyanazok. (Nincs például külön LPRINT utasítás. A nyomtatóra is a PRINT utasítás segítségével írhatunk.) Másik, hogy nem tesz szigorú különbséget parancs és utasítás között.

A C-128 BASIC nem teszi kötelezővé az értékadó utasításban a LET használatát. A **változók neve** akármilyen hosszú lehet, de megkülönböztetésükre csak az **első két karaktert** használhatjuk. Felismeri az **egész**, a **valós** és a **sztring típus**okat. Az egész, illetve a sztring típusokat a név után írt **%**, illetve **$** karakterek jelzik. A tömbváltozóknak maximum 255 indexe lehet. A tömböket a **DIM** utasításban

kell definiálni. Az egyes indexek értéke 0-tól a DIM utasításban megadott értékig terjedhet. A tömböket dinamikusan is lehet deklarálni.

A szorzás, osztás jele: *, illetve /. A hatványozás jele a felfelé mutató nyíl: ↑. Logikai kifejezésekben a <, >, <=, >=, =, <> relációs jelek használhatók.

Az egyetlen feltétel nélküli vezérlésátadó utasítás a **GOTO** , ami után csak egy **sorszám** állhat (kifejezés nem). Feltételes vezérlésátadó utasítások az

IF...THEN...{:ELSE...}
ON...GOTO...
ON...GOSUB... utasítások.

A ciklusutasítás

FOR...TO...{STEP}...

alakú. A ciklusváltozó csak **valós** lehet! A ciklusok tetszőleges mélységben **egymásba skatulyázhatók.** A lépésköz lehet negatív is.

Lehetőség van

DO...LOOP

alakú ciklusok használatára is. A DO, illetve a LOOP utasításokban **UNTIL <logikai kifejezés>**, illetve **WHILE <logikai kifejezés>** alakú utasításokat használhatunk a kilépés feltételének megadására. A ciklusból, annak lejárta előtt, az **EXIT** utasítás felhasználásával léphetünk ki.

A C-128 BASIC lehetőséget biztosít programzárójelek használatára. Ezek a **BEGIN** és **BEND** kulcsszavak. Segítségükkel az IF...THEN...:ELSE... részek több sorból is állhatnak.

Alprogramok használatát a **GOSUB, RETURN** utasításpár teszi lehetővé. Paraméterek átadására nincs lehetőség. A szubrutinhívások tetszőleges mélységben egymásba skatulyázhatók.

A C-128 **ki-/bemeneti utasítás**ainak használatához ismerni kell az ún. **elsődleges kimeneti**, illetve **bemeneti eszköz** fogalmát. Elsődleges bemeneti eszköz mindig a **billentyűzet.** Innen várja az inputot a képernyő szerkesztő és az INPUT utasítás. Az elsődleges beviteli eszközről bevitt adatok a képernyőn mindig megjelenhetnek. Az elsődleges kiviteli eszköz a képernyő. A rendszerüzenetek és a PRINT utasításban adott mennyiségek mindig ide íródnak ki. Az elsődleges kiviteli eszköz tetszőleges megnyitott file-ba átirányítható a **CMD** utasítás segítségével. Programhiba után az elsődleges kimeneti eszköz újra a képernyő lesz.

Egykarakteres beviteli utasítás a **GET** és a **GETKEY.** A GET hatására a billentyűzeten éppen lenyomott billentyű ASCII kódja a GET-et követő változóba kerül, mint egy egyhosszúságú sztring. Ha nincs benyomva egyetlen billentyű sem, akkor a változó értéke az üres sztring lesz. A GETKEY ettől abban különbözik, hogy az interpreter **addig vár**, míg le nem nyomunk legalább egy billentyűt.

Az elsődleges kimeneti eszközre való kiírásra a **PRINT** utasítás szolgál. A kiírandó mennyiségek elválasztásánál a vessző (,) és a pontosvessző (;) hatása a szokásos. Ugyancsak használhatók ezek a PRINT utasítás végén. Ha egy PRINT utasítás nem vesszőre vagy pontosvesszőre végződik, az utasítás egy CHR$(13) jelet is kiír, így a következő PRINT már egy új sorba nyomtat. Lehetőség van a PRINT USING utasítás használatával értékek megformázott kiírására is. A kiírást vezérlő karakterek a következők:

#	értékes számjegy vagy karakter
=	középre igazít
>	jobbra igazítás
-	előjel helye
.	tizedespont helye
,	elválasztójel
$	dollárjel
↑↑↑↑	exponens helye

Adatbevitelre az **INPUT** utasítást használhatjuk. Az input sor - a puffer mérete miatt - legfeljebb **160 karakter**ből állhat. Nincs INPUTLINE utasítás. Ha INPUT utasítással sztringet olvasunk be, az idézőjelet csak akkor kell kiírnunk, ha a sztring elválasztó jeleket (,;:) is tartalmaz. Az INPUT utasítás segítségével idézőjelet tartalmazó sztringet nem tudunk beolvasni.

Adatoknak a programban való elhelyezésére a **DATA** utasítást használhatjuk. A DATA-ban elhelyezett sztringeket csak akkor kell idézőjelbe rakni, ha tartalmaznak elválasztó jeleket (,;:), vagy grafikus karaktereket. A DATA-ban elhelyezett adatokat a **READ** utasítás segítségével olvashatjuk be. A **RESTORE** utasításnak lehet paramétere, ezért az olvasást tetszőleges DATA utasítástól folytathatjuk.

A file-ok használatára egységesen az **OPEN, CLOSE, INPUT#, GET#** és **PRINT#** utasítások szolgálnak. A file-ok használatához bizonyos azonosítókat kell megadnunk. Ezek a **logikai file-szám**, a fizikai **hardver szám** és a **másodlagos cím** (vagy megnyitási mód). Minden egyes perifériának rögzített hardver száma van. A másodlagos cím jelentése már egységről egységre változik. A logikai file számot az OPEN, DOPEN utasításban adjuk meg, s attól kezdve a programban vagy parancsban ezzel azonosíthatjuk a file-t.

A lemezegység kezelésére speciális utasítások szolgálnak. A **DIRECTORY**, illetve a **CATALOG** a képernyőre listázza a lemez katalógusát. A **DCLEAR** utasítás inicializálja a lemezegységet. A **BACKUP**, illetve a **COPY** utasítások lehetővé teszik a teljes lemez, illetve egyes file-ok másolását. A **CONCAT** utasítással szekvenciális fileokat fűzhetünk össze. A **SCRATCH** parancs segítségével file-okat törölhetünk a lemezről. A **HEADER** paranccsal a lemezeket formázhatjuk meg. A **COLLECT** parancs lehetővé teszi a lemez feleslegesen foglalt blokkjainak felszabadítását. Lemezes file-ok megnyitására a **DOPEN**, illetve az **APPEND** utasításokat használhatjuk. Relativ file esetén a **RECORD** utasítással közvetlenül beállíthajuk az írni/olvasni kívánt rekordot.

A fenti, a lemezegység használatát megkönnyítő utasítások a Commodore BASIC 4.0-ás változatának utasításai, s mind szintaktikailag, mind szemantikailag megegyeznek azokkal. Egyetlen eltérés: a C-128 eltérően tokenizálja ezeket. Nem lehet tehát egy BASIC 4.0-ás programot a lemezről betölteni: újra be kell gépelni.

A BASIC V7.0 teljesen új elemei a grafikus utasítások. Legfontosabb a **GRAPHIC**, amely segítségével kiválaszthatjuk a kívánt grafikus kijelzési módot. A **COLOR**-ral állíthatjuk be a keret, a háttér stb. színeit. A **SCNCLR** utasítás felhasználásával törölhetjük a teljes képernyőt. A **SCALE** a nagyfelbontású képernyő léptékét állítja be. A SCALE 0 kiadása esetén a képernyő léptéke megegyezik fizikai méreteivel, vagyis 320*200-as. A SCALE 1 parancs kiadása után az interpreter 1024*1024-esnek tekinti a képernyőt, s ennek megfelelően számítja ki az egyes pontok fizikai koordinátáit.

A grafikus kurzor mozgatására a **LOCATE** utasítás szolgál. A **DRAW** parancs segítségével rajzolhatunk pontokat, illetve egyenes szakaszokat. A **BOX** utasítás (tele) téglalapok rajzolására szolgál. A **CIRCLE** segítségével egy ellipszis adott darabját rajzolhatjuk meg. Speciális függvények tájékoztatnak a grafikus kurzor helyéről, a rajzolás színéről stb. A C-128 egy kiváló minőségű **PAINT** utasítással rendelkezik, amelyik segítségével a képernyő egyes részeit **kifesthetjük, besatírozhatjuk**. Elképzelhető, hogy a grafikus képernyő valamelyik darabját szeretnénk tárolni. Erre szolgál az **SSHAPE** BASIC utasítás, amelynek segítségével a grafikus képernyő egy téglalap alakú darabját átírhatjuk egy sztringváltozóba. A sztring tartalmát a **GSHAPE** utasítás segítségével azután akárhová visszahelyezhetjük. Ezek a parancsok megegyeznek a C-16 BASIC V3.5-ös parancsaival.

A C-128 teljes mértékben biztosítja a sprite-ok (8 db. 21x24 pontból álló képelemek) kezelését. A **SPRDEF** utasítás hatására belépünk a sprite-szerkesztőbe, ami lehetővé teszi a sprite-ok alakjának szerkesztését. A Commodore-64 esetén ez az egyik legnehezebb feladat. Mind a **normál, mind a többszínű** sprite-ok alakja megszerkeszthető. A sprite-ok alakját sztring változókban tárolhatjuk, illetve onnan is megadhatjuk. Erre való a **SPRSAV** utasítás. A sprite-ok tulajdonságainak definiálására a **SPRITE** parancs szolgál. Ezzel a sprite összes paraméterét beállíthatjuk. A többszínű sprite-ok színét a **SPRCOLOR** utasítással állíthatjuk be. Az egyes sprite-ok jellemzőinek lekérdezésére speciális függvények szolgálnak. A sprite-okat a **MOVSPR** utasítással mozgathatjuk. A mozgatást a megszakító rendszer végzi, a BASIC program közben fut tovább. A **BUMP** függvénnyel folyamatosan lekérdezhetjük, nem ütköztek-e a sprite-ok egymással vagy a háttérrel. A **COLLISION** utasítás kiadása után a megszakító rendszer figyeli a sprite-ok ütközését, s az ütközés bekövtkeztekor az utasításban megadott sorszámú sorra kerül a vezérlés. A megszakításból RETURN-nel térhetünk vissza.

A C-128 hanggenerátorának bekapcsolása a **VOL** utasítás kiadásával történik. A megszólaltatandó hangok tulajdonságait az **ENVELOPE** utasítással adhatjuk meg. A hangok szűrési paramétereinek beállítására a **FILTER**-t használhatjuk. Ezt követően a **SOUND** segítségével szólaltathatjuk meg az egyes hangokat. Egész dallamok lejátszására szolgál a **PLAY** utasítás. Hasonlóan a MOVSPR utasításhoz, a dallamot a megszakító rendszer játsza. Az ütem a **TEMPO** utasítással állítható.

BASIC programból valamennyi - a C-128-hoz csatlakoztatható - kiegészítő berendezés állapotáról információt kaphatunk. A **JOY**, **PEN** és a **POT** függvények a botkormányok, a fényceruza, illetve a potméterek aktuális értékét adják vissza.

A gépi kódú monitorba a **MONITOR** BASIC parancs kiadásával léphetünk be. A monitor parancsai egykarakteresek, a paramétereket hexadecimális alakban kell megadnunk. A monitor lehetőségeiről a 7. fejezetben

szólunk részletesen. A gépi kódú programozást több parancs támogatja. Ezek egyike a jól ismert **SYS** utasítás. A SYS paramétereiben azonban átadhatjuk a futó gépi kódú rutinnak az A,X,Y regiszter értékeit. Ezeknek a tartalmát az **RREG** utasítás segítségével kiolvashatjuk.

A Commodore 128 memóriabővítéséhez kapcsolódó parancsok már a BASIC V7.0-ba is beépültek. A **STASH**, **FETCH** és a **SWAP** utasításokkal lehetőség van a BASIC munkaterület adott részének elmentésére, visszatöltésére és cseréjére a memóriabővítés adott lapjára.

3.1.2 A BASIC interpreter működése

A C-128 képernyő szerkesztője a <RETURN> billentyű benyomását minden esetben úgy értelmezi, hogy befejeztük a BASIC parancs vagy programsor bevitelét, és átadja a vezérlést a BASIC interpreternek. Az interpreter első lépésként meghív egy rutint, amelyik a billentyűzetről bevitt utolsó sort (amely jelenleg a képernyő-memóriában található meg) áttölti az input pufferbe. Erre azért van szükség, hogy a ?"<CLR>EREDMENYEK=" típusú utasítások végrehajthatók legyenek. Ha az interpreter az utasításokat a képernyő-memóriából venné ki, a fenti utasítás a <CLR> végrehajtása után elveszne. (Ez a probléma természetesen nem merül fel abban az esetben, ha a képernyő egy meghatározott része - például az utolsó sor - szolgál a parancsok, programsorok bevitelére. Ilyen a ZX-81 rendszere.) Az input puffer a 2. lapon helyezkedik el a $0200-$02A1 címeken. A fenti rutin az inputsor végére egy 0 byte-ot is elhelyez. Miután az interpreter a bevitt sort elhelyezte az input pufferbe, megvizsgálja, hogy számmal kezdődik-e, vagy sem.

Ha igen, akkor az input sort programsornak értelmezi, és elhelyezi a memóriában. Ha nem, akkor parancsnak tekinti, és végrehajtja.

Nézzük először azt az esetet, amikor parancsot vittünk be. Az interpreter ennek felismerése után meghívja a **tokenizáló rutint**. Ennek a rutinnak a feladata, hogy az input pufferben található sorban megkeresse a BASIC alapszavakat, ezeket token-jükkel helyettesítse. Ez a rutin ismeri fel a kulcsszavak rövidítését, a ? utasítást és a <pi>-t. Ügyel arra, hogy páros számú idézőjel után szabad csak az alapszavakat tokenjükre cserélni. Az inputsor kódolása általában a sor rövidülését eredményezi.

Ezután az interpreter belép abba az ellenőrző ciklusba, amelyik a tokenizált BASIC szövegeket végrehajtja. Ez a belépési pont nem egyezik meg azzal, ahová a RUN utasítás után lép be az interpreter.

Ha az inputsor programsornak bizonyult, először a sorszám 2-byte-os egész számmá való konvertálására kerül sor. Ha ez hibát eredményez (a szám nagyobb 64000-nél) az interpreter hibajelzést küld, és visszatér a szerkesztőbe. Ha nem, sor kerül a maradék sor tokenizálására az előbb már említett rutin segítségével. Az interpreter ezután ellenőrzi, szerepel-e ilyen sorszámú sor a programban. Ha igen, akkor ezt törli a memóriából és az utána szereplő sorokat lejjebb mozgatja. Ezután megvizsgálja, van-e elég hely az új sor elhelyezésére. Ha nincs, a rutin végrehajtása ?OUT OF MEMORY ERROR hibaüzenettel megszakad, és a vezérlés visszaadódik a képernyő szerkesztőnek. Ha van hely

és az inputsor nem üres, az interpreter a megfelelő helyen annyival tolódik feljebb a memóriában, hogy a sor éppen beférjen. Ilyen módon természetesen a programsorok elején álló - és a következő sor elejére mutató - számok elromlanak. Ezért kerül sor egy további rutin végrehajtására, amelyik újraszámítja ezeket a mutatókat.

Ha az inputsor üres (tehát egyedül a sor végét jelző Ø byte-ból áll) csak a mutatók újraszámítására kerül sor. Ezért egy üres sor pl. 213 <RETURN> bevitele a sor törlését eredményezi.

Utoljára hagytuk a BASIC-kulcsszavak végrehajtását ellenőrző ciklus ismertetését. Két **belépési pont**ja van. Az egyik, amikor parancsot hajtunk végre. A másik belépési pont, amelyen keresztül a RUN parancs végrehajtása során lép be az interpreter. A rutin ellenőrzi, nincs-e lenyomva a <STOP> billentyű, majd sor kerül a következő BASIC-byte olvasására. Megvizsgálja, hogy ez nem sor végét jelző nulla-e. Parancs módban ez a ciklusból való kilépést, a képernyő szerkesztőbe való visszatérést eredményezi. Program végrehajtása közben a nulla a következő sor elején álló mutató ellenőrzését jelenti. Ha ez Ø, akkor END nélkül ért véget a program, és a vezérlés visszatér a szerkesztőbe. Ha nem, akkor a program végrehajtása egy új sorral folytatódik. Ha bekapcsoltuk a nyomkövetést, akkor az éppen végrehajtás alatt álló programsor száma kiíródik a képernyőre, ha nem, a végrehajtás azonnal megkezdődik.

Ha nem nulla byte-ot talált az interpreter, akkor megvizsgálja a következő karaktert, amely

a/ kettőspont;
b/ ASCII karakter;
c/ token;
d/ egyéb

lehet. Minden egyes esetben a rutin egy belépési pont címét helyezi a verembe. A belépési pont annak a tevékenységnek az elejére mutat, amit az adott esetben végre kell hajtani. Az a/ esetben nincs tennivaló. A b/ esetben az interpreter LET utasítást tételez fel; s az ennek megfelelő belépési pontot tölti a verembe. A c/ esetben a BASIC elején levő táblázatból megkeresi az ahhoz a tokenhez tartozó belépési pontot, és ezt tölti be a verembe. A d/ esetben a ?SYNTAX ERROR üzenetet kinyomtató hibarutin kezdőcíme töltődik a verembe.

A fenti rutin végén az interpreter egy újabb (gépi kódú) RTS utasítást hajt végre, ami a verembe töltött belépési pontra való ugrást eredményezi.

A belépési pontoknak megfelelő rutinok végén - hacsak nem történt hiba - a vezérlés az ellenőrző ciklus legelejére adódik vissza, függetlenül attól, hogy direkt vagy program módban vagyunk-e.

Külön szólunk a RUN végrehajtásáról. A RUN utasítás végrehajtása egy mutatónak a BASIC szöveg megfelelő helyére való állításával kezdődik. Ezután kerül csak a vezérlés az elenőrzési ciklusba.

A BASIC interpreter működésének ismerete elengedhetetlen a BASIC lehetőségeinek kiterjesztéséhez. A fent jelzett rutinok ugyanis általában egy JMP ($xxxx) gépi kódú utasítással kezdődnek, ahol $xxxx egy-egy RAM címet jelent (általában a 3. lapon). A címek - hacsak nem írtuk át őket - közvetlenül az indirekt ugrás alá mutatnak vissza, s

így folytatják a program végrehajtását. Ugyanakkor a programozónak lehetősége nyílik ezeknek a legfontosabb eljárásoknak az átírására.

Végül pár szót szólunk a C-128 számítógép megszakító rendszeréről, amelyik közvetlenül támogatja a BASIC interpretert.

A számítógép bekapcsolása után, amikor a tápfeszültség eléri a 4.75 V-ot, a számítógép automatikusan generál egy **RESET** jelet. Ennek hatására a programszámláló a ($FFFC) mutató értékével töltődik fel, előtte azonban a maszkolható megszakításokat a gép letiltja. A JMP ($FFFC) végrehajtása az I/O regiszterek feltöltésével, majd a legfontosabb rendszerváltozók értékének beállításával kezdődik. Ezt követően kerül sor a memória tesztelésére, illetve a BASIC munkaváltozók beállítására. Legvégül a gép bejelentkezik, kiírja a szabad memóriaterület nagyságát, és a 'READY.' üzenet kiírásával belép a képernyő szerkesztőbe. Ez az inicializálási eljárás állítja be a BASIC interpreter megszakítási rendszerét is.

A hardver megszakító rutin, amely másodpercenként mintegy ötvenszer hajtódik végre, a következő szolgáltatásokat végzi:

a/ a kurzor villogtatása;
b/ a stop billentyű lenyomásának ellenőrzése;
c/ a billentyűzet olvasása;
d/ a 0. lapon található óra aktualizálása;
e/ a megszakítások kezelése.

Ez e/ keretében kerül sor a MOVSPR és PLAY parancsok végrehajtására, illetve a COLLISION utasításnak megfelelő ütközés figyelésére. A rutinnak ez a része állítja elő a vágott képernyőt is. Természetesen a megszakító rutin ezen része csak az adott BASIC utasítások kiadása után és az utasítások teljes végrehajtásáig kerül sorra.

3.2 A BASIC programozási nyelv

3.2.1 A szintaxis

A BASIC nyelvet gyakran szokás 'angol' nyelvűnek is nevezni. Ez a kifejezés teljesen félrevezető, mert igaz ugyan, hogy a BASIC kulcsszavak angol szavak is egyben, de a programok 'nyelvtani szerkezetének' nem sok köze van az angol nyelvhez. A BASIC parancsok,sorok írását éppen úgy meg kell tanulnunk,mint bármely más program nyelvét,de még ebben az esetben is lehetnek értelmezési problémák. Helyesnek tekinthetünk-e egy 10 GOSUB 132 utasítást, ha a 132-es sor nem is létezik ?! Hogyan fogalmazzuk meg a DATA...RESTORE...READ utasítások egymáshoz való viszonyát? A problémát legegyszerűbben úgy hidalhatjuk át, ha az egyes BASIC utasításoknak külön-külön definiáljuk a szerkezetét. Ebben az esetben a parancs végrehajtása közben legalábbis ?SYNTAX ERROR hibajelzést nem kapunk. A C-128 BASIC sokban hasonlít a Commodore cég többi gépén futó BASIC-ekhez. Ezek valójában kompatibilisek, nemcsak a forrásnyelv, hanem a gépi megvalósitás szintjén is. Ha valaki elsajátítja a C-128 programozását, az azonnal képes a VC-20, a C-16, C-64, a C-128, a PET 600/700-as típusú gépek használatára.Mindegyik BASIC valójában egy egyszerűsített **Microsoft BASIC**, ahogy az a C-128-as mód logoján már meg is jelenik. Az egyszerűsítés elsősorban a perifériák kezelésére vonatkozik.

A C-128 BASIC a programsorok, parancsok szövegében levő szóközöket speciálisan kezeli. A szóközök száma a legtöbb esetben érdektelen. Akad azonban néhány kivétel. A file műveletek utasításaiban a kulcsszó és a # jel között **sohasem** lehet szóköz! A programsor elején levő és a sorszámot közvetlenül követő szóközök elvesznek.

Lehetőség van a BASIC kulcsszavak rövidítésére. Ez azt jelenti, hogy csak az alapszó első néhány karakterét gépeljük be, de ezek közül az **utolsót siftelve.** A 3.3 paragrafusban felsoroljuk, hogy melyik alapszót hogyan lehet rövidíteni. Néhány alapszót a C-64 gépen megszokottól eltérően kell rövidíteni.

3.2.2 Típusok

A Microsoft BASIC interpreterek három **változótípust** különböztetnek meg; ezek az **egész**, a **valós** és a **sztring** típusok. **A sztringeket karakterláncoknak vagy karakterfüzéreknek is szokás hívni.** A sztringek használata teszi lehetővé a szövegek kényelmes feldolgozását. A különböző típusokkal más és más műveleteket végezhetünk. A C-128 BASIC ehhez a három típushoz némileg hasonlóan kezeli a **függvénydefiníciók**at is.

3.2.3 Konstansok

A C-128 BASIC háromfajta konstans használatát engedi meg. Ezek azonosak a három típussal:

1/ egész;
2/ valós;
3/ sztring konstansok.

1/ Az **egész konstans**ok előjeles egész számok. A pozitív (+) előjel kiírása nem kötelező. A számnak a (-32768, 32767) zárt intervallumba kell esnie.

Példák:

123
-2567
9899

2/ A **valós konstans**ok a következő részekből állhatnak:

a/ a szám egész része
előjeles egész szám (I);
b/ tizedespont (.);
c/ a szám törtrésze
előjel nélküli egész szám (F);
d/ az exponens jele (E);
e/ az exponens(k)
előjeles egész szám.

A fentiek közül a tizedespont (.), vagy az exponens jelének (E) használata kötelező. Ha ezek hiányoznak, a gép a fenti konstanst egésznek tekinti. Az a/-e/ részekből álló valós konstans értéke, leszámítva a kerekítési hibákat a következő:

$$\left(I + \frac{F}{10^{n}} \right) * 10^{k},$$

ahol n az F törtrész jegyeinek száma. Az a/-c/ részek alkotják a **mantisszát**, a d/-e/ részek pedig az **exponenst**.

A legnagyobb, illetve legkisebb pozitív valós szám, amit az interpreter még tárolni tud:

MAX=1.70141183E+38
MIN=2.93873588E-39

Ha valamilyen számítás eredményeként MAX-nál nagyobb számot kapunk, a végrehajtás ?OVERFLOW ERROR hibaüzenettel megszakad. Ha a számítás eredménye MIN-nél kisebb pozitív szám, a végrehajtás folytatódik: az interpreter a 0.0 értékkel számol tovább.

A valós konstansok alakjának ismerete igen lényeges a nyomtatási kép tervezéséhez. A PRINT utasítás a valós típusú változók értékeit 9 jegyre kerekítve nyomtatja ki. Amennyiben a szám nem kisebb .01-nél és nem nagyobb 999999999-nél, a kiírás nem tartalmaz exponenst. Ha a fenti feltétel nem teljesül, a nyomtatási kép mantisszája mindig 1.-nél kisebb, de 0.1-nél nagyobb szám lesz, s a megfelelő exponens is kiíródik.

Példák:

```
123.456
2.43
-.99937
+3.1926
235.2E6
235E16
-.2E-13
.1E-2
.0123
```

3/ A **sztring** (vagy **szöveg**) **konstans**ok alfanumerikus és grafikus jelek sorozatát jelentik. A sztring konstansok hosszát csak az köti meg, hogy a képernyőről legfeljebb 160 karakter hosszú sort lehet bevinni. A sztring konstansokat idézőjelek (") közé kell írni. Igy az egyetlen alfanumerikus jel, ami nem szerepelhet sztring konstansban, az maga az idézőjel. (Sztring változó értékeként előálló sztringben már szerepelhet, a CHR$(34) felhasználásával.)

Példák:

```
"gyok="
"EREDMENYEK:"
"HOGY VAGY?"
"25.000.000$ ELEG?"
"  ?"
"" (üres sztring, nem egyenlő CHR$(0)-val!)
```

3.2.4 Változók

A C-128 BASIC interpreter a változók három típusát különbözteti meg, attól függően, hogy egész, valós vagy sztring mennyiségek tárolására szolgál. Ezen kívül megkülönböztet **egyszerű** vagy **indexes (tömb)** változókat. Az interpreter a változók típusát a nevüket követő **$**, illetve **%** jelből állapítja meg. Ezek jelentik a sztring illetve egész változókat. Ilyen jel hiányában az interpreter a változót valós típusúnak tekinti.

A valós és egész típusok közti konverziót a gép automatikusan végzi. Például az L%=L/256 értékadásban L/256-ot egészre kerekíti, ellenőrzi, hogy a [-32768, 32767] intervallumba esik-e, s ha igen, L%-hoz hozzárendeli ezt az értéket. Sztringek és számok közti konverzióra külön beépített függvényeket használhatunk: L$=STR$(L), L=VAL(L$), L%=VAL(L$). Két további konvertáló függvény a CHR$ és az ASC. (Lásd ezeket a 3.3 fejezetben!)

A változó nevek képzésének szabályai a következők:

(i) A változó nevének első karaktere csak betű lehet (A-Z).
(ii) A következő karakter tetszőleges alfanumerikus karakter (0-9,A-Z) lehet.
(iii) Ezt tetszőleges számú alfanumerikus karakter követheti, de ezek nem képezik a név részét. (**KA**TA és **KA**LAP tehát ugyanaz a név.)
(iv) Ezt esetleg egy $ vagy % követheti; jelezve, hogy a változó sztring, illetve egész típusú.
(v) A következő karakter egy (jel lehet, jelezve, hogy tömbváltozóról van szó. Ezt követik az indexek vesszővel elválasztva és a) jel, amelyek már nem részei a névnek.
(vi) A név nem tartalmazhat egyetlen BASIC alapszót sem (Például **VAL**LI, **WOR**D nem megengedett nevek). A fentartott változók TI, ST, TI$, DS, DS$, ER, EL, ERR$ lehetnek változó nevek részei, mert nem számítanak alapszónak (**FANTI** megengedett, és egyenlő FA-val).

A változók, akár parancs módban, akár program futása közben használjuk őket, a memóriában tárolt program után helyezkednek el a memóriában. Kivételt a sztringek képeznek. A fenti módon csak a nevük és egy mutató tárolódik. Ez utóbbi a sztring első karakterére mutat. Ezért a sztring-műveletek végrehajtásakor az interpreternek mindig külön meg kell győződnie arról, hogy van-e elég hely a memóriában. Ha nincs, a program futása ?OUT OF MEMORY ERROR hibaüzenettel félbeszakad.

3.2.5 A BASIC program és a változók tárolása

A BASIC program, illetve a változók a memória két különböző szeletében helyezkednek el. A BASIC program mindig a 0-ás szeleten, a rendszerváltozókat követően kezdődik. Ha grafikus képernyőt is használunk, akkor a BASIC program 10K-val feljebb kezdődik, s a 0-ás lap végéig tart. A változók az 1-es lapon, a közös memóriarész végétől ($0400) kezdődnek. Előbb az egyszerű, majd a tömbváltozók jönnek. A sztringeknek ezen a helyen csak a mutatóját tárolja az interpreter: a sztringek értékei az 1-es lap tetején találhatók. A memória felhasználását a 0. lapon levő mutatók írják le:

```
BASIC program kezdete    (45)($2D)
        Alapérték: $1C01 (grafika nélkül)
                   $4001 (grafikával)
BASIC terület vége       (55)($37)
        Alapérték: $FF00

1. szeleten
BASIC változók kezdete   (47)($2F)
        Alapérték: $0400
BASIC tömbök kezdete     (49)($31)
BASIC tömbök vége + 1    (51)($33)
BASIC sztringek kezdete  (53)($35)
```

A két külön szeleten levő BASIC munkaterületek elejét, a (45), (47) mutatókkal magunk is állíthatjuk. A többi mutató értéke a program szerkesztésétől, illetve futásától függően alakul.

Egyszerű változók tárolása

Tetszőleges, nem tömb változó 7 byte-nyi helyet foglal el a memóriában. Ezen túlmenően a sztring változók - a hosszuknak megfelelően - további byte-okat foglalnak el a BASIC munkaterület végén. Ezek esetében a sztring változó mutatója a program megfelelő helyére mutat. A tömbök az egyszerű változók után helyezkednek el a memóriában, így például az XT változó első használata azt eredményezi, hogy a tömböket 7 byte-tal feljebb tolja az interpreter, majd az XT változót a helyére rakja. A tárolás nem a legtömörebb, egész és sztring változók esetében 3, illetve 2 felesleges byte is van.

A sztringek tárolása eltér a C-64-en levőtől. Egy-egy sztringhez tartozó karaktersorozat végén egy 2-byte-os mutató található, ami visszamutat a sztring leírójára. Ez nagy mértékben meggyorsítja a szemétgyűjtési eljárást.

Tömbváltozók tárolása

A tömbváltozók tárolása a tömbre vonatkozó legfontosabb adatok tárolásával (név, dimenziószám, az egyes dimenziók nagysága) kezdődik, majd ezt követik az egyes tömbelemek 5, 3, illetve 2 byte-on, attól függően, hogy valós, sztring vagy egész típusú tömbről van-e szó. A tömbváltozó tartalmaz még egy relativ mutatót is, amelyik a következő tömbváltozó első byte-jára mutat. A tömbelemek oszlopfolytonosan helyezkednek el a memóriában, a DIM AL(1,2) deklaráció hatására például ilyen sorrendben:

AL(0,0),AL(1,0),AL(0,1),AL(1,1),AL(0,2),AL(1,2).

Ugyanezeket az elemeket mátrix alakban is elképzelhetjük:

AL(0,0)	AL(0,1)	AL(0,2)
AL(1,0)	AL(1,1)	AL(1,2)

3.2.6 Kifejezések

Az algebrában tanultakhoz hasonlóan konstansokból, változókból és egyilletve többváltozós műveletekből épithetünk fel **kifejezéseket.** A kifejezéseknek két fajtája van:

1/ **aritmetikai;**
2/ **sztring** kifejezések.

Sztring kifejezések

Először a sztring kifejezésekről szólunk, mert ez az egyszerűbb eset. Sztring kifejezések a következők lehetnek:

a/ sztring változók;
b/ két sztring kifejezés a + jellel összekapcsolva;
c/ minden olyan függvény, amelynek a nevében a $ jel szerepel. Ezek: **CHR$, ERR$, HEX$, LEFT$, MID$, RIGHT$, STR$.** (Természetesen a függvények argumentumainak megfelelő típusú kifejezéseknek kell lenniük.)
d/ a sztring kifejezések tetszés szerint zárójelezhetők.

Példák:

```
"AB CD EF"
G$+"N"
CHR$(N)
STR$(25)+RIGHT$(K$,3,6)
"NEW"+A$+B$.
```

A sztring függvények hatásáról a 3.3 paragrafusban részletesen lesz szó. A + sztring-művelet a sztringek **egymáshoz fűzését, konkatenációját** vagy **kompozícióját** jelenti. Például:

```
"KUTYA"+"FUL"+"E"="KUTYAFULE";
"ABRAKA"+DABRA"="ABRAKADABRA".
```

Az "" (üres) illetve CHR$(0) sztringek hatása a + művelet esetén nem ugyanaz:

```
A$+""=A$
A$+CHR$(0)<>A$
```

Megjegyezzük, hogy a DEF FN utasítás segítségével további sztring függvényeket **nem** definiálhatunk.

Aritmetikai kifejezések

Az aritmetikai kifejezések esete nem csak azért bonyolultabb, mert több művelet végezhető velük, hanem mert az aritmetikai kifejezések - mint speciális esetet - magukban foglalják a logikai kifejezéseket is. A logikai értékeket az interpreter 2-byte-os egész számként tárolja, s a megfelelő logikai műveletet ('és', 'nem', 'vagy' stb.) valamennyi biten egyszerre elvégzi. Ilyen módon logikai műveleteket számok közt is végezhetünk (például X% AND B% értelmes). Az interpreter a (2-byte-os egész számként tárolt) 0-t **hamis**nak, az ettől eltérő értéket **igaz**nak tekinti. **Az összehasonlítások eredményeként** azonban **mindig 0-t vagy -1-et kapunk.**

Relációs jelek

A relációs jelek (<,<=,>=,>,<>,=) segítségével két számot, vagy két sztringet hasonlíthatunk össze. Ha a számok típusa eltérő, az interpreter az egész számot lebegőpontosra alakítja át, és utána hajtja csak végre az összehasonlítást. Ha a két mennyiség eleget tesz a relációnak, eredményül -1-et, ha nem Ø-t kapunk. A relációs jelek értelme, amikor számokat hasonlítunk össze, a szokásos:

=	egyenlő
<>	nem egyenlő
>	nagyobb
<	kisebb
>=	nagyobb vagy egyenlő
<=	kisebb vagy egyenlő

Sztringek esetén = és <> jelentése értelemszerű. Az **A$<B$** jelentése a következő. Sorba vesszük az A$, B$ sztringekben szereplő jeleket, és megnézzük, melyik az a hely, ahol először eltérnek. Ha ilyen nincs, akkor A$ < B$ pontosan akkor teljesül, ha A$ rövidebb, mint B$. Ha ilyen hely van, akkor A$<B$ azt jelenti, hogy az első eltérő helyen szereplő jel ASCII kódja az A$ sztringben kisebb, mint B$-ban. Elképzelhető, hogy az A$ sztring a B$ sztring kezdőszelete (pl. A$="hazmester", B$="haz"). Ebben az esetben A$<B$ mindig teljesül.

Példák:

1=4	hamis (Ø)
14>=62	hamis (Ø)
14<62	igaz (-1)
5*5 <> 16	igaz (-1)
"ABC"<"AX"	igaz (-1)
"A"<"D"	igaz (-1)
"A"<"9"	hamis (Ø)
"XYZ">"XY"	igaz (-1)
"XYZU"<="XY9"	hamis (Ø)
""<=CHR$(Ø)	igaz (-1)
CHR$(Ø)<="@"	igaz (-1)

Az eredményről magunk is meggyőződhetünk. Adjuk ki például a PRINT 1=4 parancsot! Válaszul az interpreter az 1=4 állítás logikai értékét írja ki, ami hamis. Az eredmény tehát Ø lesz.

Logikai műveletek

A BASIC interpreter összesen három logikai műveletet ismer, egy egyváltozós (**NOT**) és két kétváltozós (**AND, OR**) műveletet. Egész számokon egy további logikai művelet is végezhető: a **kizáró vagy:** XOR(A%,B%). Szemben azonban az AND és az OR használatával a XOR paramétereit a XOR után zárójelbe írva és vesszővel elválasztva kell megadni. A XOR(A%,B%) végrehajtásakor az interpreter az A% és a B% egész szám 16 bites reprezentációján bitenként hajtja végre a kizáró vagy-ot. A XOR műveletáblája a következő:

XOR	igaz	hamis
igaz	hamis	igaz
hamis	igaz	hamis

Aritmetikai műveletek

A BASIC interpreter a négy alapműveletet, valamint a hatványozás műveletét ismeri. Ezek jele a szokásos:

+ összeadás
- kivonás, ellentett képzés
* szorzás
/ osztás
↑ hatványozás

A gyökvonás az X↑(1/Y) kifejezés segítségével végezhető el. Az interpreter az algebrából ismert műveleti hierarchia szerint dolgozik. **Egyenrangú műveletek esetén szigorúan balról jobbra haladva végzi el a műveleteket.** Például A/B/C algebrai alakja:

$$\frac{a}{b*c}$$

A kerekítési hibák miatt az algebrából ismert azonosságok nem mindig teljesülnek!

Aritmetikai függvények

A BASIC interpreter a legfontosabb matematikai függvényeket beépített rutinként ki tudja számítani. Ezek: **ABS, ASC, ATN, BUMP, COS, DEC, EXP, FRE, INSTR, INT, JOY, LEN, LOG, PEEK, PEN, POINTER, POS, POT, RCLR, RDOT, RGR, RND, RSPCOLOR, RSPRITE, RSPPOS, RWINDOW, SGN, SIN, SQR, TAN, VAL, XOR.** (A felsorolt 32 függvény közt néhány egészen speciális függvény is van. Ezek hatásáról a 3.3 paragrafusban szólunk részletesen.)

Megjegyezzük, hogy a DEF FN utasítás segítségével további - egyváltozós - aritmetikai függvényeket tudunk definiálni.
Ennyi előkészités után definiáljuk, hogyan is épülnek fel az aritmetikai kifejezések:

a/ tetszőleges egész vagy valós változó aritmetikai kifejezés;
b/ ha aritmetikai kifejezéseket műveleti jelekkel összekapcsolunk, akkor újból aritmetikai kifejezést kapunk;
c/ egy aritmetikai függvény, utána zárójelben a megfelelő típusú argumentum ugyancsak aritmetikai kifejezés;
d/ két aritmetikai vagy sztring kifejezés összehasonlítása aritmetikai kifejezés;
e/ egy vagy két aritmetikai kifejezés logikai műveletekkel összekötve újból aritmetikai kifejezés;
f/ aritmetikai kifejezések tetszés szerint zárójelezhetők;
g/ a logikai és aritmetikai kifejezések azonosak.

Példák:

```
A+B+.23
C↑(D+E)/2
((X-C)↑(D-E)/2)*10)+1
K%=1 AND M<>X
NOT (D=E)
K%=2 OR ( A=B AND M<X )
FN DEEK(X)+FN DEEK(X+2)=0
```

Az előbbi definíció egy igen érdekes sajátosságára hívjuk fel a figyelmet. Aritmetikai kifejezések szerepeltetése a logikai kifejezések közt megszokott. A C-128 azonban azonosítja a logikai és aritmetikai kifejezéseket. Tekintsük például a következő értékadást:

```
L%=ASC(L$)-48+(ASC(L$)>64)*7
```

L$ egy hexadecimális számot tartalmaz karakteres alakban. A fenti értékadás L%-hoz a karakternek megfelelő számértéket rendeli.

Hasonlóan az IF X THEN GOTO... utasításban a kifejezés akkor lesz igaz, ha X értéke Ø-tól különbözik.

Műveletek sorrendje

Mindig a magasabb rangú műveletek hajtódnak először végre. Egyenrangú műveletek esetén a műveletek végzése balról jobbra halad. A műveletek a következő sorrendben kerülnek végrehajtásra:

1/ ↑ hatványozás
2/ - ellentett képzés
3/ * / szorzás, osztás
4/ + - összeadás, kivonás
5/ >=< összehasonlítás
6/ NOT logikai nem
7/ AND logikai és
8/ OR logikai vagy

Példák (összefoglaló):

	Aritmetikai	Sztring
Konstans	2.3	"ABCD"
	-5	"EREDMENY"
	2E-1	"DRAGA"
	3.1415	
Változó	XY	AC$
	A(23)	TI$
	X(5,I+7)	UZEN$(J,8)
Kifejezés	A+B↑2	" "+G$
	2*X*X-3*Y	LEFT$(" "+C$,4)
	COS(X)-TAN(4)	MID$(A$,2,3)+C$
	FN LG(X)+ D(I)	

3.3 BASIC alapszavak ABC sorrendben

Ez a fejezet tartalmazza a C-128 BASIC interpreter utasításait és függvényeit, azok leírását, működésük részletes ismertetését. A Commodore 64 BASIC V2.0 utasításait csak felsoroljuk. Kivételt képeznek azok az alapszavak, amelyek használata valami miatt eltér a C-64-es módban való használattól. A C-64-es hivatkozások a könyv negyedik kiadására (a szövegszerkesztővel készített változatra) vonatkoznak!

Az egyes utasításokról szóló részek a következőképpen épülnek fel.

Rövidítés: A legtöbb BASIC alapszót nem kell teljes egészében beírnunk, elég az első néhány karakter beírása, amelyek közül az utolsót siftelve (emelve) kell beírnunk. A fenti címszó alatt a rövidítés 'kisbetűk/nagybetűk' karakterkészlet használata esetén látható alakját adjuk meg. A 'nagybetűk/grafikus jelek' karakterkészlet esetében a rövidítés utolsó karaktere egy grafikus jel lesz.

Token: A BASIC interpreter az alapszavakat egyetlen byte-on tárolja. Ez a byte a szóban forgó **alapszó tokenje.** Az érvényes tokenek 128 és 255 közé esnek. Vannak olyan alapszavak, amelyek tokenje két byte-ból áll. A fenntartott változóknak nincs külön tokenje. (Ezért lehetnek változónevek részei.)

Funkció: Ebben a részben röviden összefoglaljuk az utasítás hatását. A BASIC-et ismerők számára ez már általában elegendő az utasítás használatához. Kezdők feltétlenül tanulmányozzák át a példákat, illetve az azt követő magyarázatokat.

Szintaxis: Az egyes **szintaktikus egység**ek megnevezéseit **csúcsos zárójel**ek közé tesszük, például <aritmetikai kifejezés>. A **kapcsos zárójel**ek közti részek **nem kötelező**ek, opcionálisak. Ha viszont szerepelnek, akkor csak a kapcsos zárójelben megadott formában használhatjuk őket. Például LIST{-<sorszám>}.

A **kapcsos zárójelek közt, egymás alatti sorokban** szereplő részek **kötelező választás**i lehetőségeket sorolnak fel. (Lásd például az IF utasításnál.)

A szintaxis leírásánál szereplő **többi jel** (",: stb.), az **utasítás része.** Ezt a tényt a szóbanforgó részek **vastagított** szedésével is jelezzük. A szintaxis leírásánál használt szögletes és kapcsos zárójelek természetesen **nem** részei az utasításoknak!

Példák: A példákat igyekeztünk úgy összeválogatni, hogy az utasítás legtipikusabb alkalmazási lehetőségeit bemutassák. Itt emeljük ki azokat a hatásokat is, amelyek nem szokásosak, **eltérnek** a legtöbb BASIC megvalósítástól. A példákat a végrehajtásra utaló megjegyzésekkel zárjuk.

Hibalehetőségek: Itt soroljuk fel a leggyakoribb hibákat.

ABS

c-64-es mód

Rövidítés: aB Token:$B6(182)

Mód: mind parancs, mind program módban használható

Függvény: az argumentum abszolút értékével tér vissza. (Lásd C-64 49. oldal!)

Szintaxis: **ABS**(<arit.kif.>)

AND

c-64-es mód

Rövidítés: aN Token: $AF (175)

Mód: mind parancs, mind program módban használható

Két aritmetikai kifejezés **logikai és**-ét adja meg. (Lásd C-64 50. oldal!)

Szintaxis: <arit.kif.1> **AND** <arit.kif.2>

APPEND

Rövidítés: aP Token:$FE $0E (254,14)

Mód: mind parancs, mind program módban használható

Egy már létező SEQ vagy USR típusú file-t nyit meg továbbírásra

Szintaxis: **APPEND#**<lfszám>,<filenév> {,**D**<meghajtó>} {,**U**<egység>}

<lfszám> az a logikai fileszám, ami alatt a BASIC nyilvántartásba veszi a file-t. Később a PRINT#, GET#, INPUT# utasításokban erre lehet hivatkozni. értékének az 1-255 intervallumba kell esnie. <filenév> a továbbírásra megnyitandó file neve. Ha nem sztring konstans, hanem sztring kifejezés, akkor kerek zárójelek közé kell tenni. <meghajtó> a meghajtó száma: 0 vagy 1. Az 1 érték csak duál lemezegység esetén használható. <egység> a használt lemezegység hardver száma, ez általában 8 vagy 9.

Példák:

(i)
```
APPEND#2,"PELDA"
X$="PELDA": APPEND#2,(X$)
APPEND#2,"ADATOK",D1,U9
```

(ii)
```
10 DOPEN#1,"FILE",W
20 PRINT#1,"elso adat"
30 DCLOSE#1
40 :
```

```
50 APPEND#2,"FILE"
60 PRINT#2,"masodik adat"
70 DCLOSE#2
80 :
90 DOPEN#3,"FILE"
100 INPUT#3,X1$,X2$
110 DCLOSE#3
120 ?X1$,X2$
```

Az (i) alatti példák az APPEND egyszerű használatát mutatják be. A harmadik sorban a 9-es lemezegység 1-es meghajtójában kell lennie a lemeznek, amin az ADATOK nevű file van.

(ii) alatt egy rövid BASIC programot írtunk, ami az APPEND hatását mutatja. A 10-30 sorokban létrehoztunk egy "FILE" nevű file-t, amiben egyetlen sztringet írtunk ki. Az 50-70 sorokban ehhez hozzáírunk még egy sztringet, majd a 90-120 sorokban a két sztringet az X1$ és X2$ változókba visszaolvassuk és kiírjuk a képernyőre.

Hibalehetőségek: Leggyakoribb hiba, hogy egy nem létező file-t akarunk megnyitni. Ez nem csak úgy fordulhat elő, hogy a lemezen nincs a file, hanem nem megfelelő lemezegységen van. Az egyes kiértékelések kiértékelése közben is kaphatunk ?SYNTAX ERROR hibaüzenetet.

ASC

C-64-es mód

Rövidítés: aS Token: $C6 (198)

Mód: mind parancs, mind program módban használható

Sztring függvény: az argumentum sztring első karakterének ASCII kódját adja. (Lásd C-64 51. oldal!)

Szintaxis: **ASC**(<sztring kif.>)

ATN

C-64-es mód

Rövidítés: aT Token: $C1 (193)

Mód: mind parancs, mind program módban használható

Függvény: az argumentum arkusz tangensének főértékét adja. (Lásd C-64 52. oldal!)

Szintaxis: **ATN**(<arit.kif.>)

AUTO

Rövidítés: aU Token: $DC(220)

Mód: csak parancs módban használható.

Automatikus sorszámozást biztosít programíráshoz.

Szintaxis: **AUTO** {<növekmény>}

<növekmény> aritmetikai kifejezés, a sorszámozás növelését adja meg. Az automatikus sorszámozás megszüntetését az AUTO parancs paraméter nélküli kiadásával érhetjük el. <növekmény> értéke nem kell, hogy egész legyen, ilyenkor a kifejezés egész értéke lesz a sorszámnövekmény.

Példák:

```
AUTO 5
10 INPUT A,B
15 C=SQR(A*A+B*B)
20 PRINT "C=";C
25<RETURN>
AUTO
```

A példában egy három sorból álló egyszerű programot írtunk meg. Az AUTO 5 parancs kiadásával beállítottuk az automatikus sorszámozást. Ez az első programsor (a 10-es sorszámu) begépelése után azt eredményezi, hogy a következő sor elején a 15-ös sorszám automatikusan megjelenik. A második programsor beírása után a következő sor elejére kiíródik a 20-as sorszám. (Ezeket vastagon szedtük!) Ha már nem akarunk több programsort írni, akkor a <RETURN> megnyomásával kilépünk a programírásból, s egy paraméter nélküli AUTO paranccsal kikapcsoljuk az automatikus sorszámozást. Ha ezután írunk be egy programsort, akkor az új sor elejére már nem íródik ki a következő sorszám, s a kurzor a sor elejére kerül.

Az automatikus sorszámozást a program első változatának a beírásánál, vagy nagyobb programrészek hozzáírásánál használjuk. Programok javításakor általában nem célszerű az automatikus sorszámozás használata.

Hibalehetőségek: Az aritmetikai kifejezés kiértékelése közben számos hibalehetőség adódhat. Ha a kifejezés értéke nem pozitív, vagy túl nagy, akkor ?ILLEGAL QUANTITY ERROR hibajelzést kapunk.

BACKUP

Rövidítés: bA Token: $F6(246)

Mód: mind parancs, mind program módban használható.

Duál lemezegység (CBM4040 vagy illesztett 8250-es egység) használata esetén az egyik meghajtóban levő lemez tartalmát teljes egészében átmásolja a másik meghajtóban levő lemezre. Ily módon adattartalmát tekintve két **teljesen azonos** lemez jön létre. Egymeghajtós lemezegység esetén nem használható!

Szintaxis: **BACKUP D**<meghajtó1> **TO D**<meghajtó2> {U<egység>}

A <meghajtó1> a **másolandó** lemezt tartalmazó meghajtó száma, míg a <meghajtó2> a létrehozandó új lemezt tartalmazó meghajtó száma. Mindkettő értéke vagy 0, vagy 1 lehet. Az <egység> értéke a használandó lemezegység hardver egységszáma. Ez 8,9,10 vagy 11 lehet. Ha nem adjuk meg, az interpreter 8-nak veszi. AZ új lemeznek nem kell formázottnak lennie, a BACKUP újraformázza a lemezt.

Példák

```
(i)     BACKUP D0 TO D1: PRINT DS$
(ii)    BACKUP D1 TO D0 ON U9
        BACKUP D1 TO D0, U9
```

Az első példa a BACKUP leggyakoribb használatát mutatja: parancs módban a lemezről biztonsági másolatot készítünk, majd lekérdezzük a lemezegység hibacsatornáját. Ugyanezt gyakran szokás programból is használni a következő programrész segitségével:

```
1000 PRINT "KEREM A BACKUP LEMEZT AZ 1-ES MEGHAJTOBA!!!"
1005 GETKEY A$
1010 BACKUP D0 TO D1
1015 PRINT DS$: END
```

A (ii) alatti két parancs az U paraméter két lehetséges írásmódját mutatja.

Hazánkban jelenleg nem forgalmaznak a C-128-hoz (vagy a C-16-hoz, illetve a C-64-hoz) közvetlenül kapcsolható duál meghajtót. A legelterjedtebb lemezegység, a VC 1541-es, egymeghajtós. Ezért teljes lemezek másolásához speciális programokat kell használni. Részletesen erről az 5. fejezetben szólunk.

Hibalehetőségek: Egymeghajtós lemezegységek esetén az egység természetesen nem kezdi el a másolást. Ha a másolás közben olvasási vagy íráshiba történik, akkor a másolás félbeszakad. A hibacsatorna olvasásával megtudhatjuk, hogy melyik lemez melyik blokkjával milyen hiba történt. A BACKUP D0 TO D0 parancs a lemezt önmagára másolja vissza. Ez a mágnesezés felfrissítésére jó, de ilyenkor is célszerűbb új lemezre másolni.

BANK

Rövidítés: bA Token:$FE $Ø2 (254,2)

Mód: mind parancs, mind program módban használható.

Kijelöli a PEEK, POKE, SYS és WAIT utasítások által használt 64K-s szeletet.

Szintaxis: **BANK** <szelet>

<szelet> tetszőleges aritmetika kifejezés, értéke Ø-15 lehet. (A memória használatáról bővebben lásd a 7.3 paragrafust!)

Példák:
```
  (i)     BANK Ø: SYS 1255
 (ii)     BANK 15: SYS DEC("FFE2"),A,X,Y
```

Az (i) példában a Ø. lapon levő gépi kódú rutint indítjuk el. (ii) esetében a ROM-ban levő rutint hívjuk meg. A SYS-ben még a regiszterek értékét is beállítjuk.

Hibalehetőségek: Nincs.

BEGIN...BEND

Rövidítés: bE Token: $FE $18 (254,28)
beN $FE $19 (254,29)

Kezdő és záró utasításszójel: az IF...THEN...ELSE konstrukcióban használható. Nagy mértékben megkönnyítik strukturált programok írását.

Szintaxis: **BEGIN**
BEND

Az utasítás az alábbi programkörnyezetben használható:

```
IF...THEN BEGIN
...
<utasítások>
...
BEND :ELSE BEGIN
...
<utasítások>
...
BEND
```

A BASIC V2.Ø és V3.5 változatokban csak olyan IF...THEN utasítást írhattunk, amelyik egyetlen sorban fért csak el. Ez feltételes elágazások kezelését rettentő bonyolulttá tette.

Példák:

(i)
```
10 INPUT X
20 IF X<0 THEN BEGIN
30 : ?"EZ A SZAM NEGATIV!"
40 : ?"X=";X
50 BEND: ELSE BEGIN
60 : ?"EZ A SZAM POZITIV VAGY 0!"
70 : ?"X=";X;"GYOK X=";SQRT(X)
80 BEND
90 GOTO 10
```

(ii)
```
10 INPUT "A,B=";A,B
20 A=ABS(A): B=ABS(B)
30 :
40 IF A*B=0 THEN BEGIN
50 : ?"VALAMELYIK 0!"
60 : END
70 BEND
80 :
90 DO UNTIL A=B
100 : IF A<B THEN B=B-A: ELSE A=A-B
110 LOOP
120 ?"LNKO=";A
```

Két rövid programot mutatunk be a BEGIN használatára. Valójában mindkét esetben elkerülhető a BEGIN...BEND pár használata, de az utasítászárójelek éppen a programozó kényelmét szolgálják!.

Az (i) példa egy végtelenített program, amelyik inputként egy számot vár, hogy abból négyzetgyököt vonjon. Ha a szám negatív, ez lehetetlen, s ezért kiírja, hogy a szám negatív, és új adatot kér. Ha a szám nem negatív, kiírja ezt a tényt is, és kiírja a szám négyzetgyökét.

A (ii) példa két egész szám legnagyobb közös osztóját számítja ki, egy olyan módszerrel, amelyik csak a kivonás műveletét használja. (Egyesek rémálmából bizonyára ismerős...). A program elején megkérdezzük, van-e a számok közt 0, ugyanis, akkor az eljárás nem működik.

Hibalehetőségek: A BEGIN...BEND használata elsősorban szemantikai hibákat okoz: a program nem azt csinálja, amit kellene, pedig csinál valamit! Ha az IF...THEN...ELSE végrehajtása közben nem találja az utasítászárójel végét, akkor ?BEND NOT FOUND hibaüzenetet kapunk.

BLOAD

Rövidítés: bL Token: $FE $11 (254,17)

Mód: mind parancs, mind program módban használható.

PRG típusú file-t tölt be a memóriába az általunk megadott helyre.

Szintaxis:
BLOAD <filenév> {,**D**<meghajtó>}{,**U**<egység>}{,**ON B**<szelet>} {,**P**<cím>}

A programot a <filenév> nevű fileból tölti be az utasítás a memóriába. <szelet> annak a memóriaszeletnek a száma, ahová be akarjuk tölteni a programot, <cím> a töltés kezdőcíme. Ha elmaradnak ezek a paraméterek a programot a 0-ás szeletre tölti az utasítás. A kezdőcímet a file első két byte-ja adja. <meghajtó> a meghajtó száma: 0 vagy 1. Az 1 érték csak duál lemezegység esetén használható. <egység> a használt lemezegység hardver száma, ez általában 8 vagy 9.

Példák:

```
  (i)     BLOAD "RUTINOK",B1,P1024
 (ii)     1500 X$="RUTINOK"
          1510 SZ=1
          1520 CI=1024
          1530 BLOAD (X$), B(SZ), P(CI)
```

Az első példa a "RUTINOK" nevű programot tölti be az első lapra, közvetlenül a közös rész után (<cím>=1024). A (ii) ugyanezt a feladatot végzi el, de mutatja, hogy abban az esetben, ha a paraméterek nem konstansok, kerek zárójelek közé kell tenni őket.

Hibalehetőségek: A kerek zárójelek hiánya gyakran okoz ?SYNTAX ERROR hibaüzenetet. Ha a file nincs a lemezen, akkor ?FILE NOT FOUND hibajelzést kapunk. Ha rossz helyre töltünk a memóriába, akkor programjaink tönkremehetnek.

BOOT

Rövidítés: bO Token: $FE $1B (254,27)

Mód: mind parancs, mind program módban használhatjuk.

Gépi kódú programok (automatikus) betöltésére és indítására szolgál.

Szintaxis **BOOT** { <filenév> {,**D**<meghajtó>}{,**U**<egység>} }

Ha a <filenév> szerepel, akkor a specifikált meghajtóban levő lemezről betölti a <filenév> nevű file-t, és a belépési pontjáról elindítja. <meghajtó> a meghajtó száma: 0 vagy 1. Az 1 érték csak duál lemezegység esetén használható. <egység> a használt lemezegység hardver száma, ez általában 8 vagy 9.

Ha paraméter nélkül használjuk a BOOT utasítást, akkor a 8-as egység 0-ás meghajtójában levő lemez 1. sávjának 0. szektorában keresi a betöltéshez szükséges információt. Lásd az 5.4 paragrafust erről!

Példák:

```
          BOOT "SEGITSEG"
          BOOT "SEGITSEG",U9
```

Hibalehetőségek: Ha a program nincs a lemezen ?FILE NOT FOUND hibaüzenetet kapunk. Ha a paraméter nélküli BOOT utasítást használjuk, az interpreter először ellenőrzi, hogy az 1. sáv 0. szektorának első három byte-ja CBM. Ha nem, az utasítás végrehajtása abbamarad.

BOX

Rövidítés: nincs Token: $E1(225)

Mód: mind parancs, mind program módban használható.

A 40 oszlopos, nagyfelbontású képernyőre egy téglalapot rajzol. Az utasítás paraméterei a téglalap nagyságát, dőlésszögét, színét határozzák meg. Megadhatjuk továbbá, hogy a téglalap belsejét is kiszínezze-e az interpreter.

Szintaxis:

BOX {<színtípus>},<x1>,<y1>{,{<x2>,<y2>}}{,{<szög>}}{,{<töltés>}}

A <színtípus> a lehetséges négy színtípus (0-3) egyike lehet. Ha elmarad, az 1-nek, az írás színének felel meg. <x1>,<y1> a téglalap bal felső, míg <x2>,<y2> a téglalap jobb alsó csúcsának a koordinátái. <szög> a téglalap oldalainak a vízszintessel bezárt szögét adja meg radiánokban. A <töltés> értéke 0 vagy 1 lehet. Ha 0, akkor az utasítás csak a téglalap határát rajzolja meg. Ha 1, akkor a téglalap belsejét is kiszínezi a kiválasztott <színtípus>-nak megfelelő színnel.

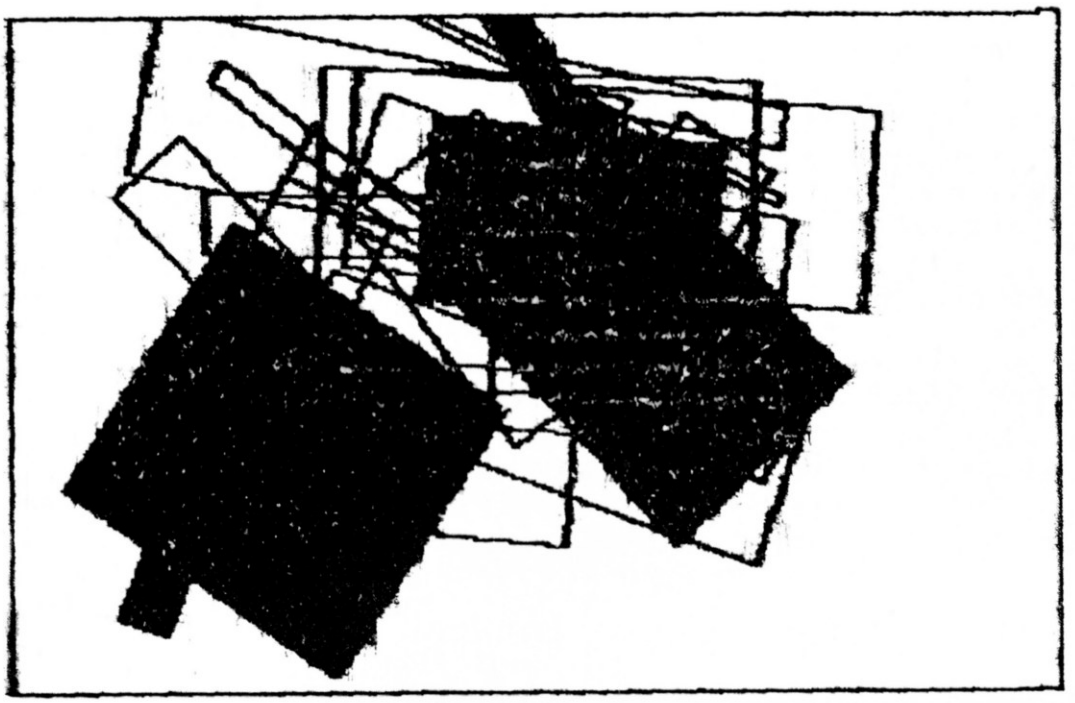
Példák:

```
  (i)     GRAPHIC 1,1: BOX 1,10,10,309,189,0,0
          GRAPHIC 1,1: BOX ,10,10,309,189
          GRAPHIC 1,1: LOCATE 309,189: BOX ,10,10
 (ii)     GRAPHIC 1,1
          BOX 1,10,10,80,80
          BOX 0,10,10,80,80
(iii)     10 GRAPHIC1,1
          20 FOR I=1 TO 30
          30 A1=RND(0)*160:B1=RND(0)*100
          40 A2=RND(0)*160:B2=RND(0)*100
          50 IF RND(0)>.8 THEN X=1: ELSE X=0
          60 BOX 1,A1+30,B1+10,A1+A2+30,B1+B2+10,60*RND(0),X
          70 NEXT I
```

Az első példa mindhárom sora ugyanazt végzi el: a képernyőre rajzol egy téglalapot. Az első sorban valamennyi paramétert szerepeltetjük. Az utasítás második formájában három paraméter is hiányzik. A harmadik esetben a jobb alsó csúcspont koordinátái is hiányoznak. Ebben az esetben az interpreter a grafikus kurzor pillanatnyi helyzetét használja. Ennek megfelelően a LOCATE utasítás segítségével a grafikus kurzort a leendő téglalap jobb alsó csúcsára állítjuk.

A (ii) alatti példa a színtípus használatát mutatja be. Az első BOX utasítás rajzol egy téglalapot a képernyőre. Ezután ugyanazt a téglalapot rajzoljuk meg, de most a háttér színével. Ez az előző téglalap **törlését** eredményezi.

A (iii) alatti program véletlenszerűen kiválasztott téglalapokat rajzol a képernyőre.



Hibalehetőségek: Az aritmetikai kifejezések kiértékelése közben számos hiba történhet. A téglalap nem kell, hogy teljes egészében a képernyőre essen, ebben az esetben az interpreter a téglalap képernyőre eső darabját rajzolja csak meg. A használt színtípusnak összhangban kell lennie a kiválasztott grafikus móddal. Ha nem, akkor ?ILLEGAL QUANTITY ERROR hibajelzést kapunk. A paraméterek elhagyása esetén a megfelelő számú vesszőt ki kell tenni, különben a paraméterek összekeverednek. Ha nem adtuk még ki a megfelelő GRAPHIC utasítást, akkor ?NO GRAPHIC AREA hibaüzenetet kapunk.

BSAVE

Rövidítés: bS Token: $FE $10 (254,16)

Mód: mind parancs, mind program módban használható.

Szintaxis:
BSAVE <filenév> {,**D**<meghajtó>}{,**U**<egység>}
{,**ON B**<szelet>}{,**P**<cím1> **TO P**<cím2>}

A memória megadott részét az utasításban specifikált meghajtóban levő lemezre <filenév> alatt elmenti (kiírja). A lemezen a <filenév> nevű file nem létezhet. Ha igen a név elé egy @ karaktert is be kell írnunk. <szelet> a memória 64K-s szeletét jelöli ki, ha elmarad a rendszer 0-nak értelmezi. A <cím1> a kezdőcím, <cím2> a végcím. A végcímnek megfelelő memóriahely még kiíródik a lemezre. A <meghajtó> értéke 0 vagy 1 lehet. Az 1 érték csak duál lemezegység esetén használható. <egység> a használt lemezegység hardver száma, ez általában 8 vagy 9. Ha valamelyik paraméter - a <filenév> kivételével - kifejezés, akkor kerek zárójelek közé kell tenni.

Példák
(i)
```
BSAVE "KEP", ON B0, P1024 TO P2047
?"<CLR>"
BLOAD "KEP", ON B0
```

A példában a 40 oszlopos monitor képernyő memóriájának tartalmát kiirtuk a lemezre a "KEP" nevű PRG típusú file-ba. Ezt úgy ellenőriztük, hogy a képernyőt töröltük, s a file tartalmát a BLOAD utasítás segítségével visszatöltöttük.

Hibalehetőségek: Ha valamelyik paraméter nem konstans, akkor kerek zárójelek közé kell tenni. Ellenkező esetben ?SYNTAX ERROR hibajelzést kapunk. Ha a file már létezik, és nem használtuk a @ karaktert a név elején, akkor ?FILE EXISTS hibaüzenetet kapunk.

BUMP

Rövidítés: bU Token: $CE $03 (206,03)

Mód: mind parancs, mind program módban használható.

Függvény: a sprite-ok ütközését tároló regiszterek értékét adja vissza.

Szintaxis: **BUMP**(<aritmetikai kifejezés>)

Az <aritmetikai kifejezés> értéke 1 vagy 2 lehet. Ha 1, akkor a sprite-sprite ütközés, ha 2, akkor a sprite-háttér ütközés regiszterének értékét kapjuk meg. BUMP értéke a 0-255 intervallumba esik. Ha BUMP=0, akkor nem történt ütközés. Ha valamelyik bit magas, akkor a neki megfelelő sprite ütközött.

Példák:
```
IF BUMP(1) AND 2↑5 THEN...
  IF BUMP(2)<>0 THEN...
```

Az első esetben a vezérlésátadás akkor történik meg, ha az 5-ös sorszámú sprite ütközött valamelyik másik sprite-tal. (Vigyázat: erre a sprite-ra a sprite-okat mozgató utasításokban 6.-ként kell hivatkozni!). A második esetben a vezérlésátadás akkor következik be, ha akármelyik sprite ütközik a háttérrel.

Hibalehetőségek: Ha a függvény argumentumának értéke nem 1 vagy 2, akkor ?ILLEGAL QUANTITY ERROR hibajelzést kapunk.

CATALOG

Rövidítés: cA Token: $FE $0C (254,12)

Mód: mind parancs, mind program módban használható.

A képernyőre listázza a katalógust. Hatása azonos a DIRECTORY utasítással. Lásd ott!

CHAR

Rövidítés: chA Token: $E0 (224)

Mód: mind parancs, mind program módban használható.

Segítségével mind a karakteres, mind a 40 oszlopos nagyfelbontású képernyő tetszőleges helyére írhatunk karaktereket.

Szintaxis:
CHAR {<színtípus>},<x>,<y> {,<sztring kif.>} {,<inverz-jel>}}

A <színtípus> a kiírandó karakterek színét adja meg. Ha elhagyjuk, az az 1-es színtípusnak - az írás színe - felel meg. Az <x>,<y> aritmetikai kifejezések a kiírás kezdőpozícióját határozzák meg, amit mind karakteres, mind nagyfelbontású képernyő esetén karakterhelyekben kell megadni. Az <inverz-jel> aritmetikai kifejezés értéke 0, vagy 1 lehet. Ha 1, a sztring kifejezés karakterei inverz alakban kerülnek a képernyőre. Nagyfelbontású képernyő használata esetén a vezérlő karakterek helyett is grafikus jelek íródnak ki.

Példák:

```
  (i)     GRAPHIC 0,1: CHAR 1,10,10,"ABRAKADABRA"
          GRAPHIC 1,1: CHAR 1,10,10,"ABRAKADABRA"
 (ii)     GRAPHIC 1,1: CHAR 1,10,10,"KUTYAFULE",1
          GRAPHIC 1,1: CHAR 1,10,26,"KUTYAFULE"
(iii)     10 GRAPHIC 1,1
          15 CHAR 1,13,12,"I "+CHR$(211)+" COMMODORE-128"
          20 BOX ,100,92,235,106
          25 GETKEY A$: GRAPHIC 0
```

Az (i) alatti két parancs a karakteres, illetve a nagyfelbontású képernyő használatát mutatja be. Jól látható, hogy a nagyfelbontású képernyő használata esetén is a 10 karaktersort, nem pedig rasztersort jelent.

A (ii) alatti példa a nagyfelbontású képernyőre írja a "KUTYAFULE" feliratot először inverz, azután normál alakban. A (iii) példa mutatja, hogy sztringkifejezések is használhatók az utasításban.

Hibalehetőségek A színtípus értéke (0-3) csak a kiválasztott grafikus módnak megfelelő lehet. Ha ettől eltér, akkor ?ILLEGAL QUANTITY ERROR hibajelzést kapunk. Ugyanez vonatkozik az <inverz-jel>-re is. (értéke 0, vagy 1 lehet.)

CHR$ C-64-es mód

Rövidítés: cH Token: $C7 (199)

Mód: mind parancs, mind program módban használható

Sztring függvény: adott ASCII kódból egy egy karakter hosszú sztringet állít elő. ASC inverze. (Lásd C-64 53. oldal!)

Szintaxis: **CHR$**(<arit.kif.>)

CIRCLE

Rövidítés: cI Token: $E2(226)

Mód: mind parancs, mind program módban használható.

Az utasítás segítségével a 40 oszlopos nagyfelbontásu képernyőre ellipszist, illetve egy ellipszis meghatározott darabját rajzolhatjuk.

Szintaxis
CIRCLE {<színtípus>},{<x>,<y>},<xr>
{,{<yr>}{,{<ks>}{,{<vs>}{,{<szög>}{,{<nv>}}}}}}

A <színtípus> a megrajzolandó ellipszis pontjainak a színét határozza meg. Ha elhagyjuk, akkor az interpreter az 1-es színtípust használja. Az <x>,<y> koordinátapár az ellipszis középpontját, <xr>,<yr> a két tengelyét, <sz> pedig az ellipszis nagytengelyének a vízszintessel bezárt szögét határozza meg. Ha az <x>,<y> értékeket elhagyjuk, az interpreter a grafikus kurzor aktuális értékét tekinti az ellipszis középpontjának. Ha <yr> hiányzik, akkor az interpreter az <yr>=<xr> értékkel dolgozik, vagyis kört rajzol. Ha <szög> hiányzik, akkor az interpreter a 0 értékkel dolgozik. Az ellipszis egyik nagytengelye tehát vízszintes, a másik függőleges lesz.

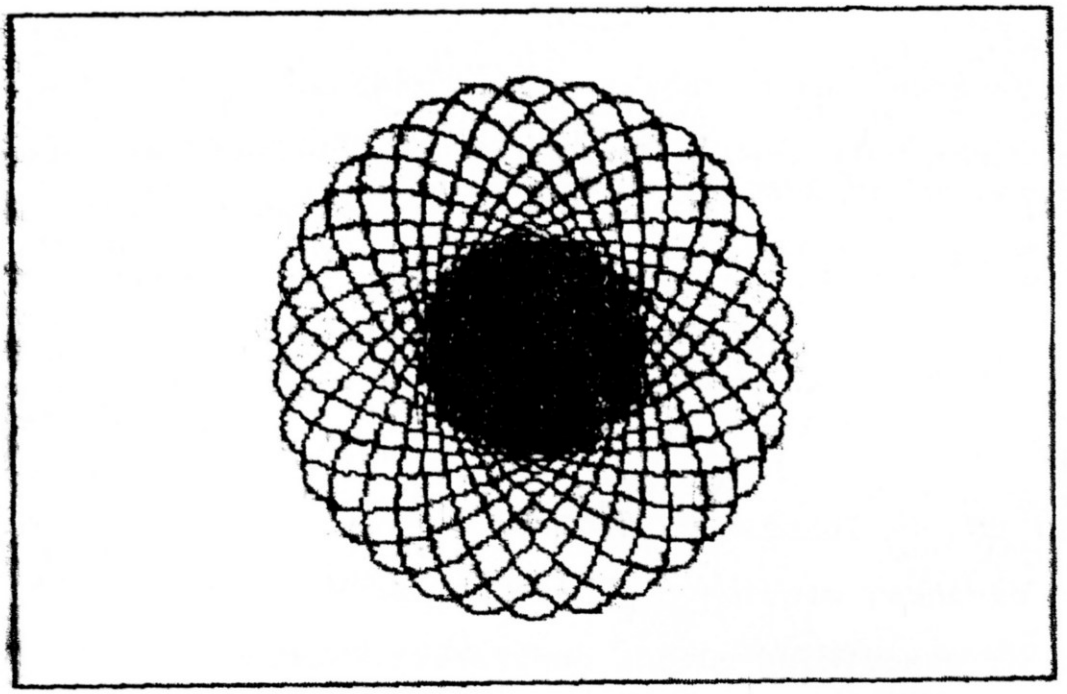
A további paraméterek a rajzolás módját határozzák meg. **Az interpreter az ellipszis egymástól - a középpontjából nézve - <nv> szögtávolságra levő pontjait rajzolja meg, és ezeket egyetlen szakasszal köti össze.** Így ha <nv>-t 120-nak választjuk, akkor az 'ellipszis' egy háromszög lesz! Ha <nv>-t nem adjuk meg, akkor az interpreter 2-vel számol. A <ks>,<vs> értékek az ellipszis (illetve a neki megfelő töröttvonal) első, illetve utolsó pontjának az ellipszis nagytengelyétől mért szögtávolságát adják meg. Ha például <ks>=0 és <vs>=90, akkor az interpreter csak egy negyed ellipszist rajzol. Ha nem adjuk meg ezeket a paramétereket, akkor az interpreter a <ks>=0, <vs>=360 értékekkel számol, azaz a teljes ellipszist megrajzolja. Valamennyi szöget fokokban kell megadni.

Példák:

(i)
```
GRAPHIC 1,1
CIRCLE 1,160,100,45,45
CIRCLE 1,160,100,48,100
CIRCLE 0,160,100,48,100
CIRCLE 1,160,100,48,100,,90
CIRCLE 1,100,100,40,40,,,120
CIRCLE 1,100,100,50,50,,,45
```

(ii)
```
GRAPHIC 1,1
LOCATE 160,100
CIRCLE 1,,45,45
```

(iii)
```
10 GRAPHIC 1,1
20 FOR I=0 TO 14
30 CIRCLE 1,159,100,30,80,,,12*I
40 NEXT I
50 PAINT 1,159,100
60 GETKEY A$
70 GRAPHIC 0
```



(iii) eredménye

Az (i) alatti parancsokat egymás után kiadva a CIRCLE valamennyi lehetőségét kipróbálhatjuk. Az utolsó három parancs azonban nem ellipszist, hanem négyzetet, háromszöget, illetve hatszöget fog rajzolni, az <nv> paraméter megadása miatt.

A (ii) alatti példa az (i) alatti első ellipszis rajzolását ismétli meg oly módon, hogy a kör középpontja a grafikus kurzor helye, amit előzőleg egy LOCATE utasítás segítségével beállítottunk.

A (iii) példa egy néhány körből álló rajzot készít.

Hibalehetőségek: A <színtípus> értéke (0-3) összhangban kell, hogy legyen a grafikus üzemmóddal. Ha nem, ?ILLEGAL QUANTITY ERROR hibajelzést kapunk. Az aritmetikai kifejezések kiértékelése közben számos hibalehetőség adódik. A paraméterek értéke nem lehet negatív.

CLOSE

C-64-es mód

Rövidítés: cl0 Token: $A0 (160)

Mód: mind parancs, mind program módban használható

Lezárja a paraméterként megadott logikai file-t. (Lásd C-64 54. oldal!)

Szintaxis: **CLOSE** <log.fileszám>

CLR

C-64-es mód

Rövidítés: cL Token: $9C (156)

Mód: mind parancs, mind program módban használható.

Törli a program valamennyi változóját, a vermet és végrehajt egy RESTORE-t is. (Lásd C-64 55. oldal!)

Szintaxis: **CLR**

CMD

C-64-es mód

Rövidítés: cM Token: $9D (157)

Mód: mind parancs, mind program módban használható.

Módosítja az elsődleges output perifériát. (Lásd C-64 55. oldal!)

Szintaxis: **CMD** <log.fileszám> {<nyomtatási kép>}

COLLECT

Rövidítés: colL Token: $F3(243)

Mód: mind parancs, mind program módban használható.

Újraszervezi a lemezegységen levő file-okhoz tartozó blokkokat.

Szintaxis: **COLLECT** {**D**<meghajtó>}{**ON U**<egység>}

Az utasítás az <egység> hardverszámú lemezegység <meghajtó> sorszámú meghajtójában levő lemez blokkjait szervezi újjá. Ha elhagyjuk a hardver egységszámot, akkor az utasítás a 8-as lemezegységre vonatkozik. Ha elhagyjuk a meghajtó számát, akkor a COLLECT mindig a 0-ás meghajtóra vonatkozik. Az utasítás a véletlen file-ok blokkjait törli. Ilyen lemezekre tehát ne alkalmazzuk az utasítást.

Példák:

```
  (i)     COLLECT: PRINT DS$
 (ii)     COLLECT D1
(iii)     COLLECT D1 ON U10
```

Az utasítás (i) alatt használt formája a leggyakoribb. A géphez egyetlen egymeghajtós lemezegységet kapcsoltunk, aminek hardver száma 8. Az utasítás a lemez blokkjait újjászervezi.

A (ii) alatti formát abban az esetben használhajuk, ha a C-128-hoz egy 8-as hardver számú duál lemezegységet csatlakoztattunk. Ekkor a parancs az 1-es számú meghajtóban levő lemezt szervezi újjá.

A (iii) esetben gépünkhöz egy 10-es egységszámú duál lemezegységet csatlakoztattunk, s a parancs annak az 1-es meghajtójában levő lemez blokkjait szervezi újjá.

Hibalehetőségek: Ha a parancsban szereplő egységszámú lemezegység nincs a rendszerben, akkor ?DEVICE NOT PRESENT ERROR hibajelzést kapunk. Ha egymeghajtós lemmezegység esetén az 1-es meghajtóra hivatkozunk, akkor nem kapunk hibajelzést, csak a lemezegység lámpája kezd el villogni. A hibacsatorna kiolvasásakor ilyenkor ?DRIVE NOT READY üzenetet kapunk (hibakódja 74). Ha az újjászervezés közben olvasási vagy íráshiba történik, az újjászervezés abbamarad, és a keletkezett hibáról a hibacsatorna olvasásával tájékozódhatunk.

COLLISION

Rövidítés: coL Token: $FE $17 (254,23)

Mód: mind parancs, mind program módban használható.

Utasítja a BASIC interpretert, hogy sprite-ok ütközése esetén milyen BASIC alprogramot hajtson végre.

Szintaxis: **COLLISION** <típus> {,<sorszám>}

A <típus> értéke 1,2 vagy 3 lehet. Az egyes értékek jelentése a következő:

1 Sprite-sprite ütközés
2 Sprite-háttér ütközés
3 Fényceruza pozitiv átmenet

A <sorszám> annak az alprogramnak a kezdő sorszáma, amelyet az esemény bekövetkeztekor végre kell hajtani. Az alprogramból RETURN utasítással kell kilépni.

Példák:

(i)
```
10 COLLISION 2,1200
...
<utasítások>
...
1200 IF (BUMP(2) AND 1)=0 THEN RETURN
1200 : SPRITE 1,0#0
1210 : RETURN
```

Példánkban az a célunk, hogy a megszakító rendszer által mozgatott 1-es sorszámú sprite álljon meg, ha a háttérnek ütközik. Ehhez a program elején (10.sor) kiadunk egy COLLISION utasítást. Ha akármelyik sprite ütközik a háttérrel, akkor az 1200-as sorra kerül a vezérlés. A BUMP függvény segítségével megkérdezzük, vajon az 1-es sprite ütközött-e, s ha igen, akkor megállítjuk.

Hibalehetőségek: Ha <típus> értéke nem megfelelő, akkor ?ILLEGAL QUANTITY ERROR hibajelzést kapunk. Rosszul szerkesztett programok esetén nem a kívánt hatást érjük el. A COLLISION végrehajtásához szükséges ellenőrzéseket utasításonként végzi a rendszer. Gondoskodni kell az ütemezésről.

COLOR

Rövidítés: coL Token: $E7(231)

Mód: mind parancs, mind program módban használható.

Az egyes szintípusok valódi színét állíthatjuk be. Az öt szintípus a következő:

szintípus	jelentés
0	háttér(papír) színe (40 oszlop)
1	grafikus írás (tinta) színe
2	több szín#1
3	több szín#2
4	keret színe
5	szöveg írás színe
6	háttérszín (80 oszlop)

Szintaxis: **COLOR** <szintípus>,<szín>

A <szintípus> a 0-6 intervallumba kell hogy essen. A <szín> a lehetséges 16 színnek megfelelő 1-16 számérték bármelyike lehet.

Példák:

```
  (i)    COLOR 0,3
 (ii)    COLOR 1,I
```

Az (i) példa a háttér színét pirosra változtatja. A (ii) példa a grafikus utasításban használt írás színét az I. színre állítja be.

Hibalehetőségek: Az aritmetikai kifejezések kiértékelése közben számos hibalehetőség adódik. Ha ezek értéke nem esik a jelzett határok közé, akkor ?ILLEGAL QUANTITY ERROR hibajelzést kapunk. Nem megfelelően megválasztott színkombinációk esetén a képernyő olvashatatlanná válik.

CONCAT

Rövidítés: cO Token: $FE $13 (254,19)

Mód: mind parancs, mind program módban használható.

SEQ típusú file-ok összefűzésére szolgál.

Szintaxis:
CONCAT {**D**<meghajtó1>,} <filenév1> TO {**D**<meghajtó2>,} <filenév2>
{**ON U**<egység>}

Az <egység> hardver számú lemezegység <meghajtó1>-ben levő <filenév1> nevű file-ját hozzáfűzi ugyanezen lemezegység <meghajtó2> sorszámú meghajtójában levő <filenév2> nevű file-jához. Mindkettő file SEQ típusú lehet csak. Ha az <egység> és a <meghajtó>-k elmaradnak, akkor a rendszer a 8-as egység 0-ás meghajtójával dolgozik.

```
  (i)    CONCAT "UPDATE" TO "ADATOK"
 (ii)    10 X$="UPDATE"
         20 CONCAT (X$) TO "ADATOK" ON U9
```

Az (i) alatti parancs az "ADATOK" nevű file-hoz hozzáfűzi az "UPDATE" nevű file-t. Mindegyik file a 8-as egység Ø-ás meghajtójában levő lemezen található. Az "UPDATE" tartalma nem változik. A (ii) példa ugyanazt a feladatot végzi el. Mivel az egyik file nevét sztringváltozóban tároljuk, azt kerek zárójelek közé kell tenni. A két file a 9-es lemezegységben levő lemezen kell hogy létezzen.

Hibalehetőségek: Ha nem konstansokat használunk az utasításban és a kifejezést nem tesszük kerek zárójelek közé, akkor ?SYNTAX ERROR Hibaüzenetet kapunk. Ha az "UPDATE" hozzáírása közben a lemezen írás vagy olvasási hiba történik, nem kapunk hibajelzést, csak a lemez lámpája villog. A hiba okát a lemez hibacsatornájának kiolvasásával tudhatjuk meg. Ha valamelyik file nem létezik, akkor ?FILE NOT FOUND hibaüzenetet kapunk.

CONT

C-64-es mód

Rövidítés: nincs Token: $9A (154)

Mód: csak parancs módban használható.

A <STOP> billentyűvel, vagy a STOP utasítással megállított program futását folytatja. (Lásd C-64 57. oldal!)

Szintaxis: **CONT**

COPY

Rövidítés: coP Token: $F4(244)

Mód: mind parancs, mind program módban használható.

Lehetővé teszi egyes file-ok, vagy a lemez valamennyi file-jának másolását.

Szintaxis:
COPY {**D**<meghajtó1>,} <filenév1> **TO** {**D**<meghajtó2>,} <filenév2>
{**ON U**<egység>}

<meghajtó1> és <meghajtó2> értéke a használandó meghajtók sorszámát adja meg. értékük Ø, vagy 1 lehet. Ha elmaradnak, azt az interpreter Ø-val dolgozik. <egység> a lemezegység hardver számát definiálja. Ha nem adjuk meg, az 8-nak számít. <filenév1> a másolandó file neve, <filenév2> pedig a másolati példány neve.

Példák:

```
  (i)    COPY DØ, "REGI" TO D1, "UJ"
         COPY DØ, "REGI" TO "UJ"
         COPY DØ, "PROGRAM" TO D1, "PRG" ON U9

 (ii)    COPY DØ TO D1
(iii)    COPY DØ, "PROGRAM.*" TO D1, "*"
         COPY DØ, "???.PRG" TO D1, "*"
```

Ezek a példák önmagukért beszélnek. A második sorban álló COPY parancs mutatja azt a lehetőséget, hogy egy file-nak ugyanazon a lemezen is létrehozhatjuk - természetesen más néven - a másolatát.

A (ii) alatti parancs a COPY egy speciális használatát mutatja. Ha hiányoznak a file-nevek, akkor a lemezegység **valamennyi file-t egyenként** átmásolja a másik meghajtóban levő lemezre. Ha ezen már voltak file-ok, azok rajta maradnak a lemezen.

A (iii) példa azt mutatja, hogy a file-nevekben szereplő speciális karakterek hatása a COPY utasításnál is érvényesül. Az első sorban levő COPY **valamennyi** "PROGRAM." sztringgel kezdődő nevü file-t - ugyanazzal a névvel - átmásol. Ilyen esetben az új névnek megfelelő sztring egyetlen "*" karakterből kell hogy álljon. A második COPY parancs a D0-ás meghajtóban levő lemezen levő összes olyan file-t átmásolja - változatlan néven - melyek neve pontosan 7 karakter hosszú és az utolsó négy karakter ".PRG".

Hibalehetőségek: Az aritmetikai és sztringkifejezések kiértékelése közben számos hiba adódhat. Ha az aritmetikai kifejezések értéke nem megfelelő, akkor ?ILLEGAL QUANTITY ERROR hibajelzést kapunk. Ha a másolás közben olvasási, vagy íráshiba történik a másolás félbeszakad, és a hibának megfelelő üzenetet kapunk. Ha a hiba jellege olyan, akkor előfordulhat, hogy csak a lemezegység hibalámpája jelzi. Ilyenkor a hibacsatorna olvasásával (?DS$) kaphatjuk meg a hibaüzenetet. Ugyanez a helyzet, ha a másolandó file nincs a lemezen, vagy egy olyan néven akarjuk tárolni a másolatot, amilyen nevű file masolata már létezik.

COS

C-64-es mód

Rövidítés: nincs Token: $BE (190)

Mód: mind parancs, mind program módban használható.

Függvény: az argumentum koszinuszát adja vissza. (Lásd C-64 57. oldal!)

Szintaxis: **COS(** <arit.kif.> **)**

DATA

C-64-es mód

Rövidítés: dA Token: $83 (131)

Mód: mind parancs, mind program módban használható.

A READ utasítással beolvasható adatokat tárolja. (Lásd C-64 58. oldal!)

Szintaxis: **DATA** <konstans-lista>

DCLEAR

Rövidítés: dclE Token: $FE $15 (254,21)

Mód: mind parancs, mind program módban használható.

Inicializálja a lemezegység adott meghajtóját.

Szintaxis: **DCLEAR** {**D**<meghajtó>} {**ON U**<egység>}

A <meghajtó> értéke Ø vagy 1 lehet. Az egységszám a 8-11 intervallumba kell, hogy essen. Ha ez az értékek elmarad, az interpreter a 8-as egységgel dolgozik. Ha a meghajtó száma hiányzik, akkor duál lemezegységek esetén mind a kettőt inicializálja a lemezegység.

Példák:

```
  (i)     DCLEAR
 (ii)     DCLEAR D1 ON U9
(iii)     DCLEAR D(NN)
```

(i) a leggyakoribb használata ennek a parancsnak. Hatására az interpreter inicializálja a 8-as lemezegység (mindkét) meghajtóját. A (ii) példában a 9-es hardver számú duál lemezegység 1-es meghajtóját inicializálja a rendszer. (iii) mutatja, hogyha a paraméterek valamelyike nem konstans, akkor kerek zárójelek közé kell tenni.

Hibalehetőségek: Ha a lemezegység valamiért nem képes végrehajtani a parancsot, előfordulhat, hogy nem kapunk hibajelzést. Ilyenkor a hibacsatorna kiolvasásával tudhatjuk meg, mi is okozta a hibát.

DCLOSE

Rövidítés: dC Token: $FE $ØF (254,15)

Mód: mind parancs, mind program módban használható.

Lezárja a lemezen DOPEN vagy OPEN utasítással megnyitott file-okat.

Szintaxis: **DCLOSE**{**#**<lfszám>} {**ON U** <egység>}

Az <egység> hardver számú lemezegységen <lfszám> logikai fileszám alatt megnyitott file-t zárja le. Ha az <lfszám> elmarad, akkor valamennyi file-t lezárja. Ha nem adjuk meg az <egység> paramétert, az utasítás a 8-as egységre vonatkozik.

Példák:

```
DCLOSE#2
DCLOSE#2 ON U8
DCLOSE ON U9
```

Az első két példának ugyanaz a hatása, a 2-es logikai számmal megnyitott file-t a rendszer lezárja a 8-as egységen. A harmadik példa a 9-es egységen az összes megnyitott file-t lezárja.

Hibalehetőségek: Akkor sem kapunk hibajelzést, ha egy olyan file-t zárunk le, ami még nem volt megnyitva. Az utasítás a BASIC táblázatai alapján zárja le a lemezen a file-okat. Ha ezek valamiért törlődtek, akkor a lemezen magán nem lesznek lezárva a file-ok! Ilyenkor célszerő a DCLEAR utasítást kiadni. Ha erre sem zárodnak le a file-ok, akkor valamilyen lemezmonitor programmal segíthetünk csak.

DEC

Rövidítés: nincs Token: $D1(209)

Mód: mind parancs, mind program módban használható.

A függvény az argumentumaként szereplő sztringet hexadecimális számnak tekinti, és annak decimális értékével tér vissza.

Szintaxis: **DEC(**<sztring.kif.>**)**

A sztring maximum 4 hexadecimális számjegyet(0-9,A-F) tartalmazhat.

Példák:

```
  (i)     PRINT DEC("AF") : REM =175
          PRINT DEC("1000") : REM =4096
 (ii)     10 DO WHILE X<256
          20 PRINT X,HEX$(X),DEC(HEX$(X))
          30 LOOP
(iii)     210 DO UNTIL C$=CHR$(13)
          220 INPUT A$,B$: GETKEY C$
          230 PRINT HEX$(DEC(A$)+DEC(B$))
          240 GETKEY C$
          250 LOOP
```

Az (i) alatti két parancs DEC közvetlen hatását példázza. A (ii) alatti rövid program azt mutatja be, hogy a DEC függvény a HEX$ függvény inverze. A program az első 255 számot írja ki decimális, hexadecimális, majd ismét decimális alakban (20. sor).

A (iii) alatti példa segítségével hexadecimális számok összeadását gyakorolhatjuk. A program indítás után két hexadecimális számot vár inputként. Ha ezeket a számokat fejben (vagy papíron) összeadtuk, nyomjunk meg egy tetszőleges billentyűt, a gép kiírja a választ. Ha ezután a <RETURN> billentyűt nyomjuk meg, a program futása abbamarad. Bármely más billentyű megnyomása esetén újabb két szám összeadását végeztethetjük el.

Hibalehetőségek: A sztring kifejezés kiértékelése közben számos hibalehetőség adódhat. Ha a sztring értéke nem esik a $0000-$FFFF intervallumba, akkor ?ILLEGAL QUANTITY ERROR hibajelzést kapunk.

DEF

C-64-es mód

Rövidítés: dE fn Token: DEF $96 (150)
FN $A5 (165

Mód: csak program módban használható.

Egyváltozós aritmetikai függvényt definiál. (Lásd C-64 59. oldal!)

Szintaxis: **DEF FN**<változó>(<változó>)=<arit.kif.>

DELETE

Rövidítés: deL Token: $F7(247)

Mód: csak parancs módban használható.

Több programsort törölhetünk egyszerre.

Szintaxis: **DELETE** <sorszám.1>{-{<sorszám.2>}}

<sorszám.1> az első **már**, <sorszám.2> pedig az utolsó **még** törlendő sor száma. A sorszámok csak előjel nélküli egész számok lehetnek.

Példák:

```
DELETE 168
DELETE 1000-
DELETE 1000-2400
DELETE -300
```

A példák önmagukért beszélnek. A paraméterek használata megegyezik a LIST parancséval. Ha megadjuk a törlendő első, illetve utolsó sor számát, akkor ezek a sorok is törlődnek. A harmadik sorban szereplő DELETE parancs tehát az 1000., illetve a 2400. sort is törli (és az összes köztük levőt).

Hibalehetőségek: Nincs, de ha nem vigyázunk, nem kívánt sorokat is törölhetünk. Mint minden törlő jellegű paranccsal, ezzel is óvatosan kell bánni.

DIM

C-64-es mód

Rövidítés: dI Token: $86 (134)

Mód: mind parancs, mind program módban használható

Tömbök deklarálására szolgál. (Lásd C-64 61. oldal!)

Szintaxis: **DIM** <tömbelem-lista>

DIRECTORY

Rövidítés: diR Token: $EE(238)

A lemezegység katalógusát a képernyőre listázza.

Szintaxis:
DIRECTORY {**D**<meghajtó>} {, **U**<egység>} {, <filenév>}

<meghajtó> a meghajtó sorszámát adja meg. értéke csak 0, vagy 1 lehet. A második aritmetikai kifejezés a lemezegység hardver száma. <filenév> a listázandó file-ok nevét adja meg. Használhatjuk a * és ? karaktereket. Ha elmarad, valamennyi file-t kilistázza az interpreter.

Példák:
```
DIRECTORY
DIRECTORY D1, U9, "BASIC*"
```

Az első sor a 8-as lemezegység 0-ás meghajtójában levő lemez katalógusát listázza a képernyőre. A második sorban álló parancs hatására a 9-es lemezegység 1-es meghajtójában levő lemez "BASIC"-kel kezdődő nevű file-jai kerülnek csak kilistázásra.

Hibalehetőségek: Az aritmetikai és sztringkifejezések kiértékelése közben számos hiba történhet. Ha a lemezegység jelez hibát, azt csak a hibacsatorna kiolvasásával kapjuk meg. A DIRECTORY-t nem használhatjuk CMD-vel együtt. De ha át is irányítottuk a rendszer outputot a nyomtatóra, a DIRECTORY azt visszairányítja a képernyőre.

DLOAD

Rövidítés: dL Token: $F0(240)

Mód: mind parancs, mind program módban használható.

A lemezegység adott file-ját tölti be a memóriába.

Szintaxis:
DLOAD <filenév> {,**D**<meghajtó>}{,**U**<egység>}

A <filenév> a betöltendő file nevét, <meghajtó> a meghajtó sorszámát, <egység> pedig a lemezegység hardver számát jelenti. Ha a file neve ? vagy * karaktert tartalmaz, akkor az első - a file névnek megfelelő - file-t tölti be az interpreter.

Példák:
```
  (i)    DLOAD "BEMUTATO
         DLOAD "BEMUTATO",D1
 (ii)    A$="BEMUTATO": DLOAD (A$),D1
(iii)    DLOAD "PROGRAM",D0 ON U9
```

Az (i) alatt felsorolt két parancs a DLOAD leggyakoribb használatát mutatja be. Az első esetben a "BEMUTATO" nevű program a 8-as lemezegység 0-ás meghajtójában levő lemezről, a második esetben ugyanezen lemezegység 1-es meghajtójáról töltődik be.

A (ii) alatti példa azt mutatja, hogy a DLOAD paraméterei közt nem csak sztring konstansok szerepelhetnek. Ebben az esetben azonban a kifejezést kerek zárójelek közé kell tenni.

A (iii) példában a töltés a 9-es lemezegység 0-ás meghajtójáról történik. A D0 elhagyható lenne.

Hibalehetőségek: Az aritmetikai és sztring kifejezések kiértékelése közben számos hiba történhet. Ha a meghajtó sorszáma és a lemezegység hardver száma nem esik a lehetséges értékek intervallumába, akkor ?ILLEGAL QUANTITY ERROR hibajelzést kapunk. Ha a szóban forgó lemezegység nincs a rendszerben, akkor ?DEVICE NOT PRESENT ERROR hibajelzést kapunk. Ha az adott nevű file nincs a lemezen, akkor ?FILE NOT FOUND ERROR hibajelzést kapunk.
A töltés közben előforduló hibákról általában nem kapunk jelzést, csak a lemezegység lámpája jelzi a hibát. A hiba természetéről a lemez hibacsatornájának kiolvasásával kapunk információt.

DO

Rövidítés:nincs Token: $EB(235)

Mód: mind parancs, mind program módban használható.

Lehetővé teszi a DO és a LOOP utasítások közti programrész ismételt végrehajtását. A **kilépés feltételét** a DO-t, vagy a LOOP-ot követő WHILE és UNTIL utasításokban adhatjuk meg. A ciklusból az EXIT utasítás segítségével bármikor kiléphetünk.

Szintaxis: **DO** { **UNTIL** / **WHILE** } <logikai.kif.>]

A DO és a hozzátartozó LOOP utasítás közti programrészt **ciklusmagnak** hívjuk. A DO vagy a LOOP utasításban szereplő UNTIL, illetve WHILE utasításokat követő logikai kifejezéseket a **kilépés feltételének** nevezzük. Ezeknek kell a WHILE esetében hamissá, az UNTIL esetében igazzá válni ahhoz, hogy a ciklus lejárjon. A ciklusmagban elhelyezett EXIT utasításokat kilépési pontoknak hívjuk. Egy ilyen utasítás végrehajtása a ciklusból való azonnali kilépést eredményez.

Példák: (i)
```
10 DO WHILE A$="": GET A$: LOOP
10 DO UNTIL A$<>"": GET A$: LOOP
```

(iia)
```
FOR I=KE TO VE STEP LE
...
NEXT I
```

(iib)
```
I=KE: DO
...
I=I+LE
LOOP WHILE (LE<0 AND I>=VE) OR (LE>=0 AND I<=VE)
```

```
(iic)    I=KE: DO WHILE (LE<0 AND I>=VE) OR (LE>=0 AND I<=VE)
         ...
         I=I+LE
         LOOP
```

Az (i) példa a DO egy tipikus használatát mutatja be. A program végrehajtása addig áll egy helyben, míg meg nem nyomunk egy billentyűt. A két 10. sor csak az UNTIL/WHILE megválasztásában tér el egymástól. Az első esetben a ciklus csak akkor ismétlődik, ha a WHILE-t követő logikai feltétel igaz, azaz A$ egy üres sztringet tartalmaz. A második esetben a ciklus akkor ismétlődik csak, ha az UNTILT követő feltétel nem igaz, azaz A$<>"" hamis. Mindkettő ekvivalens a 10 GETKEY A$ programsorral.

A (iia) alatti példa egy FOR ciklust tartalmaz. A (iib) alatti példa ennek DO ciklussal való helyettesítését mutatja be. A ciklusból való kilépést ellenőrző WHILE azért olyan bonyolult, mert nem ismerjük LE előjelét. Annak ismeretében az OR valamelyik tagját elég beírni. Ha például LE>0, akkor a LOOP igy írható:

```
LOOP WHILE I<=VE
```

A (iic) egyetlen apróságban tér el a FOR ciklustól. A FOR ciklus legalább egyszer mindig lefut. A WHILE előrehozása a DO-ba azt eredményezi, hogy KE>VE esetén a ciklus egyetlen egyszer sem fut le. Bizonyos esetekben ez nagyon hasznos lehet.

A FOR átírása is mutatja, hogy a DO ciklust elsősorban olyan esetekben használjuk, amikor ciklusváltozóra nincs szükség, vagy a ciklusváltozók közül nem tudunk egy meghatározót kiválasztani. A DO ciklus lényegesen gyorsabban fut, mintha IF...THEN GOTO... alakú struktúrával valósítanánk meg.

A DO utasítások tetszőleges mélységben egymásba skatulyázhatók:

```
DO
 ...
 DO
  ...
 LOOP UNTIL ...
 ...
LOOP WHILE ...
```

A DO utasítás végrehajtása az azt esetleg követő kilépési feltétel kiértékelésével kezdődik. Ha ez alapján ki kell lépni a ciklusból, akkor az interpreter a DO utasítást **követően** elkezdi az első LOOP utasítást keresni, s ha megtalálta, az azt követő első utasításra adja a vezérlést. Ha még nem járt le a ciklus, akkor a visszatérést biztosító információk a verembe kerülnek, s az interpreter rátér a DO-t követő első utasítás végrehajtására.

Hibalehetőségek: A DO-t esetleg követő kilépési feltétel kiértékelése közben számos hiba adódhat. A FOR, GOSUB is használja a vermet, így az egyes struktúrák egymásba skatulyázásának határt szab a verem nagysága. Ha már nincs elég hely a veremben, akkor ?OUT OF MEMORY ERROR hibajelzést kapunk. Ha a ciklus lejárt, és az interpreter nem találta meg a megfelelő LOOP utasítást, akkor ?LOOP NOT FOUND ERROR hibajelzést kapunk. Rosszul strukturált programok gyakran adják ezt a hibajelzést.

DOPEN

Rövidítés: dO Token: $FE $0D (254,13)

Mód: mind parancs, mind program módban használható.

Lemezes file-ok megnyitására szolgál.

Szintaxis:
DOPEN#<lfszám>,<filenév> {,**L**<rek.hossz>}
{,**D**<meghajtó>}{,**U**<egység>}{,**W**}

Az interpreter <lfszám> alatt megnyit egy <filenév> nevű file-t a <meghajtó> és <egység> által meghatározott lemezegységen. A megnyitandó file egyéb paramétereit az L és a W opció használata határozza meg.

Ha sem az L, sem a W opció nem szerepel, akkor egy már meglevő file-t nyitunk meg. Ha ennek típusa SEQ, akkor a file-t csak olvashatjuk. Ha a file típusa REL, akkor írhatjuk is és olvashatjuk is.

Ha az L opciót használjuk, akkor egy REL típusú file-t akarunk létesíteni. Az L paraméterben <rek.hossz> ilyenkor a REL file rekordhosszát adja meg. Ennek a 4-254 intervallumba kell esnie. Ha a relativ file már létezett, akkor a lemezegység ellenőrzi a rekordhosszt, s csak akkor nyitja meg, ha azonos. A @ karaktert relativ file esetében nem használhatjuk.

Ha a W opció szerepel, akkor egy új SEQ típusú file-t akarunk írásra megnyitni. Ebben az esetben a @ karaktert használhatjuk a már létező <filenév> file felülírására.

Az L és a W paramétereket nem használhatjuk egyszerre.

Példák:

```
  (i)     DOPEN#2,"PROBA"
 (ii)     DOPEN#3,"ADATOK",L234
(iii)     DOPEN#2,"PROBA",U9,W
 (iv)     DOPEN#2,"@PROBA",W
```

Az (i) példában a "PROBA" nevű szekvenciális file-t nyitjuk meg olvasásra. A (ii) példában egy relativ file-t létesítünk, amelynek rekordhossza 234. (iii) a "PROBA" nevű file-t hozza létre a 9-es lemezegységben levő lemezen, s nyitja meg írásra. (iv)-et akkor használjuk, ha a "PROBA" nevű file-t felül akarjuk írni.

Hibalehetőségek: Ha olvasásra nemlétező file-t nyitunk meg, akkor ?FILE NOT FOUND ERROR hibaüzenetet kapunk. Hasonlóan, ha egy létező SEQ típusú file-t írásra kisérlünk meg megnyitni, akkor ?FILE EXISITS ERROR hibaüzenetet kapunk.

DRAW

Rövidítés: dR Token: $E5(229)

Mód: mind parancs, mind program módban használható.

A nagyfelbontású képernyőre rajzol pontokat, szakaszokat.

Szintaxis:
DRAW {<színtípus>},{<x1>,<y1>} **TO** <x2>,<y2>...

A megrajzolt pontok színe <színtípus> színű lesz. Ha a paraméter elmarad, akkor az interpreter az 1-es színtípust (írás színe) használja. értékének a kiválasztott grafikus móddal összhangban kell lennie. <x1>,<y1> a kezdőpont koordinátái. Ha elmarad, akkor a kezdőpont a grafikus kurzor pillanatnyi helyzete lesz. A TO-val bevezetett részek a szakasz végpontjának koordinátáit adják meg. **Mind a kezdőpont, mind a TO-s rész tetszés szerinti sokszor ismételhető.** Igy egyetlen DRAW utasítással több töröttvonal is rajzolható.

Példák:

```
  (i)     DRAW 1,100,160
 (ii)     DRAW 0,100,160
(iii)     DRAW 1,0,0,319,199
(iv)      DRAW 1,100,100 TO 150,100 TO 150,150 TO 100,150

(v)        50 REM RESZEG TENGERESZ
          100 GRAPHIC 1,1: LOCATE 160,100
          150 DO
          200 X=319*RND(0): Y=199*RND(0)
          250 DRAW ,TO X,Y
          300 GET A$
          350 LOOP WHILE A$=""
          400 GETKEY A$
          450 GRAPHIC 0
```

Az (i) alatti példa a nagyfelbontású képernyőre rajzol egyetlen pontot. A parancs kiadása előtt ki kell adni a megfelelő GRAPHIC parancsot, különben hibajelzést kapunk. A (ii) példa ugyanezt a pontot törli. A (iii) példa a nagyfelbontású képernyőn keresztben meghúzza az egyik átlót.

A (iv) alatti példában a DRAW paramétereinek ismételhetőségét használjuk fel egy négyzet megrajzolására.

(v)-ben egy részeg tengerész bolyong fel-alá a képernyőn, útjának megrajzolására használjuk a DRAW utasítást.

Hibalehetőségek: Az aritmetikai kifejezések kiértékelése közben számos hiba történhet. Ha a <színtípus> értéke nem felel meg a használt grafikus módnak, akkor ?ILLEGAL QUANTITY ERROR hibajelzést kapunk. A paraméterek elhagyása esetén ügyelni kell a vesszők megfelelő elhelyezésére. Ha nem adtuk még ki a megfelelő GRAPHIC utasítást, akkor ?NO GRAPHICS AREA ERROR hibajelzést kapunk.

DS és DS$

Fenntartott BASIC változók

A DS valós, illetve a DS$ sztring változókat a C-128 speciálisan kezeli. Bármelyikükre való hivatkozás a lemezegység hibacsatornájának olvasását eredményezi. DS-ben a hiba kódja, DS$-ban pedig a teljes üzenetet kapjuk vissza. A rendszer mindig az utoljára használt lemezegység hibacsatornáját olvassa, kezdetben a 8-as egységszámúét. A hibaüzenetekről, a hibacsatornáról részletesen az 5. fejezetben lesz szó.

Példák:
```
  (i)     BACKUP D0 TO D1: ? DS$
 (ii)     SCRATCH "PROBA*": ? DS$
```

Mindkét példa azt szemlélteti, hogy általában minden egyes lemezművelet után célszerű a hibacsatornát kiolvasni. Ez program módban természetes, hiszen ilyenkor a lemezegység lámpájára sem hagyatkozhatunk. (i) alatt egy teljes lemez másolása után olvassuk ki a hibacsatornát és íratjuk ki a képernyőre az üzenetet.

A (ii) példában a "PROBA" névvel kezdődő valamennyi file-t töröljük a lemezről. Ilyenkor mindenképpen célszerű a hibacsatorna kiolvasása, hogy megtudjuk, pontosan hány file-t is töröltünk.

Hibalehetőség: Nincs.

DSAVE

Rövidítés: dS Token: $EF(239)

Mód: mind parancs, mind program módban használható.

A memóriában tárolt BASIC program lemezre történő kiírását (mentését) végzi.

Szintaxis:
DSAVE <filenév> {,**D**<meghajtó>}{,**U**<egység>}

A <filenév> annak a file-nak a neve, ahová a memóriában levő programot menteni kell. Abban az esetben, ha egy már meglevő file tartalmát akarjuk felülírni, akkor a filenévnek a @ jellel kell kezdődnie. A <meghajtó> és az <egység> aritmetikai kifejezések értéke a meghajtó, illetve a lemezegység hardver számát adja meg.

Példák:
```
  (i)     DSAVE "SZAMLAZO
 (ii)     DSAVE "@SZAMLAZO
(iii)     DSAVE "RUTINOK",U9
(iv)      DSAVE "PROGRAMOK",D1,U9
```

Az (i) példa a DSAVE egyik leggyakoribb használatát mutatja. A memóriában levő BASIC program teljes egészében a "SZAMLAZO" nevü

lemezes file-ba kerül. (A záró " jelet nem is kell kiírni.) A lemezegység aktivizálásakor a képernyőn a

SAVING "0:SZAMLAZO"

felirat jelenik meg. A READY. üzenet megjelenése jelzi, hogy a kiírás befejeződött.

(ii)-ben a memóriában levő programot a már meglevő "SZAMLAZO" nevű file-ba írjuk ki. A file eredeti tartalma természetesen elvész. Az utasítás akkor is korrektül végrehajtódik, ha a "SZAMLAZO" nevű file még nem létezik a lemezen.

A (iii)-as példa a memória tartalmát a 9-es lemezegység 0-ás meghajtójában levő lemezre menti ki "RUTINOK" file név alatt. Ez a file még nem létezhet a lemezen.

A (iv) példa a D paraméter használatát mutatja. A memória tartalma a 9-es lemezegység 1-es meghajtójában levő lemezre íródik ki.

Hibalehetőségek: A kifejezések kiértékelése közben számos hiba adódhat. Ha a <filenév> nem sztring konstans, akkor kerek zárójelek közé kell tenni:

A$="PROGRAM": DSAVE (A$)

Ellenkező esetben ?SYNTAX ERROR hibaüzenetet kapunk.

Ha a használni kívánt egység nincs a rendszerben, akkor ?DEVICE NOT PRESENT ERROR hibajelzést kapunk. Ha az adott nevű file már létezik, s a név előtt nem szerepel a @ karakter, akkor ?FILE EXISTS ERROR hibaüzenetet kapunk. Ha a lemezre írás közben íráshiba történik, a mentés azonnal befejeződik. A hibát ilyenkor csak a lemezegység lámpája jelzi. A hibacsatorna kiolvasásával megtudhatjuk a hiba pontos okát is.

DVERIFY

Rövidítés: dV Token: $FE $14 (254,20)

Mód: mind program, mind parancs módban használható.

A memóriában levő BASIC programot hasonlítja össze egy lemezes file tartalmával.

Szintaxis: **DVERIFY** <filenév> {,D<meghajtó>}{,U<egység>}

A <meghajtó> értéke csak 0 vagy 1 lehet. A <filenév> nevű file-nak léteznie kell a specifikált lemezen. Ha a D és U paraméterek hiányoznak, akkor a 8-as egység 0-ás meghajtóját használja a rendszer.

Példák:

```
  (i)     DSAVE "PROGRAM"
          DVERIFY "PROGRAM"
 (ii)     DVERIFY "LEPTETO",U9
```

Az (i) példa a DVERIFY leggyakoribb hsználatát mutatja: egy program kiírása után ellenőrizzük, nem történt-e hiba. A (ii) példában az ellenőrizendő file-nak a 9-es egységen kell lennie.

Hibalehetőségek: Ha a memória és a lemezes file tartalma nem egyezik ?VERIFY ERROR hibajelzést kapunk. Ha a lemezes file nem létezik, akkor ?FILE NOT FOUND ERROR hibajlezést kapunk. Ha a file létezik ugyan, de nem PRG típusu, akkor ?FILE TYPE MISMATCH ERROR hbaüzenetet kapunk.

EL és ER

Fenntartott BASIC változók

Mind a két változó a C-128 hibakezelő rendszeréhez tartozik. Ha működése közben az interpreter hibát észlel, a vezérlés a hibakezelő rutinra adódik át. A hibakezelő rutin a BASIC program része is lehet. Ebben az esetben EL a hibát közvetlenül okozó BASIC sor számát, míg ER a hiba kódját tartalmazza. ER értéke az 1-43 intervallumba esik. Ha nem történt hiba, akkor EL=65535, ER=-1.

Példák:

```
(i)     10 TRAP 2000
        20 ?0/0
        30 ?0/1
        40 END
        2000 PRINT "HIBA TORTENT!!!"
        2010 PRINT "HIBA KODJA=";ER
        2020 PRINT "HIBAS SOR SORSZAMA=";EL
        2040 RESUME NEXT
```

Példánk egy hibakezelő rutint mutat be, amelyik nem tesz egyebet, csak kiírja a hiba kódját és helyét. A program futtatásakor a 20. sorban hiba történik, ezért a 2000-es hibakezelő rutinra adódik a vezérlés. A hiba kódja 20 lesz (?DIVISION BY ZERO ERROR), s a hibás sor száma ugyancsak 20. A 2040-es sor elérése után a program futása a 30. sornál folytatódik.

Hibalehetőségek: Nincs.

END

C-64-es mód

Rövidítés: nincs Token: $80 (128)

Mód: mind parancs, mind program módban használható.

Befejezi a program futását. A változók értéke megmarad. (Lásd C-64 62. oldal)

Szintaxis: **END**

ENVELOPE

Rövidítés: eN Token: $FE $0A (254,10)

Mód: mind parancs, mind program módban használható.

A SID hanggenerátor chip egy ADSR szintetizátor. Ennek paramétereinek beállítására szolgál az utasítás.

Szintaxis:
ENVELOPE <sorszám> {,{<fel>}{,{<le>}{,{<ki>}
{,{<el>}{,{<hullám>}{,{<imp>}}}}}}}

A program segítségével 10 burkológörbe definíciót tárolhatunk, s a PLAY, SOUND utasításokban elég a definíció számát megadni. <sorszám> értéke 0-9 lehet, ez adja a definició számát. Az egyes paraméterek jelentése a következő:

<fel>	felfutás (0-15)	<hullám>	hullámforma (0-4)
<le>	lecsengés (0-15)	0	háromszög
<ki>	kitartás (0-15)	1	fűrészfog
<el>	elengedés (0-15)	2	négyszög
<imp>	impulzusszélesség (0-4095)	3	fehérzaj
		4	körmoduláció

Az impulzusszélesség csak négyszögimpulzus esetén érdekes.
A szimmetrikus jelnek 2048 felel meg.

A burkológörbe definíciók kezdeti értéke a következő:

<sorszám>	<fel>	<le>	<ki>	<el>	<hullám>	<imp>	<hangszer>
0	0	9	0	0	2	1536	zongora
1	12	0	12	0	1		harmonika
2	0	0	15	0	0		orgona
3	0	5	5	0	3		dob
4	9	4	4	0	0		furulya
5	0	9	2	1	1		gitár
6	0	9	0	0	2	512	csemballó
7	0	9	9	0	2	2048	orgona
8	8	9	4	1	2	512	trombita
9	0	9	0	0	0		xilophon

Példák:
```
  (i)    ENVELOPE 6,0,9,0,0,2,512
 (ii)    ENVELOPE 3,2,6,15,0,3
```

Az (i) példában a 6-os burkológörbe eredeti definícióját állítottuk vissza. A (ii) példában egy új hangzást próbálunk ki.

Hibalehetőségek: Ha a paraméterek nem esnek a jelzett intervallumba, akkor ?ILLEGAL QUANTITY ERROR hibajelzést kapunk. Előfordulhat, hogy úgy állítjuk be a paramétereket, hogy igen 'csúnya' hangzást kapunk!

ERR$

Rövidítés: eR (a $ már nem kell!) Token: $D3(211)

Mód: mind parancs, mind program módban használható.

Sztringfüggvény, amelyik az argumentumaként megadott sorszámú hiba szövegével tér vissza.

Szintaxis: ERR$(<arit.kif.>)

Az aritmetikai kifejezés értékének az 1-41 intervallumba kell esnie.

Példák:
```
(i)     ? ERR$(2) : REM = "FILE OPEN"
(ii)    10 FOR I=1 TO 41
        20 PRINT I;". HIBA=";ERR$(I)
        30 NEXT I
```

Az első példa eredményeként a megjegyzésben feltüntetett hibaüzenetet kapjuk. A (ii) példa kiírja a képernyőre az összes hibaüzenetet a hiba számkódjával együtt.

Hibalehetőségek: Az aritmetikai kifejezés kiértékelése közben számos hibalehetőség adódik. Ha ennek értéke nem esik az 0-41 intervallumba, akkor ?ILLEGAL QUNTITY ERROR hibajelzést kapunk.

EXIT

Rövidítés: exI Token: $ED(237)

Mód: mind parancs, mind program módban használható.

DO...LOOP ciklusokban kilépési pontok elhelyezésére szolgál.

Szintaxis: EXIT

Az EXIT a ciklusból feltétel nélküli kilépést eredményez. Az interpreter megkeresi az EXIT-et követő első LOOP-ot, s az azt követő első utasításra adja át a vezérlést.

Példák:
```
(i)     10 INPUT A%,B%
        20 DO
        30 M%=B%-INT(B%/A%)*A%
        40 IF M%=0 THEN EXIT
        50 B%=A%: A%=M%
        60 LOOP
        70 ? "LNKO=";A%
```

Az (i) a LOOP-nál látható példa módosítása. Az A%,B% legnagyobb közös osztóját számítja ki, de úgy szerveztük a számítást, hogy a kilépésre a ciklus közepén kerüljön sor.

Hibalehetőségek: Ha az interpreter nem találja meg a LOOP utasítást, akkor ?LOOP NOT FOUND ERROR hibaüzenetet kapunk. Ha LOOP-hoz nem találja meg a megfelelő DO utasítást, akkor ?LOOP WITHOUT DO ERROR hibajelzést kapunk.

EXP

C-64-es mód

Rövidítés: exP Token: $BD (189)

Mód: mind parancs, mind program módban használható.

Függvény: az e alapú exponenciális függvényt számítja ki. (Lásd C-64 63. oldal!)

Szintaxis: **EXP(**<airt.kif.>**)**

FAST

Rövidítés: fA Token: $FE $25 (254,37)

Mód: mind parancs, mind program módban használható.

Az órajelet 2MHz-re állítja és kikapcsolja a VICII chipet. A 80 oszlopos képernyőn a kijelzés változatlan.

Szintaxis: **FAST**

Ha 40 oszlopos képernyőt használunk, a processzor csak 1 MHz-es órajellel tud működni. Ha fel akarjuk gyorsítani a rendszer futását, akkor ezzel a paranccsal érhetjük el. Cserébe, nem látunk semmit a 40 oszlopos képernyőn. Hosszabb számítások esetén lehet hasznos.

Hibalehetőségek: Nincs.

FETCH

Rövidítés: fE Token: $FE $21 (254,33)

Mód: mind parancs, mind program módban használható.

A memóriabővítésről tölt be adatokat a BASIC munkaterületre.

Szintaxis: **FETCH** <db.szám>,<b.cím>,<k.cím>,<szelet>

Az utasítás hatására az interpreter <db.szám> byte-ot átmásol a memóriabővítés <szelet>-nek megfelelő 64K-s szeletéről. A másolást a <k.cím>-től kezdi. A byte-okat az interpreter a BANK utasításban meghatározott szeletre tölti, a <b.cím>-től kezdődően.

Példák:

```
BANK 0
FETCH 1000,1024,40000,7
```

A két parancs hatására a képernyőmemóriába másolódik a memóriabővítés 7. szeletén a 40000-es címnél kezdődő memóriarész.

Hibalehetőségek: A paraméterek értékének értelemszerűen a megfelelő tartományba kell esni. Ha nem, akkor ?ILLEGAL QUNTITY ERROR hibajelzést kapunk. Az utasítás csak akkor hatásos, ha egy memóriabővítő kártyát használunk!

FILTER

Rövidítés: fI Token: $FE $03 (254,3)

Mód: mind parancs, mind program módban használható.

Beállítja a SID chip szűrési paramétereit. A szűrés még nem kezdődik meg.

Szintaxis: **FILTER** {<frek>}{,{<al>}{,{<sáv>}{,{fel}{,{<rez>}}}}}

Az egyes paraméterek jelentése a következő:

<frek>	a szűrő levágási frekvenciája
<al>	aluláteresztő szűrő ki/be kapcsolása
<sáv>	a sávszűrő ki/be kapcsolása
<fel>	a felüláteresztő szűrő ki/be kapcsolása
<rez>	rezonancia

Az <al>,<sáv>,<fel> értéke 0 vagy 1 lehet. 1 jelenti a szűrő bekapcsolását. A levágási frekvenciát meghatározó értéknek a 0-2047 intervallumba kell esnie. A <rez> értékének a 0-15 intervallumba kell esnie. A nagyobb érték nagyobb rezonanciát jelent. Ha érezni is akarjuk a szűrés eredményét, a PLAY utasításban be kell kapcsolni a szűrőt.

Példák:

```
  (i)     FILTER 1900,,,1
 (ii)     FILTER 200,,1,,15
```

Az (i)-es példában a felüláteresztő szűrőt állítottuk be. A levágási frekvencia elég nagy, ezért csak a magas hangok maradnak meg. (Kb. 7000 kHz felett.) (ii)-ben a sávszűrőt kapcsoltuk be, maximális rezonanciával.

Hibalehetőségek: Ha a paraméterek nem esnek a jelzett intervallumba, akkor ?ILLEGAL QUANTITY ERROR hibajelzést kapunk. A SID chip néha kiakad', ilyenkor csak a gép ki, majd újbóli visszakapcsolásával életszthetjük' fel.

FOR C-64-es mód

Rövidítés: fO Token: FOR $81 (129)
stE STEP $A4 (164)
TO $A9 (169)

Mód: mind parancs, mind program módban használható.

Ciklusutasítás. A ciklusváltozó (csak valós!), a kezdőérték, a növekmény és a végérték adható meg. Ha a növekmény elmarad, a rendszer 1-gyel számol. (Lásd C-64 64. oldal!)

Szintaxis: **FOR**<változó>=<kezdőérték>**TO**<végérték>{**STEP**<növekmény>}

FRE C-64-es mód

Rövidítés: fR Token: $B8(184)

Mód: mind parancs, mind program módban használható.

A függvény a szabad memóriaterület nagyságával tér vissza.

Szintaxis: **FRE**(<szelet>)

Ez a függvény már a BASIC V2.0-ban is megtalálható, ott azonban érdektelen, mi is az argumentuma. <szelet> értéke 0 vagy 1 lehet. Ennek megfelelően a 0-ás vagy 1-es lap szabad memóriakapacítását adja vissza. Ha <szelet>=1, akkor még egy ún. szemétgyűjtést is végez, s a felesleges sztringdarabokat 'kidobálja'.

Példák:
(i) ? FRE(0)
(ii) T=FRE(0): DIM X(100,10): ?T-FRE(0)
(ii) ? FRE(1)

(i) kiírja, hogy a program és változók számára mennyi hely van még. (ii) kiírja, hogy egy X(100,10) tömb mennyi helyet foglal a memóriában. (iii) megadja a szabad sztringterület nagyságát, de előbb elvégzi a szemétgyűjtést.

Hibalehetőségek: Nincs.

GET C-64-es mód

Rövidítés: gE Token: $A1(161)

Mód: csak program módban használható.

Az utasításban megadott input file-ból karaktereket olvas be egyesével. Ha a file-t nem adjuk meg, a billentyűzetről olvas. (Lásd C-64 68. oldal!)

Szintaxis: **GET**{#<log.fileszám>,} <változólista>

GETKEY

Rövidítés: getkE Token: nincs

Mód: csak program módban használható.

A program futása felfüggesztődik egészen addig, míg nem nyomunk le egy billentyűt. Ezután ennek ASCII kódja, mint egy egy karakter hosszú sztring a GETKEY-t követő változóba kerül.

Szintaxis: GETKEY <változólista>

A <változólista> egymástól vesszővel elválasztott változókat tartalmazhat.

Példák: (ia) 10 GETKEY A$

(ib) 10 GET A$: IF A$="" THEN GOTO 10

(ii)
```
1000 PRINT "SZAM1=?";: X$=""
1010 GETKEY A$
1020 IF A$=CHR$(13) THEN X=VAL(X$): RETURN
1030 IF A$<"0" OR A$>"9" THEN GOTO 1010
1040 X$=X$+A$
1050 ?CHR$(157)+A$+"?";: GOTO 1010
```

(iii) 260 GETKEY A$: GRAPHIC 0

Az (ia), illetve az (ib) példák a GET és a GETKEY viszonyát mutatják be. A GETKEY - úgy ahogy az (ib) alatti IF utasítás mutatja - egészen addig vár, míg legalább **egy billentyűt le nem nyomunk.**

A GETKEY lehetőséget nyújt ellenőrzött inputra. A (ii) alatt látható alprogram segítségével egész számokat olvashatunk be. Számjegyen kívül a rutin egyéb karaktert nem fogad el. A beírást a <RETURN> leütése fejezi be. A kurzor helyett egy ? látható.

A (iii) példa a GETKEY egy másik tipikus használatát mutatja. A nagyfelbontású képernyőre a program eleje rajzolt valamit. A kép egészen addig látható, míg meg nem nyomunk egy billentyűt. Ekkor a 260-as programsor visszaváltja a karakteres képernyőt.

Hibalehetőségek: A GETKEY A utasítás szintaktikusan helyes. Ha azonban nem számjegy billentyűt nyomunk meg, akkor ?SYNTAX ERROR hibaüzenetet kapunk.

GO64

Rövidítés: nincs Token: nincs

Mód: mind parancs, mind program módban használható.

A Commodore 128 számítógép áttér C-64-es üzemmódba. Az interpreter viiszakérdez: **ARE YOU SURE?**. Ha erre az <Y>, majd a <RETURN> megnyomásával válaszolunk, a gép azonnal áttér C-64-es módba.

Szintaxis: GO64

Hibalehetőségek: Nincs.

GO

C-64-es mód

Rövidítés: nincs Token: $CB (203)

Mód: mind parancs, mind program módban használható.

Feltétel nélküli vezérlésátadás. (Lásd C-64 69. oldal!)

Szintaxis: GO {TO} <sorszám>

GOSUB

C-64-es mód

Rövidítés: goS Token: $8D (141)

Mód: mind parancs, mind program módban használható.

Alprogram (szubrutin) meghívása. Visszatérés a hívás utánra a RETURN utasítással. (Lásd C-64 70. oldal!)

Szintaxis: GOSUB <sorszám>

GOTO

C-64-es mód

Rövidítés: gO Token: $89 (137)

Mód: mind parancs, mind program módban használható.

Feltétel nélküli vezérlésátadás. (Lásd C-64 72. oldal!)

Szintaxis: GOTO <sorszám>

GRAPHIC

Rövidítés: gR Token: $DE(222)

Mód: mind parancs, mind program módban használható.

Segítségével beállíthatjuk a kijelzési módot.

Szintaxis: GRAPHIC <mód> {,{<törlés>},<szöveg>}
GRAPHIC CLR

Első paraméterének értéke megadja, hogy a kijelzés mód milyen legyen:

érték	Kijelzési mód
0	40 oszlopos, normál, karakteres üzemmód
1	40 oszlopos, normál, nagyfelbontású üzemmód
2	40 oszlopos, normál, vágott képernyő
3	40 oszlopos, többszínű, nagyfelbontású üzemmód
4	40 oszlopos, többszínű, vágott képernyő
5	80 oszlopos, normál, karakteres képernyő

A <törlés> értéke 0 vagy 1 lehet. Ha 1, az érintett képernyők törlődnek. Ha 0, akkor az előző tartalmuknak megfelelő információ jelenik meg. Ha a paramétert nem adjuk meg, az 0-nak számít.

A <szöveg> paraméter értéke 0-24 lehet. Megadja, hogy vágott képernyő használatakor hányadik sorban kezdődjék a szöveges képernyő. Ha nem adjuk meg, a rendszer 19-el számol. Ez 5 szövegsort ad. (Lásd 6.1 paragrafust!)

Példák:
```
(i)     GRAPHIC 1,1
(ii)    GRAPHIC 2,1,24
(iii)   GRAPHIC 3

 (iv)   GRAPHIC CLR
```
Úgy hisszük a példák önmagukért beszélnek. Az első két esetben a képernyők törlődnek is, míg a harmadik esetben nem. A (ii) példában **vágott képernyőt** használunk. Szöveg írására csak az utolsó két sort használhatjuk.

A grafikus képernyő 9Kbyte helyet foglal a memóriában. Ha valamilyen formában használjuk, akkor a BASIC program ennyivel lejjebb fog kezdődni a memóriában és az interpreter a program eltolását automatikusan elvégzi. A (iv) alatti alak újra felszabadítja ezt a területet, s a programot visszatolja.

Hibalehetőségek Az aritmetikai kifejezés kiértékelése közben számos hiba történhet. Ha értékük nem esik a jelzett intervallumba, akkor ?ILLEGAL QUANTITY ERROR hibajelzést kapunk.

GSHAPE

Rövidítés: gS Token: $E3(227)

Mód: mind parancs, mind program módban használható.

Az SSHAPE utasítással definiált - téglalap alakú - képet másolhatjuk vissza a nagyfelbontású képernyő tetszőleges helyére.

Szintaxis: **GSHAPE** <sztringváltozó> {,{<x>,<y>{,<mód>}}}

A téglalap jobb felső sarkának koordinátáit az <x>,<y> koordinátapár határozza meg. Ha elhagyjuk, akkor a grafikus kurzor pillanatnyi helyével számol az interpreter. A <mód> paraméter a képernyőre való kiírás módját határozza meg. Ha a képernyő egy adott

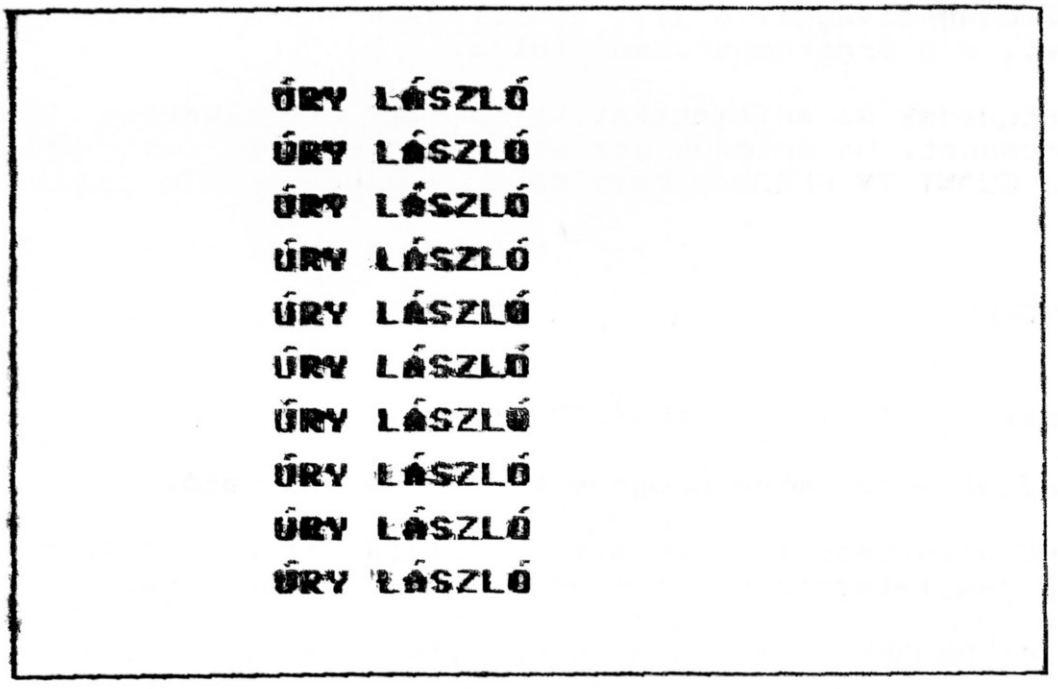
(raszter)pontjának az értéke R, s a GSHAPE utasításban szereplő sztring ennek a pontnak megfelelő bitjének értéke Y, akkor a <mód>-tól függően R új értéke a következő lesz:

<mód>	a pont(R) új értéke	
0	Y	(másolás)
1	1-Y	(inverz alak)
2	Y OR R	(logikai vagy)
3	Y AND R	(logikai és)
4	Y EOR R	(logikai kizáró vagy)

Ha a <mód> értékét nem adjuk meg, akkor az interpreter 0-val számol tovább.

Példák:
```
10 GRAPHIC 1,1
 20 CHAR ,1,1,"A"
 30 DRAW ,12,6 TO 14,5
 40 SSHAPE A$,8,5,16,16
 50 CHAR ,1,1,"O"
 60 SSHAPE O$,8,5,16,16
 70 CHAR ,1,1,"U"
 80 SSHAPE U$,8,5,16,16
 90 GRAPHIC 1,1
 95 FOR X=3 TO 21 STEP 2
100 GSHAPE U$,80,8*X-3: CHAR ,11,X,"RY L"
110 GSHAPE A$,120,8*X-3: CHAR ,16,X,"SZL"
115 GSHAPE O$,152,8*X-3
116 NEXT X
120 GETKEY A$: GRAPHIC 0
```

A fenti program futásának eredménye:



Hibalehetőségek: A sztring- és aritmetikai kifejezések kiértékelése közben számos hiba adódhat. Ha a <mód> értéke nem esik a 0-4 intervallumba, akkor ?ILLEGAL QUANTITY ERROR hibajelzést kapunk.

HEADER

Rövidítés: heA Token: $F1(241)

Mód: mind parancs, mind program módban használható.

Lehetőséget nyújt lemezek megformázására és törlésére.

Szintaxis:
HEADER <lemeznév>, D<meghajtó> {,I<id>} {,U<egység>}

A <lemeznév> sztring a lemez új neve lesz. Ha nem sztring konstans, akkor kerek zárójelek közé kell tenni. A két aritmetikai kifejezés a meghajtó, illetve a lemezegység hardver egységszáma. Az I paraméterben szereplő <id.> két karakteres azonosító. Ha elmarad, az utasítás nem formázza meg, csak törli a lemezt. Újonnan vásárolt lemezeket - kivéve természetesen ha program van rajtuk - az I paramétert is használó HEADER utasítással kell megformázni.

Példák:
```
 (ia)     HEADER "PROGRAMOK",D0
 (ib)     A$="PROGRAMOK"
          HEADER (A$),D0
 (ii)     HEADER "PROGRAMOK",D0,IAA
(iii)     HEADER "ADATOK.2",D1 ON U9
```

Az (ia) példa a 8-as lemezegység 0-ás meghajtójában levő - már megformázott - lemez tartalmát törli, és a lemezt "PROGRAMOK" névre nevezi át. Az (ib)-ben alatt látható két parancs ugyanezt a feladatot oldja meg, de a HEADER parancsban most sztringet használunk. Ennek megfelelően kerek zárójelek közé kell tenni.

A (ii) példa a (8-as lemezegység) 0-ás meghajtójában levő lemezt formázza meg. Ezt követően a lemez már használható programok, illetve adatok tárolására.

(iii) az egységszám megadására mutat példát.

Tekintettel arra, hogy a HEADER parancs a lemezen esetleg rajta levő információt teljes egészében megsemmisíti, a HEADER utasítás kiadása után az interpreter visszakérdez:

ARE YOU SURE?

Ha erre az <Y> megnyomásával válaszolunk, az utasítást az interpreter végrehajtja. Ha az <N> megnyomásával válaszolunk, az utasítás figyelmen kívül marad.

Hibalehetőségek: Ha a <sztring> nem sztring konstans, és nem tettük kerek zárójelbe, akkor ?SYNTAX ERROR hibajelzést kapunk. Ha olyan lemezegységre hivatkozunk, ami nincs a rendszerben, akkor ?DEVICE NOT PRESENT ERROR hibajelzést kapunk. Vigyázzunk arra, hogy csak olyan lemezt töröljünk le, amelyikre nincs szükségünk! Ha a lemezen nincs kivágva az írást engedélyező rés, akkor nem kapunk hibajelzést, csak a lemezegység lámpája jelzi a hibát. A törlés és a formázás természetesen nem hajtódik végre.

HELP

Rövidítés: heL Token: $EA(234)

Mód: mind parancs, mind program módban használható.

Hiba után kiírja a hibát okozó sort.

Szintaxis: **HELP**

A hibát okozó sornak azt a része, amit az interpreter nem tudott feldolgozni inverz vagy aláhúzott alakban jelenik meg. A <HELP> billentyű megnyomása ekvivalens a HELP parancs kiadásával.

Hibalehetőségek: Nincs.

HEX$

Rövidítés: hE Token: $D2(210)

Mód: mind parancs, mind program módban használható.

Sztring függvény, amelyik az argumentumaként megadott szám hexadecimális alakjával tér vissza.

Szintaxis: **HEX$(**<arit.kif.>**)**

Az aritmetikai kifejezésnek a 0<=N<=65535 intervallumba kell esnie.

Példák:

```
  (i)     PRINT HEX$(32) : REM = "20"
          PRINT HEX$(45) : REM = "2C"
 (ii)     ? HEX$(DEC("FFFF")) : REM = "FFFF"
```

Az (i) alatti példák HEX$ közvetlen hatását mutatják. (ii) azt példázza, hogy HEX$ a DEC inverze. Ennek megfelelően bármilyen hexadecimális szám is áll a "FFFF" helyén, a (ii) példa mindig a beírt számot mutatja.

Hibalehetőségek: Az aritmetikai kifejezés kiértékelése közben számos hiba adódhat. Ha értéke nem esik a jelzett intervallumba, akkor ?ILLEGAL QUANTITY ERROR hibajelzést kapunk.

IF...THEN...ELSE C 64-es mód

Rövidítés: nincs Token: IF $8B(139)
THEN $A7(167)
ELSE $D5(213)

Mód: mind parancs, mind program módban használható.

Feltételes vezérlésátadás. Ha az IF kulcsszót követő feltétel

teljesül, akkor a feltételt követő utasítás kerül végrehajtásra. Ha nem, akkor az interpreter ellenőrzi, hogy van-e ELSE része az utasításnak. Ha nincs, akkor az IF utasítást követő első utasítás kerül végrehajtásra. Ha van, akkor az ELSE-t követő rész. Az interpreter figyelembe veszi az esetleg használt utasítászárójeleket (BEGIN, BEND). C-64-es üzemmód esetén az ELSE nem használható. (Lásd C-64 73. oldal!)

Szintaxis:

IF <arit. kifejezés> { **THEN** { <sorszám> / <utasítás> } / **GOTO** <sorszám> } {:**ELSE** <utasítás>}

<sorszám> csak előjel nélküli konstans lehet!

Példák:

```
(i)   FOR I=1 TO 1000: X=RND(1): GOSUB 1000:IF T$="*" THEN NEXT
(ii)  100 IF P=72 THEN P=0: GOSUB 1200
(iii) 200 IF X=1 THEN IF A=0 AND B=0 THEN GOSUB 300
(iv)  600 IF (X<0) AND (X>0) THEN IDE NEM KERUL A VEZERLES!!!
```

Az első példánk az 1000-rel kezdődő alprogramot ellenőrzi. Az X=RND(1) utasítás X értékét véletlenszerűen állítja be. Az alprogram használja az X értékét és visszatéréskor beállítja a T$ értékét. A ciklus abban az esetben jár le teljes egészében, ha T$="*" minden szubrutinhívás esetén teljesül. Példánk mutatja, hogyan lehet az IF utasítást parancs módban is használni.

A (ii) példa ellenőrzi, hogy P értéke elér-e egy bizonyos határt (72), s ha igen, akkor ezt 0-ra változtatja, és meghívja az 1200 alatt kezdődő alprogramot.

Következő példánk azt mutatja, hogy az IF utasítások egymásba ágyazhatók.

Az utolsó programsor egy hibás program része, a feltétel sohasem lesz igaz, ezért a vezérlés a THEN utáni - szintaktikusan helytelen - részre sohasem kerül.

Hibalehetőségek: A kifejezés kiértékelése közben számos hibajelzést kaphatunk. A GO TO alak nem megengedett;

```
IF X<>0 GO TO 10
```

tehát szintaktikus hibát eredményez. Ha a GOTO egy nem létező sorra mutat, akkor ?UNDEF'D STATEMENT ERROR hibajelzést kapunk. Ha aritmetikai kifejezést használunk a feltétel helyén, akkor csak a 0 számít hamisnak. IF X<>0 THEN... és IF X THEN... tehát ekvivalensek.

INPUT C-64-es mód

Rövidítés: nincs Token: $85 (133)

Mód: csak program módban használható.

A billentyűzetről adatokat olvas be. (Lásd C-64 74. oldal!)

Szintaxis: **INPUT** {"<szöveg>";} <változólista>

INPUT# C-64-es mód

Rövidítés: in Token: $84 (132)

Mód: mind parancs, mind program módban használható.

A megadott logikai perifériáról adatokat olvas be. (Lásd C-64 77. oldal!)

Szintaxis: **INPUT#**<log.fileszám>, <változólista>

INSTR

Rövidítés: inS Token: $D4(212)

Mód: mind parancs, mind program módban használható.

Két-, vagy háromváltozós függvény, amelyik megadja, hogy az első paraméterként szereplő sztringben hol kezdődik a második sztring. Ha egyáltalán nem szerepel, akkor a függvény értéke 0 lesz. Az összehasonlítás a harmadik paraméterként megadott pozíciótól kezdődik.

Szintaxis:
INSTR(<sztring.1>,<sztring.2> {,<arit.kif.>})

Példák:

```
  (i)     PRINT INSTR("KUTYA","U"): REM =2
          PRINT INSTR("KUTYA","J"): REM =0
( ii)     PRINT INSTR("KUTYA","U",3): REM =0

(iii)     100 INPUT A$,B$,C$
          110 N=INSTR(A$,B$)
          120 IF N=0 THEN GOTO 100
          130 L1=LEN(B$)
          140 X$=LEFT$(A$,N-1)+C$+RIGHT$(A$,N+L1)
          150 PRINT X$
          160 GOTO 100
```

Az (i) példa az INSTR hatását mutatja be. A (ii) példában használjuk az összehasonlítás kezdőpozíciójának beállítási lehetőségét. Ekkor a "KUTYA" sztringben az "U" már nem szerepel!

A (iii) alatti kis program az A$ sztringben a B$ szövegrészt a C$ szöveggel cseréli fel (feltéve, hogy megtalálja).

Hibalehetőségek: A kifejezések kiértékelése közben számos hiba adódhat. Maga az INSTR hiba nélkül hajtódik végre.

INT

C-64-es mód

Rövidítés: nincs Token: $B5 (181)

Mód: mind parancs, mind program módban használható.

Függvény: az argumentum egész értékét adja. (Lásd C-64 79. oldal!)

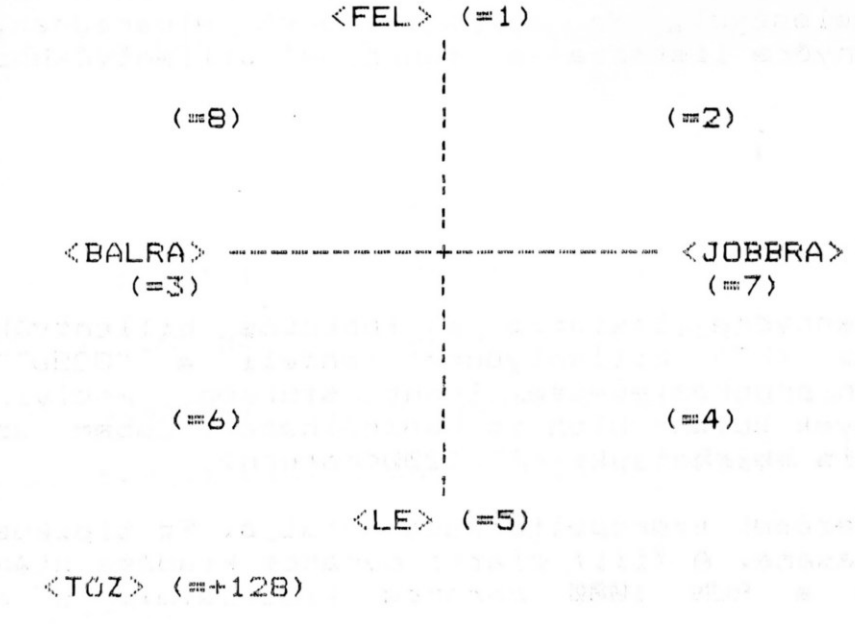
Sintaxis: **INT(<arit.kif.>)**

JOY

Rövidítés: j0 Token: $CF(207)

Mód: mind parancs, mind program módban használható.

Az argumentumként megadott botkormány (joystick) állapotával tér vissza. Az egyes értékek jelentése a következő:



Szintaxis: **JOY(<arit.kif.>)**

Példák:

(i)
```
10 PRINT JOY(1): GOTO 10
```

(ii)
```
10 DATA "ESZAK",ESZAKKELET","KELET","DELKELET"
20 DATA "DEL","DELNYUGAT","NYUGAT","ESZAKNYUGAT"
30 FOR I=1 TO 8: READ IR$(I): NEXT I
40 DO
50 X=JOY(0): T=X AND 128: IR=X AND 127
60 IF IR<>0 THEN PRINT IR$(IR)
70 IF T<>0 THEN PRINT "TUZ!!!!": ELSE PRINT
80 LOOP
```

Az (i) program folyamatosan a képernyőre írja az első botkormány állapotát. A (ii) program ugyanazt a feladatot végzi el, csak szöveggel írja ki a botkormány állapotát.

Hibalehetőségek: Az aritmetikai kifejezés kiértékelése közben számos hiba adódhat. Ha értéke nem 1, vagy 2, akkor ?ILLEGAL QUANTITY ERROR hibajelzést kapunk.

KEY

Rövidítés: kE Token: $F9(249)

Mód: mind parancs, mind program módban használható.

Segítségével megváltoztathatjuk, illetve megnézhetjük a funkció billentyűkhöz rendelt szövegrészeket.

Szintaxis:
KEY {<sorszám>,<sztring>}

A <sorszám> a funkciós billentyű számát, a sztring kifejezés a hozzárendelt sztringet adja meg. <sorszám> értéke így csak 1-8 lehet. A parancs kiadása után a megfelelő funkció billentyű lenyomása ekvivalens a <sztring> begépelésével. Ha a paraméterek elmaradnak, akkor az interpreter a képernyőre listázza a funkciós billentyűkhöz rendelt sztringeket.

Példák:

```
  (i)     KEY
 (ii)     KEY 3,"GOSUB"
(iii)     KEY 2,"RUN1000"+CHR$(13)
```

Az (i) alatti példa a képernyőre listázza a funkciós billentyűk jelentését. A (ii) példa az <F3> billentyűhöz rendeli a "GOSUB" karaktersorozatot. Erre olyan program esetén lehet szükség, amelyik sok önálló részből áll, amelyek külön-külön is használhatók. Ebben az esetben a GOSUB 1200-at így is beírhatjuk: <F3>1200<return>

A (iii) példa a sztringkifejezések szerepeltetését mutatja. Ez tipikus a <RETURN> billentyű hozzáírására. A (iii) alatti parancs kiadása után az <F2> lenyomása ekvivalens a RUN 1000 parancs kiadásával, s a program azonnal el is indul!

Hibalehetőségek: A kifejezések kiértékelése közben számos hiba történhet. Ha az aritmetikai kifejezés értéke nem esik az 1-8 intervallumba, akkor ?ILLEGAL QUANTITY ERROR hibajelzést kapunk.

LEFT$ C-64-es mód

Rövidítés: leF Token: $CB (200)

Mód: mind parancs, mind program módban használható.

Stringfüggvény: a sztring argumentumának kezdőszeletével tér vissza. (Lásd C-64 79. oldal!)

Szintaxis: **LEFT$**(<sztring kif.>,<arit.kif>)

LEN C-64-es mód

Rövidítés: nincs Token: $C3 (195)

Mód: mind parancs, mind program módban használható.

Függvény: az argumentum string hosszát adja. (Lásd C-64 80. oldal!)

Szintaxis: LEN(<arit.kif>)

LET C-64-es mód

Rövidítés: lE Token: $88 (136)

Mód: mind parancs, mind program módban használható.

értékadó utasítás. (Lásd C-64 81. oldal!)

Szintaxis:LET <változó> = <kifejezés>

LIST C-64-es mód

Rövidítés: lI Token: $C3 (195)

Mód: mind parancs, mind program módban használható.

Kilistázza a program adott sorait. (Lásd C-64 82. oldal!)

Szintaxis: LIST {<sorszám>}{-{<sorszám>}}

LOAD C-64-es mód

Rövidítés: lO Token: $93(147)

Mód: mind parancs, mind program módban használható.

Adott perifériáról betölt egy PRG file-t. (Lásd C-64 83. oldal!)

Szintaxis: LOAD {<filenév>}{,<egység}{,<mód>}

LOCATE

Rövidítés: loC Token: $E6(230)

Mód: mind parancs, mind program módban használható.

A grafikus kurzor helyzetét állíthatjuk be. A nagyfelbontású képernyő tartalma nem változik meg. A grafikus kurzor pillanatnyi helyzetét az RDOT(0),RDOT(1) koordinátapár adja meg.

Szintaxis: **LOCATE** <x.koord>,<y.koord>

Az első aritmetikai kifejezés az x, a második az y koordináta értékét határozza meg.

Példák:
```
  (i)    LOCATE 160,100
 (ii)    LOCATE 80,100
```

Az (i) példa a grafikus kurzort a normál, nagyfelbontású képernyő közepére állítja. A (ii) ugyanezt teszi többszínű, nagyfelbontású üzemmód esetén.

Hibalehetőségek: Az aritmetikai kifejezések kiértékelése közben számos hiba adódhat. Ha a koordináta értékek egy nem létező pontra mutatnak, akkor ?ILLEGAL QUANTITY ERROR hibajelzést kapunk.

LOG C-64-es mód

Rövidítés: nincs Token: $BC (188)

Mód: mind parancs, mind program módban használható.

Függvény: az argumentum e-alapú logaritmusát adja. (Lásd C-64 85. oldal!)

Szintaxis: **LOG(** <arit.kif> **)**

LOOP

Rövidítés: loO Token: $EC(236)

Mód: mind parancs, mind program módban használható.

A DO ciklusok záró utasítása. A LOOP-ot követő kilépési feltétel értékétől függően vagy a LOOP-ot követő első utasítás, vagy a LOOP-hoz tartozó DO utasítás hajtódik végre.

Szintaxis:

LOOP { **UNTIL** / **WHILE** } <logikai kif.>

Ha a LOOP utasítás után nem áll sem UNTIL, sem WHILE, akkor a vezérlés mindig visszakerül a megfelelő DO utasításra. Az UNTIL esetében a logikai kifejezésnek igaznak, a WHILE esetében pedig hamisnak kell lennie ahhoz, hogy a ciklus lejárjon. Ellenkező esetben a vezérlés mindig a megfelelő DO utasításra tér vissza.

Példák:

```
  (i)    DO: PRINT "I LOVE COMMODORE!": LOOP
 (ii)    10 INPUT A%,B%
         20 DO
         30 R%=A%
         40 A%=B%-INT(B%/A%)*A%: B%=R%
         50 LOOP WHILE A%<>0
         60 PRINT "LNKO=";R%
(iii)    1200 DO
         1210 LOOP WHILE (JOY(1) AND128)=0
```

(i) a LOOP parancs módban való használatát mutatja. Az interpreter egy végtelen ciklusban folyamatosan újra és újra a képernyőre írja az "I LOVE COMMODORE" szöveget.

A (ii) program az A% és B% egész számok legnagyobb közös osztóját számítja ki az euklideszi algoritmus segítségével. A kilépés feltétele, hogy a maradék 0 legyen.

A (iii) példában addig 'jár' a ciklus, mig az 1-es botkormány <TŰZ> gombját meg nem nyomjuk.

Hibalehetőségek: A logikai kifejezés kiértékelése közben számos hiba adódhat. Ha az interpreter nem találja meg a LOOP-hoz tartozó DO-t, akkor ?LOOP WITHOUT DO ERROR hibaüzenetet kapunk. Ennek oka lehet egy nem megfelelő helyen végrehajtott LOOP, NEXT vagy RETURN utasítás is.

MID$ C-64-es mód

Rövidítés: mI Token: $CA (202)

Mód: mind parancs, mind program módban használható.

Stringfüggvény: egy sztring adott darabját adja vissza. (Lásd C-64 86. oldal!)

Szintaxis: **MID$**(<sztring kif.>,<kezdő poz.>[,<hossz>])

MONITOR

Rövidítés: mO Token: $FA(250)

Mód: csak parancs módban használható.

Belépés a gépi kódu monitorba.

Szintaxis: **MONITOR**

A parancs kiadásával a C-128 beépített gépi kódú monitorába lépünk be. A BASIC utasításokat a kilépésig nem használhatjuk. A kilépésre az **X** monitor parancs szolgál. A gépi kódú monitorról részletesen a 7. fejezetben szólunk.

MOVSPR

Rövidítés: mO Token: $FE $06 (254,6)

Mód: mind parancs, mind program módban használható.

Sprite-ok elhelyezésére és mozgatására szolgál.

Szintaxis:

MOVSPR <sp.szám>,<x.koord>,<y.koord>

MOVSPR <sp.szám>,{+ _}<x.koord>,{+ _}<y.koord>

MOVSPR <sp.szám>,<szög>#<seb.>

Az egyes szintaktikailag eltérő alakok az utasítás más és más formáját mutatják.Az első alak az <sp.szám> számú sprite-ot a képernyő megadott részére helyezi. Az <sp.szám> értéke így 1-8 lehet csak.

A második alak relatív pozícionálást biztosít. A két (esetleg előjeles) koordináta a sprite új helyzetét a jelenlegihez képest mutatja.

Az utasítás első két fajtája statikus. A harmadik a sprite-ok mozgatására szolgál. A mozgatást a megszakító rendszer végzi. A sprite jelenlegi helyzetéből elindul. Sebességét a <seb.> paraméter adja meg. értékének a 0-15 intervallumba kell esnie. Minél nagyobb az érték, annál nagyobb sebességgel mozog a sprite. <szög> a mozgatás irányát adja meg.

Példák:

```
  (i)     MOVSPR 1,100,130
          MOVSPR 1,-50,-40
 (ii)     X=50: Y=40: MOVSPR 1,-X,-Y
          MOVSPR 1,+X,+Y
(iii)     MOVSPR 1,90#3
```

Az (i) példa először a 100,130 koordinátájú pontra helyezi az 1-es sprite-ot, majd ehhez képest -50,-40-nel elmozdítja. Ennek hatására a sprite az X=100-50=50, illetve Y=130-40=90 koordinátájú pontban jelenik meg. Nem 'mozog' oda, egyszerűen eltűnik a régi helyéről, s az új helyen jelenik meg. A (ii) példa az (i) folytatásaként azt mutatja, hogy a +/- hatása kifejezések használatával is meg van. Nem az számít, hogy a kifejezés értéke pozitív vagy negatív, hanem, hogy az utasítás alakjában szerepel-e az előjel!

A (iii) példa az előző sprite-ot vizszintesen mozgatja keresztül a képernyőn meglehetősen lassan.

Megjegyezzük, hogy a fenti parancsok kiadása után lehet, hogy semmi sem lesz látható, ehhez a sprite többi paraméterét is be kell állítani. Erről lásd a 6.1 paragrafust.

Hibalehetőségek: Az aritmetikai kifejezéseknek a megfelelő értéktartományba kell esniük. Ha nem, akkor ?ILLEGAL QUANTITY ERROR hibajelzést kapun k.

NEW C-64-es mód

Rövidítés: nincs Token: $A2(162)

Mód: mind parancs, mind program módban használható.

Törli a BASIC programot és az összes változót. (Lásd C-64 87. oldal!)

Szintaxis: **NEW**

NEXT C-64-es mód

Rövidítés: nE Token: $82(130)

Mód: mind parancs, mind program módban használható.

Ciklusmag záró utasítása. (Lásd C-64 88. oldal!)

Szintaxis: **NEXT** <változó lista>

NOT C-64-es mód

Rövidítés: nO Token: $A8 (168)

Mód: mind parancs, mind program módban használható.

Aritmetikai kifejezés 'logikai tagadása'. (Lásd C-64 89. oldal!)

Szintaxis: **NOT** <arit.kif.>

ON C-64-es mód

Rövidítés: nincs Token: $91 (145)

Mód: mind parancs, mind program módban használható.

Többirányú feltételes programelágazást teszlehetővé. (Lásd C-64 90. oldal!)

Szintaxis: **ON** <feltétel> { **GOSUB** / **GOTO** } <sorszám lista>

OPEN

C-64-es mód

Rövidítés: oP Token: $9F (159)

Mód: mind parancs, mind program módban használható.

Adott logikai fileszámhoz tartozó file megnyitása. (Lásd C-64 91. oldal!)

Szintaxis: **OPEN** <log.fileszám> {,<egység>}{,<mód>}{,<filenév>}

OR

C-64-es mód

Rövidítés: nincs Token: $B0 (176)

Mód: mind parancs, mind program módban használható.

Két aritmetikai kifejezés 'logikai vagy'-át számítja ki. (Lásd C-64 93. oldal!)

Szintaxis: <arit.kif.> **OR** <arit.kif.>

PAINT

Rövidítés: pA Token: $DF(223)

Mód: mind parancs, mind program módban használható.

Lehetőséget nyújt arra, hogy a 40 oszlopos nagyfelbontású képernyő valamely zárt, folytonos vonallal körülhatárolt részét **befessük**, **besatírozzuk**. Ehhez az utasításban a festés színtípusán kívül a befestendő terület egy pontját kell megadnunk. A festés a nagyfelbontású képernyő tartalmától függ.

Szintaxis: **PAINT** {<színtípus>}{, {<x>,<y>}{,<mód>}}

A besatírozott pontok színe a <színtípusnak> megfelelő szín lesz. Ha elhagyjuk, akkor az 1-es színnel (az írás színe) dolgozik a gép. <x>,<y> a befestendő terület egy pontja, ha elmarad, akkor a grafikus kurzor aktuális helyén kezdődik a festés. A <mód> paraméter a határvonal értelmezését adja meg. Ha értéke 0, akkor a festés a bekapcsolt pontokig tart, ha pedig 1, akkor a háttérszínű pontokig. Mind a négy paraméter tetszőleges aritmetikai kifejezés lehet.

Példák:

(i)
```
10 GRAPHIC 1,1
20 CIRCLE ,160,100,60,60
30 PAINT ,160,100
```

(ii)
```
10 GRAPHIC 1,1
20 FOR I=0 TO 11
30 CIRCLE 1,159,100,20,80,,,15*I
40 NEXT I
60 FOR I=1 TO 6
65 FOR J=0 TO 1
70 PAINT J,0,0,J
80 NEXT J
90 NEXT I
100 GETKEY A$
110 GRAPHIC 0
```

Az (i) példa a PAINT egyszerű hatását mutatja be. A (ii) példában felváltva festünk az 1-es és a 0-ás színnel. Ennek hatására a képernyőn levő ábra végül eltűnik.

Hibalehetőségek Az aritmetikai kifejezések kiértékelése közben számos hiba adódhat. A <színtípus> értékének összhangban kell lennie a választott kijelzési móddal. Ha nem, akkor ?ILLEGAL QUANTITY ERROR hibajelzést kapunk.

PEEK

C-64-es mód

Rövidítés: peE Token: $C2 (194)

Mód: mind parancs, mind program módban használható.

Függvény: a memória (BANK által kiválasztott) címét olvassa. (Lásd C-64 94. oldal!)

Szintaxis: **PEEK**(<arit.kif.>)

PEN

Röidítés: pE Token: $CE $04 (206,4)

Mód: mind parancs, mind program módban használható.

A fényceruza helyzetéről ad felvilágosítást.

Szintaxis: **PEN**(<arit.kif.>)

Az argumentum értékének a 0-4 intervallumba kell esnie. Az egyes értékek jelentése a következő:

0 a fényceruza X-koordinátája a nagyfelbontású képernyőn (60-320)
1 a fényceruza Y-koordinátája a nagyfelbontású képernyőn

(50-200)
2 a fényceruza oszlopa a 80 karakteres képernyőn
3 a fényceruza sora a 80 karakteres képernyőn
4 aktív volt-e a fényceruza

A függvény 0-3 argumentum esetén a jelzett koordinátaértékekkel tér vissza. Ha az argumentum 4, akkor a függvény 1-el válaszol, ha a legutolsó hívása óta használtuk a fényceruzát, különben 0. Ez a funkció nem használható a COLLISION-nel együtt.

Példák:

```
(i)     1000 DO UNTIL PEN(4): LOOP
        1010 PRINT PEN(2);PEN(3)
```

A 1000 sorban addig vár a program, míg a fényceruzát a képernyő felé nem fordítjuk, s utána kiírja a 80 oszlopos képernyőn hová helyeztük a ceruzát.

Hibalehetőségek: Ha az argumentum értéke nem a megadott, ?ILLEGAL QUANTITY ERROR hibajelzést kapunk.

PLAY

Rövidítés: pL Token: $FE $04 (254,4)

Mód: mind parancs, mind program módban használható.

Dallamok játszását teszi lehetővé.

Szintaxis: **PLAY** <sztring>

A megszakító rendszer a <sztring> karaktereinek megfelelő dallamot eljátsza.

A sztringben szereplő egyes karakterek jelentése a következő:

```
On  a 7 lehetséges oktáv kiválasztása, n=0,...,6
Tn  a 10 lehetséges hullámforma kiválasztása, n=0,...,9
      (ezeket definiálhatjuk az ENVELOPE utasítással)
Un  a hangerő állítása, n=0,...,9
Vn  a három generátor közül melyik szólaltassa meg
      a hangot, n=0,...,2
Xn  a FILTER-ben megadott szűrés bekapcsolása
      n=1 bekapcsolás, n=0 kikapcsolás
N     a játszható hangok egyike: A,B,C,D,E,F,G
#N  fél hanggal magasabb hang
$N  fél hanggal alacsonyabb hang
W   egész hang
H   fél hang
Q   negyed hang
I   nyolcad hang
S   tizenhatod hang
.N  pontozott ütem
R   szünet
M   megállítja a játszást
```

Példák:
(i)
```
10 a$="cdefga#ac"
20 x$=""
30 for i=50 to 52
40 x$=x$+"o"+chr$(i)+a$:next i
50 x$=x$+"rrr"
60 vol 15
70 tempo 40
80 for i=48 to 57
90 play "t"+chr$(i)+x$
100 next i
```

Az (i) alatti példa skálázik a C-128-on. A 10. sorban beállítjuk egy oktáv lejátszásához szükséges a$ sztringet. A 30-40 ciklusban ezt négy oktávra bővítjük, eléírva az oktávváltáshoz szükséges karaktereket is. Az 50. sorban egy rövid szünetet adunk hozzá az x$-hoz. Végül a 80-100 ciklusban mind a 10 lehetséges hullámformával kipróbáljuk.

Hibalehetőségek: Ha a sztringben a fenti karakterektől eltérőt talál, akkor ?ILLEGAL QUANTITY ERROR hibajelzést kapunk.

POINTER

Rövidítés: poI Token: $CE $0A (206,10)

Mód: mind parancs, mind programmódban használható.

Függvény, ami az argumentumaként megadott változó kezdőértékével tér vissza. Ez a cím mindig az 1. szeletre utal.

Szintaxis: **POINTER**(<változó>)

Példák:
(i)
```
A%=100
AP%=POINTER(A%)
BANK 1
? 256*PEEK(AP%)+PEEK(AP%+1)
```

A példában az A% változónak 100-at adunk értékül. Ennek hatására az 1.szeleten az interpreter tárolja a változót is, és az értékét is. Hogy hogyan, arról lásd a 3.2 paragrafust. A POINTER függvény segítségével megkereshetjük a változót, és magunk is kiszámíthatjuk az értékét.

A függvény elsősorban a gépi kódú alprogramok írására hasznos.

Hibalehetőségek: Ha a változó nem létezik, a függvény 0-val tér vissza.

POT

Rövidítés: pO Toke: $CE $02 (206,2)

Mód: mind parancs, mind program módban használható.

A függvény a négy csatlakoztatható potméter értékével tér vissza.

Szintaxis: POT(<sorszám>)

<sorszám> a potméter sorszáma, ez 1,2,3 vagy 4 lehet. A függvény értéke a 0-255 intervallumba esik. Ha valamelyik potméteren benyomjuk a 'TÜZ' gombot is, a függvény értéke 256-tal megnő.

Példák:
```
  (i)     IF POT(1)>255 THEN PRINT "TUZZZ!!!!"
 (ii)     100 DO UNTIL POT(1)<=255: LOOP
```

Ha az 1-es potméter 'TŰZ' gombját benyomtuk, akkor (i) kiírja a képernyőre, hogy "TUZZZ!!!!". A (ii) példában a program addig vár, míg el nem engedjük az 1-es potméter 'TŰZ' gombját.

Hibalehetőségek: Ha a POT argumentuma nem esik a megfelelő intervallumba, ?ILLEGAL QUANTITY ERROR hibajelzést kapunk.

POKE C-64-es mód

Rövidítés: poK Token: $97 (151)

Mód: mind parancs, mind program módban használható.

A memória (BANK által kiválasztott) címének tartalmát írja felül. (Lásd C-64 95. oldal!)

Szintaxis: POKE <cím>, <érték>

POS C-64-es mód

Rövidítés: nincs Token: $B9 (185)

Mód: mind parancs, mind program módban használható.

Függvény: kiszámítja a kurzor helyét a logikai sorban. (Lásd C-64 96. oldal!)

Szintaxis: POS(<változó>)

PRINT C-64-es mód

Rövidítés: ? Token: $99 (153)

Mód: mind parancs, mind program módban használható.

A megadott értékeket nyomtatja ki az elsődleges outputra, ami általában a képernyő. A kiírás vezérlésére a ,; valamint a SPC(, TAB(függvények szolgálnak. (Lásd C-64 97. oldal!)

Szintaxis: **PRINT** {<nyomtatási kép>}

PRINT# C-64-es mód

Rövidítés: pR Token: $98 (152)

Mód: mind parancs, mind program módban használható.

A megadott értékeket nyomtatja ki az adott logikai file-ba. A kiírás vezérlésére a ,; valamint a SPC(, TAB(függvények szolgálnak. (Lásd C-64 99. oldal!)

Szintaxis: **PRINT#**<log.fileszám> {,<nyomtatási kép>}

PRINT USING

Rövidítés: ?usI Token: USING $F3(251)

Mód: mind parancs, mind program módban használható.

Lehetővé teszi számok és sztringek formázott kiírását. A formátumot megadó sztringben a következő - a kiírás formátumát vezérlő - karakterek szerepelhetnek:

Karakter	Jelentés
#	értékes számjegy vagy karakter
+	a pozitív előjel kiíratása
-	szám előjelének helye
.	tizedespont helye
,	elválasztó jel
$	dollárjel helye
↑↑↑↑	exponens helye
=	középre igazítás
>	jobbra igazítás

Szintaxis: **PRINT**{#<lf>} **USING** <formátum>;<kifejezés lista>{;}

<lf> az a logikai fileszám, ahová a formázott kiírás történik. Ha elmarad, akkor az elsődleges outputra ír a rendszer. A <formátum> tetszőleges sztringkifejezés lehet. Ennek a karakterei határozzák meg a kifejezések kiíratásának formáját. A <kifejezés lista> tetszőleges kifejezések **vesszővel elválasztott** sorozata.

A C-128 interpreter párhuzamosan dolgozza fel a <formátum> sztringet és a <kifejezés listát>. Ha a kettőt nem találja összeegyeztethetőnek, akkor a feldolgozást abbahagyja, és ?SYNTAX ERROR hibaüzenettel abbamarad a program futása. Még ha a kiíratási formátum értelmezhető is, előfordulhat, hogy az előírásnak megfelelően nem jeleníthető meg a szóban forgó mennyiség. Sztringek esetén ilyenkor a sztring néhány utolsó karaktere elvész. Számok esetén az értékes jegyek és a tizedespont helyen *-ok nyomtatódnak ki.

Ha a <formátum> előbb merül ki, mint a <kifejezés lista>, akkor az interpreter újra kezdi a <formátum> feldolgozását.

Ha az utasítás végére nem írunk pontos vesszőt (;), az utasítás még egy CHR$(13) karaktert is kiír, s így a következő kiíró utasítás új sorban kezdődik.

Példák:

```
  (i)     100 DEF FN G(X)=4*X↑2-6*X+2
          110 FOR I=1 TO 60
          120 PRINT USING "    ####.##";FN G(I)
          130 NEXT I
 (ii)     120 PRINT USING "    ####.##↑↑↑↑"; FN G(X)
          120 PRINT USING "    #.#####↑↑↑↑"; FN G(X)

(iii) 1000 A$="Commodore 128"
      1010 PRINT USING "########################";A$
      1020 PRINT USING "#######################>";A$
      1030 PRINT USING "#######################=";A$
```

(i)-ben a 100. sorban definiált G(X) függvény helyettesítési értékeit listázzuk ki. Az utolsó számok helyett már csillagok jelennek meg, mert a kiírandó szám a kívánt módon már nem írható ki (túl nagy). (ii)-ben szereplő két 120-as sor az első példa azonos sorának a módosítása. Az exponenciális kijelzés miatt itt már nem történik túlcsordulás. A második fajta kijelzés felel meg a zsebszámológépeken gyakran szereplő lebegőpontos kijelzési módnak.

A (iii) példa a sztring kifejezések kiíratásánál használható =, illetve > karakterek szerepét mutatja. Normál esetben a sztring balra igazítva íródik ki. Az '=' használata esetén középre, a '>' használata esetén pedig jobbra igazítva. Ha a kiírandó sztring hosszabb, mint a mezőszélesség, akkor a sztring utolsó karakterei elvesznek. A mezőszélességbe a # jeleken kívül az = és a > jel is beleszámít.

Hibalehetőségek: A kifejezések kiértékelése közben számos hiba adódhat. A <formátum> sztring karaktereinek típusban és alakban meg kell felelnie a kifejezések típusának. Ha ez nem teljesül, akkor ?SYNTAX ERROR hibajelzést kapunk.

PUDEF

Rövidítés: pU Token: $DC(221)

Mód: mind parancs, mind program módban használható.

Lehetőséget ad a PRINT USING-ban használt egyes jelek átdefiniálására.

Szintaxis: PUDEF <sztring kifejezés>

A <sztring kifejezés> értékeként kapott sztring maximum négy karakterből áll. Ezek a PRINT USING-ban használt következő karaktereknek felel meg:

Sorszám	Jelentés
1	elválasztó jel
2	vessző
3	tizedespont
4	$ jel

Példák:
(i) PUDEF " ;,"
(ii) PUDEF "*,.@"

Az (i) parancs kiadása után a PRINT USING tizedespont helyett tizedesvesszőt nyomtat. Ennek megfelelően a ","-t valami másra, például ";"-re kell cserélni. A (ii) példában az elválasztó jelet - ami normál körülmények között szóköz, - a "*" karakterre cseréljük, míg a "$" jelet "@"-ra. Ha lenne Ft karakterünk, akkor arra is cserélhetnénk.

Hibalehetőségek: A kifejezés kiértékelése után egy legfeljebb 4 hosszúságú sztringet kell kapnunk. Ha nem, akkor az interpreter ?ILLEGAL QUANTITY ERROR hibajelzést küld.

RCLR

Rövidítés: rC Token: $CD(205)

Mód: mind parancs, mind program módban használható.

Az aritmetikai függvény az i. színtípushoz rendelt szín kódját adja vissza. Az i lehetséges értékei:

i értéke	Színtípus
0	háttér színe (40 oszlop)
1	írás színe (40 oszlop)
2	több szín#1
3	több szín#2
4	keret színe (40 oszlop)
5	írás színe (80 oszlop)
6	háttér színe (80 oszlop)

Szintaxis: RCLR(arit.kif.)

Az argumentum értékének a 0-6 intervallumba kell esnie.

Példák:

```
  (i)     PRINT RCLR(4): REM = 14
 (ii)     1000 DATA FEKETE,FEHER,PIROS,ENCIAN
          1010 DATA BIBOR,ZOLD,KEK,SARGA
          1020 DATA NARANCSSARGA,BARNA,KEKESZOLD,ROZSASZIN
          1030 DATA KEKESZOLD,VILAGOSKEK,SOTETKEK,VILAGOSZOLD
          1040 RESTORE 1000
          1050 FOR I=0 TO 15: READ SZ$(I): NEXT I
          1060 PRINT "<CLEAR>"
          1070 PRINT "HATTER SZINE:";SZ$(RCLR(0))
          1080 PRINT "KURZOR SZINE:";SZ$(RCLR(1))
          1090 PRINT "KERET SZINE: ";SZ$(RCLR(4))
          1100 PRINT "TOBBSZIN#1:  ";SZ$(RCLR(2))
          1110 PRINT "TOBBSZIN#2:  ";SZ$(RCLR(3))
          1120 GETKEY A$: RETURN
```

Az (i) példa a képernyőre írja a keret színének kódját. A (ii) alatt megírt alprogram kiírja a 40 oszlopos képernyő valamennyi színtípusának színét.

Hibalehetőségek: Az aritmetikai kifejezések kiértékelése közben számos hiba adódhat. Ha értéke nem esik a 0-6 intervallumba, akkor ?ILLEGAL QUANTITY ERROR hibajelzést kapunk.

RDOT

Rövidítés:rD Token:$D0(208)

Mód: mind parancs, mind program módban használható.

Az argumentum értékétől függően a grafikus kurzor színtípusával, illetve x és y koordinátájával tér vissza:

érték	Jelentés
0	a grafikus kurzor x koordinátája
1	a grafikus kurzor y koordinátája
2	a grafikus kurzor színtípusa

Szintaxis: **RDOT(**<arit.kif.>**)**

Az argumentum értékének a 0-2 intervallumba kell esnie.

Példák:

```
  (i)     10 GRAPHIC 1,1
          20 LOCATE 160,100
          30 PRINT RDOT(0),RDOT(1): REM =160,100
          40 PRINT RDOT(2): REM = 0 A HATTER SZINTIPUSA
```

A fenti egyszerű program az RDOT hatását mutatja be. A LOCATE utasítás a kurzort a (160,100) pontra állítja, ennek megfelelően RDOT(0),RDOT(1) értéke ugyanez. A nagyfelbontású képernyőt a 10. sorban töröltük, és azóta semelyik pontját sem kapcsoltuk be. Ennek megfelelően minden pont színe háttérszínű (=0).

Hibalehetőségek: A kifejezés kiértékelése közben számos hiba adódhat. Ha értéke nem esik a 0-2 intervallumba, akkor ?ILLEGAL QUANTITY ERROR hibajelzést kapunk.

READ

C-64-es mód

Rövidítés: rE Token: $87 (135)

Mód: mind parancs, mind program módban használható.

A DATA sorokban megadott konstansokat olvassa be. (Lásd C-64 100. oldal!)

Szintaxis: **READ** <változó lista>

RECORD#

Rövdítés: rE Token: $FE $12 (254,18)

Mód: mind parancs, mind program módban használható.

A DOPEN vagy OPEN utasítással megnyitott REL típusú file rekordszámát állítja be.

Szintaxis: **RECORD#**<lfszám>,<rek.szám> {,<byte>}

Az <lfszám> alatt nyilvántartásba vett REL típusú megnyitott file író/olvasó fejét a <rek.szám> sorszámú rekord <byte>-adik karakterhelyére állítja. A következő PRINT#, GET# vagy INPUT# utasítás innen fog kezdődni. Ha a <byte> elmarad, az 1-nek számít.

Példák:

(i)
```
1200 RECORD#2,1000
1210 IF DS<>50 THEN DCLOSE#2: END
```

A példában a rekordszámot az 1000. rekordra állítottuk. Ha az 50 számú hibát kaptuk az nem baj, mert ez azt jelenti, hogy egy új rekordot írtunk a file-hoz.

RENAME

Rövidítés: reN Token:$F5(245)

Mód: mind parancs -, mind program módban használható.

Lemezes file-ok átnevezésére használhatjuk. Paramétereiben a file régi és új nevét, továbbá a lemez meghajtójának adatait kell megadni.

Szintaxis:
RENAME <régi név> **TO** <új név> {,**D**<meghajtó>} {,**U**<egység>}

A <meghajtó> értékének a 0-1, az <egység> értéke a 8-11 intervallumba kell esnie. Ha a paramétereket nem adjuk meg, akkor az utasítás a 8-as lemezegység 0-ás meghajtójában levő lemezre vonatkozik. Ha a <régi név>, illetve az <új név> paraméterek nem sztring konstansok, akkor kerek zárójelek közé kell tenni őket.

Példák:

```
  (i)     RENAME "ZAJ" TO "ZOREJ"
 (ii)     A$="ZAJ": B$="ZOREJ"
          RENAME (A$) TO (B$)
(iii)     RENAME "LOADER" TO "TOLTO" , D1 ON U9
```

Az (i) példa önmagáért beszél. A (ii) példa ugyanazt a feladatot látja el, mint az első, de bemutatja a paraméterek közt sztringváltozók használatát. A (iii) példa a 9-es lemezegység 1-es meghajtójában levő lemezen levő "LOADER" nevű file-t nevezi át "TOLTO"-re.

Hibalehetőségek: Ha a paraméterek nem esnek a megengedett intervallumba, akkor ?ILLEGAL QUANTITY ERROR hibajelzést kapunk. Ha a szóbanforgó lemezegység nincs a rendszerben, akkor ?DEVICE NOT PRESENT ERROR hibaüzenetet kapunk. Ha a <régi név> nevű file nem létezik, akkor ?FILE NOT EXISTS ERROR hibajelzést kapunk. Ha a lemezre írás vagy olvasás közben hiba történik, akkor nem kapunk hibajelzést, csak a lemezegység lámpája villog. Ilyenkor a hibacsatorna kiolvasásával kapjuk meg a hiba pontos okát.

REM C-64-es mód

Rövidítés: nincs Token: $8F (143)

Mód: mind parancs, mind program módban használható.

Megjegyzések elhelyezésére szolgál. A sor REM-et követő része nem kerül végrehajtásra. (Lásd C-64 101. oldal!)

Szintaxis: **REM** <karakter sorozat>

RENUMBER

Rövidítés: renU Token: $F8(248)

Mód: csak parancs módban használható.

A memóriában tárolt program sorait számozza át. A GOTO, GOSUB, IF, RESUME, TRAP, RESTORE utasításokban levő sorszámokat is jól módosítja.

Szintaxis: **RENUMBER** {<új> {,{<növ>} {,<régi>}}}

A program sorszámait a <régi> sorszámú sortól kezdve átszámozza. Ennek a száma <új> lesz. Az átszámozott programsorok száma <növ>-vel növekszik.Ha az első két paraméter valamelyike hiányzik, 10-zel dolgozik az interpreter. Ha a harmadik paraméter hiányzik, az egész programot átszámozza a C-128.

Példák:

```
  (i)     RENUMBER
 (ii)     RENUMBER ,,4500
(iii)     RENUMBER 0,1
```

Az (i) alatti parancsot használjuk talán a leggyakrabban. Hatására az átszámozott program első sorának száma 10 lesz, s a többi sor számozása is 10-zel növekszik. A (ii) parancsot általában akkor használjuk, ha egy meglevő programhoz - pl. a 4500 sortól kezdve - hozzáírtunk egy új részt. Ilyenkor csak a 4500., illetve az ennél nagyobb sorszámú sorokat kell átszámozni. Változások a kisebb sorszámú sorokban is lesznek, hiszen a teljes programban átírásra kerülnek a módosuló hivatkozások.

Hibalehetőségek: A program két ok miatt lehet átszámozhatatlan. Előfordulhat, hogy az átszámozás után túl nagy (>64000) sorszámot kapnánk. Ekkor a ?LINE TOO LARGE ERROR hibajelzést kapjuk. Másik ok, hogy pl. egy GOTO utasítás nem létező sorszámra hivatkozik. Ekkor ?UNRESOLVED REFERENCE ERROR hibaüzenetet kapunk.

RESTORE C-64-es mód

Rövidítés: reS Token: $8C(140)

Mód: mind parancs, mind program módban használható.

Az utasítás hatására a READ utasítások a RESTORE utasításban megadott számú sortól kezdik el olvasni a DATA-kban tárolt adatokat. Ha a paraméter elmarad, az egy RESTORE 0 utasításnak felel meg. C-64-es üzemmódban a RESTORE-nak nem lehet paramétere. (Lásd C-64 102. oldal!)

Szintaxis: **RESTORE** {<sorszám>}

A <sorszám> csak előjel nélküli egész szám lehet.

Példák:

```
(i)     950 DATA ....
        980 DATA ....
        1000 DATA 169,0,141,....: REM GEPI KODU PROGRAM
        1020 DATA ....
        1340 RESTORE 1000
        1345 :
        1350 FOR L=828 TO 912:READ X:POKE L,X:NEXT

(ii)    2010 READ X$: IF X$="END" THEN RESTORE
```

Az első példa a <sorszám> használatát mutatja; hogyan lehet segítségével a DATA utasítások közt egy adott részt megtalálni. Először végrehajtjuk a RESTORE 1000 utasítást, majd ezt követően olvassuk be a gépi kódú rutin byte-jait.

A második példában az utolsó DATA tétel egy "END" sztring. Ennek az olvasása után a RESTORE a DATA mutatót az első utasításra állítja. (Nincs egy külön változó, amelyik azt mutatná, hogy a DATA-k elfogytak.)

A NEW, RUN, CLR utasítások egyben egy RESTORE 0-t is végrehajtanak. Program indítása előtt - parancs módban - a RESTORE-t így csak GOTO <sorszám> típusú indítás esetén érdemes használni.

Hibalehetőség: Nincs. Előfordulhat azonban, hogy egy nem megfelelő sorszámú RESTORE miatt egy READ utasítás végrehajtása közben kapunk hibajelzést.

RESUME

Rövidítés: resU Token: $D6(214)

Mód: csak program módban használható.

A hibakezelő (TRAP) rutinból való kilépés jelzésére szolgál.

Szintaxis: **RESUME** {<sorszám>} **NEXT**

<sorszám> előjel nélküli egész szám kell hogy legyen. Ha a RESUME utasítást paraméter nélkül használjuk, a program - figyelmen kívül hagyva a hibajelzéseket folyamatosan fut tovább. A NEXT használata esetén a hibakezelő rutin a hibás utasítást követő első BASIC utasításra adja vissza a vezérlést. Ha <sorszám>-ot megadjuk, a vezérlés arra kerül. A GOTO-tól ez abban különbözik, hogy a rendszer kilép a hibarutinból.

Példák:

```
(i)     10 TRAP 1000
        20

        ....
```

```
1000 REM HIBAKEZELŐ RUTIN
1010 ?"A HIBAT FIGYELMEN KIVUL HAGYTUK
1020 RESUME NEXT
```

A példában egy olyan hibakezelő rutint mutatunk be, amelyik bármilyen hiba esetén kiírja a képernyőre az 1010. sorban látható szöveget, majd mintha nem is történt volna hiba, folytatja a program futását. Ezt a RESUME-ban használt NEXT paraméter segítségével érjük el.

Hibalehetőségek: Ha a RESUME-ban megadott sor nem létezik, akkor a program futása ?UNDEF'D STATEMENT ERRROR hibajelzéssel megszakad. A RESUME utasítás csak a hibakezelő rutinban lehet. Ha ettől eltérő helyen kerül rá a vezérlés, akkor ?CAN'T RESUME ERROR hibajelzést kapunk.

RETURN

C-64-es mód

Rövidítés: reT Token: $8E (142)

Mód: mind parancs, mind program módban használható.

Alprogramból való vissatérés. (Lásd C-64 103. oldal!)

Szintaxis: **RETURN**

RGR

Rövidítés: rG Token: $CC(204)

Mód: mind parancs, mind program módban használható.

Az egyváltozós függvény az érvényes grafikus mód kódjával tér vissza. A változó **álargumentum** (dummy variable) csak a szintaxis miatt szerepel. A kapott kód jelentése:

érték	Kijelzési mód
0	40 oszlopos, normál, karakteres képernyő
1	40 oszlopos, normál, nagyfelbontású képernyő
2	40 oszlopos, normál, vágott képernyő
3	40 oszlopos, többszínű, nagyfelbontású képernyő
4	40 oszl0pos, többszínű, vágott képernyő
5	80 oszlopos, karakteres képernyő

Szintaxis: **RGR(<kifejezés>)**

Példák:

```
(i)     GRAPHIC 1,1: ? RGR(0): REM = 1
        GRAPHIC 3: ? RGR(121): REM = 3
        GRAPHIC 3: ? RGR(X+Y): REM = 3
```

Példáink az RGR hatását mutatják be. Az utolsó két sor mutatja, hogy az RGR függvény értéke független az argumentumától.

Hibalehetőségek: Nincs.

RIGHT$ C-64-es mód

Rövidítés: rI Token: $C9 (201)

Mód: mind parancs, mind program módban használható.

Sztring függvény: az argumentum sztring adott hosszúságú végszeletével (jobboldali karaktereivel) tér vissza. (Lásd C-64 104. oldal!)

Szintaxis: **RIGHT$(** <sztring kif.> , <arit.kif.>)

RND C-64-es mód

Rövidítés: rN Token: $BB (187)

Mód: mind parancs, mind program módban használható.

Függvény: a 0-1 intervallumba eső véletlen számokat generál. A argumantum előjelétől a generálási eljárás függ csak. (Lásd C-64 105. oldal!)

Szintaxis: **RND(** <arit.kif.> ,

RREG

Rövidítés: rR Token: $FE $09 (254,9)

Mód: mind parancs, mind program módban használható.

Gépi kódú alprogram futtatása után a regiszterek értékét adja vissza.

Szintaxis: **RREG** {<Avar>}{,{<Xvar>}{,{<Yvar>}{,<Svar>}}}

A SYS utasítással meghívott alprogramokat az interpreter úgy indítja el, hogy az A,X,Y és S regisztereket rendre a $0006, $0007, $0008, illetve a $0005 memóriákból tölti fel, s a regiszterek értékét egy RTS gépi kódú utasítás végrehajtása után visszamásolja. Az RREG utasítás ezek értékét tölti a változókba.

Példák:

```
  (i)    RREG A,X,Y,S
 (ii)    S=PEEK(5): A=PEEK(6): X=PEEK(7): Y=PEEK(8)
```

(ii) pontosan ugyanazt végzi el, mint az (i) parancs. (i) használatához azonban nem kell tudni, pontosan hol is vannak ezek a változók a memóriában elhelyezve.

Hibalehetőségek: Ha az RREG utasitásban kifejezést vagy sztringet használunk paraméterként, akkor ?SYNTAX ERROR hibaüzenetet kapunk.

RSPCOLOR

Rövidítés: rspC Token: $CE $07 (206,7)

Mód: mind parancs, mind program módban használható.

A függvény a sprite-többszín regiszterek értékét adja.

Szintaxis: **RSPCOLOR(<sorszám>)**

<sorszám> tetszőleges aritmetikai kifejezés, értéke azonban 1 vagy 2 lehet. RSPCOLOR értéke pedig a lehetséges 16 féle színkód: 1-16.

Példák:

```
? RSPCOLOR(1)
```

A példa az első sprite-többszín regiszter értékét irja ki a képernyőre.

Hibalehetőségek: Ha a <sorszám> értéke nem 1 vagy 2, akkor ?ILLEGAL QUANTITY ERROR hibajelzést kapunk.

RSPPOS

Rövidítés: rS Token: $CE $05 (206,5)

Mód: mind parancs, mind program módban használható.

A kétváltozós függvény a sprite-ok helyzetének és sebességének a lekérdezésére szolgál.

Szintaxis: **RSPPOS(<sprite>,<param>)**

<sprite> a sprite sorszáma, igy értéke az 1-8 intervallumba esik. A <param> értéke 0-2 lehet. Az egyes értékek jelentése a következő:

0	x koordináta;
1	y koordináta;
2	sebesség (0-15).

Példák:

(i)
```
? RSPPOS(3,1)
```

(ii)
```
10 FOR I=1 TO 8
20 IF RSPPOS(I,2)<>0 THEN PRINT I;"MOZOG!"
30 NEXT I
```

Az (i) példa megkérdezi, mennyi a 3. sprite x koordinátája. A (ii) program kiírja a mozgó sprite-okat.

Hibalehetőségek: Ha a függvény argumentumai nem esnek a megadott intervallumba, akkor ?ILLEGAL QUNTITY ERROR jelzést kapunk.

RSPRITE

Rövidítés: rspR Token: $CE $06 (206,6)

Mód: mind parancs, mind program módban használható.

A függvény a sprite-ok SPRITE utasítással beállított jellemzőit adja meg.

Szintaxis: **RSPRITE(<sorszám>,<param>)**

<sorszám> a sprite sorszám, értéke 1-8 lehet. <param> az 1-5 intervallumba kell, hogy essen. Jelentése:

<param>	<jelentés>
0	1=aktiv, 0=nem;
1	a sprite színe (1-16);
2	0=sprite látszik a háttér előtt, 1=nem;
3	1=X-irányú nagyítás, 0=nem;
4	1=Y-irányú nagyítás, 0=nem;
5	1=többszínü sprite, 0=nem.

Példák:

```
(i)     SPRITE 3,1,,0,1,,1
        ?RSPRITE(3,0): REM =1, aktiv
        ?RSPRITE(3,2): REM =0, a sprite látszik
        ?RSPRITE(3,3): REM =1, X-irányú nagyítás
        ?RSPRITE(3,4): REM =0, nincs Y-irányú nagyítás
        ?RSPRITE(3,5): REM =1, többszínű
```

A példa - úgy gondoljuk - magáért beszél.

Hibalehetőségek: Ha a paraméterek nem esnek a megadott intervallumba, akkor ?ILLEGAL QUANTITY ERROR hibajelzést kapunk.

RUN C-64-es mód

Rövidítés: rU Token: $8A(138)

Mód: mind parancs, mind program módban használható.

A memóriában vagy a lemezen tárolt BASIC programot elindítja. A változók előző értéke elvész, a RUN ugyanis végrehajt egy CLR és egy RESTORE utasítást is.

Szintaxis:

```
RUN {<filenév> {,D<meghajtó>} {,U<egység>}
    {<sorszám>}
```

C-64-es üemmódban csak a RUN {<sorszám>} alak használható.

Példák:

```
  (i)     RUN
 (ii)     RUN 1000
(iii)     5000 IF X<>0 THEN RUN
 (iv)     RUN "PROBA", U9
```

Az első két sor a RUN parancs módban való használatát szemlélteti. (iii)-ban, ha X<>0, akkor a RUN utasítás törli az összes változót és a program elölről kezd el futni. (iv) hatására a 9-es hardver számú lemezegységről betöltődik a "PROBA" nevű program és elindul.

A <RUN> billentyű lenyomása ekvivalens a

```
DLOAD "*"<RETURN>
RUN  <RETURN>
```

billentyűzésével, tehát a lemezről betölti az első programot, és azonnal el is indítja.

Hibalehetőség: Ha a RUN utasításban megadott sorszámú programsor nem létezik, akkor ?UNDEF'D STATEMENT ERROR hibajelzést kapunk. Ha a megadott nevű file nem létezik, akkor ?FILE NOT FOUND ERROR hibajelzést kapunk. Az X=80: RUN X hatására a program nem a 80., hanem a 0. sortól kezd futni, de nem kapunk hibajelzést!

RWINDOW

Rövidítés: rW Token: $CE $09 (206,9)

Mód: mind parancs, mind program módban használható.

A használt ablak méretét és típusát adja meg.

Szintaxis: **RWINDOW**(<param>)

A <param> argumentum értéke 0,1 vagy 2 lehet. Az egyes értékek jelentése:

0	a használt ablak sorainak száma (0-24);
1	a használt ablak oszlopainak száma (0-79);
2	40 vagy 80 oszlopos kijelzés.

Példák: Lásd WINDOW alatt!

Hibalehetőségek: Ha a <param> értéke nem megfelelő, akkor ?ILLEGAL QUANTITY ERROR üzenettel megszakad a program futása.

Rövidítés: sA Token: $94 (148)

Mód: mind parancs, mind program módban használható.

A BASIC program elmentése adott file-ba. (Lásd C-64 107. oldal!)

Szintaxis: **SAVE** {<filenév>}{,<egység>}{,<mód>}

SCALE

Rövidítés: scA Token: $E9(233)

Mód: mind parancs, mind program módban használható.

A nagyfelbontású képernyő léptékének megadására szolgál.

Szintaxis:
SCALE {<mód>}{,<xmax>,<ymax>}

Ha <mód> értéke 0, akkor a nagyfelbontású képernyőt 320*200-as, többszínű üzemmódban pedig 160*200-s lesz. (Ez megfelel a fizikai méreteknek.) Ha <mód> értéke 1, akkor a nagyfelbontású képernyő léptékét magunk választhajuk meg. A vízszintes és függőleges felbontást <xmax> és <ymax> adja meg. <xmax> értéke normál képernyő esetén a 320-1023, többszínű képernyő esetén a 160-1023 intervallumba eshet. <ymax> értékének pedig a 200-1023 intervallumba kell esnie.

Példák:
```
  (i)    SCALE 0
 (ii)    GRAPHIC 1,1: SCALE 1,1: CIRCLE ,512,512,300,300
```

Az (i) példa a fizikai méreteknek megfelelő kijelzésre való visszatérést eredményezi. (ii)-ben a CIRCLE utasításban a két féltengelyt egyenlőnek adtuk meg, a képernyőn mégis egy 'igazi' ellipszis látható. Ez azért van, mert a transzformáció x és y irányban nem egyforma méretű.

Hibalehetőségek: Ha a paraméter értéke nem 0 vagy 1, akkor ?ILLEGAL QUANTITY ERRROR hibajelzést kapunk.

SCNCLR

Rövidítés: sC Token: $E8(232)

Mód: mind parancs, mind program módban használható.

A kiválasztott grafikus módnak megfelelő képernyő(ke)t törli.

Szintaxis: **SCNCLR** {<sorszám>}

<sorszám> értéke a következő lehet:
0 40 oszlopos, normál, karakteres képernyő;
1 40 oszlopos, normál, nagyfelbontású képernyő;
2 40 oszlopos, normál, vágott képernyő;
3 40 oszlopos, többszínű nagyfelbontású képernyő;
4 40 oszlopos, többszínű vágott képernyő;
5 80 oszlopos, karakteres képernyő.

Példák:
(i) GRAPHIC 1: SCNCLR
(ii) GRAPHIC 3: SCNCLR

Az (i) példa törli a nagyfelbontású képernyőt. A (ii) példában vágott képernyőt használunk, ennek megfelelően a SCNCLR törli a **teljes** karakteres és a **teljes** nagyfelbontású képernyőt.

Hibalehetőségek: Nincs. (De a képernyő tartalma elvész!)

SCRATCH

Rövidítés: sC Token: $F2(242)

Mód: mind parancs, mind program módban használható.

A paramétereiben megadott meghajtóban levő, adott nevű file-t törli a lemezről.

Szintaxis:
SCRATCH <filenév> {,D<meghajtó>} {,U<egység>}

A két aritmetikai kifejezés a lemezegység meghajtóját, illetve a hardver számát adja meg. Ha a filenév nem sztring konstans, akkor kerek zárójelek közé kell tenni. A <név>-ben a * és ? karakterek jelentése a szokásos.

Példák:
(ia) SCRATCH "KUKAC"
(ib) X$="KUKAC": SCRATCH (X$)
(ii) SCRATCH "TEMP*",D1
(iii) SCRATCH "????.PRG",D0 ON U10

Az (ia) és (ib) alatti parancsok egyaránt a 8-as lemezegység 0-ás meghajtójában levő lemezről törlik a "KUKAC" nevü file-t. A (ii) parancs törli a 8-as lemezegység 1-es meghajtójában levő lemezről az összes "TEMP"-pel kezdődő nevű file-t.

A (iii) parancs a 10-es lemezegység 0-ás meghajtójában levő összes olyan file-t törli, amelyik neve pontosan 8 karakterből áll, s az utolsó négy karakter ".PRG"

A parancsok végrehajtása előtt a rendszer visszakérdez:

ARE YOU SURE? (Biztos benne?)

Ha igen, akkor a <Y>, majd a <RETURN> billentyűt kell válaszként

megnyomni. Erre az interpreter elküldi a lemezegységnek a megfelelő parancsot. Ha az <N> billentyűt nyomjuk meg, a parancs figyelmen kívül marad.

Hibalehetőségek: A kifejezések kiértékelése közben számos hiba adódhat. Ha az aritmetikai kifejezések értéke nem megfelelő akkor ?ILLEGAL QUANTITY ERROR hibajelzést kapunk. Ha a lemezegység nincs a rendszerben, akkor ?DEVICE NOT PRESENT ERROR hibajelzést kapunk. Ha a törlés közben írási vagy olvasási hiba történik, akkor erről csak a hibacsatorna kiolvasásával szerezhetünk információt.

SGN

C-64-es mód

Rövidítés: sG Token: $84 (180)

Mód: mind parancs, mind program módban használható.

Függvény: az argumentum előjelével tér vissza. (Lásd C-64 108. oldal!)

Szintaxis: **SGN(** <arit.kif> **)**

SIN

C-64-es mód

Rövidítés: sI Token: $BF (191)

Mód: mind parancs, mind program módban használható.

Függvény: a argumentum sinuszát adja. (Lásd C-64 109. oldal!)

Szintaxis: **SIN(** <arit.kif> **)**

SLEEP

Rövidítés: sL Token: $FE $0B (254,11)

Mód: mind parancs, mind program módban használható.

Meghatározott ideig felfüggeszti az interpreter működését.

Szintaxis: **SLEEP** <idő>

Az <idő> értéke 1-65535 lehet. Az idő másodpercekben értendő.

Példák:

```
  (i)    SLEEP 1
 (ii)    1200 X%=RND(1)*10: SLEEP X%
```

Az (i) alatti parancs kiadás után 1 másodpercre lefagy a rendszer. (ii)-ben egy 0-9 másodperc között véletlenszerűen megválasztott időre

áll meg a program futása.

Hibalehetőségek: Nincs.

SLOW

Rövidítés: sl0 Token: $FE $26 (254,38)

Mód: mind parancs, mind program módban használható.

A processzor órajelét 1MHz-re állítja.

Szintaxis: **SLOW**

A 40 oszlopos képernyő csak 1MHz-es órajel esetén használható. Ha egy számítássorozat kezdetén kiadtuk a FAST parancsot, akkor az eredmények megjelenítésére ki kell adnunk a SLOW parancsot.

Hibalehetőségek: Nincs.

SOUND

Rövidítés: s0 Token: $DA(218)

Mód: mind parancs, mind program módban használható.

A hanggenerátor chip meghatározott oszcillátorát adott ideig és adott hangmagassággal bekapcsolja.

Szintaxis:
SOUND<ossz>,<frek>,<ütem> {{,<mód>}{,{<max.frek>}
{,{<idő>} {,<hullám>} {,{<imp>}}}}}

<ossz> az osszcillátor száma, ami megszólaltatja a hangot, értéke 1,2 vag 3 lehet. <frek> a hang frekvenciáját kifejező szám. Ennek értéke 0-65535 lehet, egyenesen arányos a megszól aló hang magasságával. <ütem> azt az időt adja meg, amíg a hang szól. Ezt 1/60 másodpercben kell megadni, maximális értéke 32767 lehet.

Lehetőség van a hang megszólalásának erejét is programozni. Ennek módját adja a <mód> paraméter. Lehetséges értékei:

0 emelkedő
1 süllyedő
2 osszcilláló

<max.frek> a modulált hang maximális frekvenciáját, <idő> pedig a modulálás idejét adja meg. értékükre a <frek>-nél, illetve az <ütem>-nél mondottak érvényessek. <hullám> a megszólaló hang hullámformáját adja meg:

0 háromszög
1 fűrészfog
2 négyszögimpulzus
3 fehér zaj

Lehetőség van a négyszögimpulzus arányainak megadására. Az <imp> paraméternek a 0-4095 intervallumba kell esnie. 2048 felel meg a szimmetrikus jelnek.

Példák:
(i) SOUND 2,800,6000
(ii) SOUND 1,770,300

Az (i) példa két percig a 800 értéknek megfelelő frekvenciájú ún. fehér zajt produkál. (ii) rövid időre megszólaltatja a normál zenei 'A' hangot.

Hibalehetőségek Az aritmetikai kifejezések értékének a jelzett intervallumba kell esnie. Ha nem, akkor ?ILLEGAL QUANTITY ERRROR hibaüzenetet kapunk.

SPC (

C-64-es mód

Rövidítés: sP Token: $A6 (166)

Mód: mind parancs, mind program módban használható.

A PRINT utasításban szóközök nyomtatására használható. (Lásd C-64 110. oldal!)

Szintaxis: **SPC(** <arit.kif.>)

SPRCOLOR

Rövidítés: sprC Token: $FE $08 (254,8)

Mód: mind parancs, mind program módban használható.

A többszínű sprite-ok két közös színét állítja be.

Szintaxis: **SPRCOLOR** {<szín1>}{,<szín2>}

Az utasítás segítségével nem az egyes sprite-ok színét lehet megadni (ahogy azt az utasítás neve sugallja), hanem a többszínű sprite-ok **közös** színét. <szín1> és <szín2> értéke ennek megfelelően az 1-16 intervallumba kell hogy essen.

Példák:
(i) SPRCOLOR 1,2
(ii) X=1: SPRCOLOR X,X+1

Az (i) utasítás hatására a két sprite többszín értéke 1 (fekete), illetve 2 (fehér) lesz. (ii)-nek természetesen ugyanez a hatása, de mutatja, hogy az utasításban tetszőleges kifejezéseket is használhatunk.

Hibalehetőségek: Ha a paraméterek értéke nem esik az 1-16 intervallumba, akkor ?ILLEGAL QUANTITY ERROR hibajelzést kapunk.

SPRDEF

Rövidítés: sprD Token: $FE $1D (254,29)

Mód: mind parancs, mind program módban használható.

Bekapcsolja a C-128 sprite szerkesztőjét.

Szintaxis: SPRDEF

Az utasítás kiadása után a C-128 sprite szerkesztőjébe lépünk be, s lehetőségünk nyílik a sprite-ok alakjának definiálására. A SPRDEF utasításhoz a 40 oszlopos képernyőt kell használnunk. (A 80-ason nincsenek sprite-ok!) A képernyő bal felső sarkában egy eltérő színű 21*24 karakterhelyből álló téglalap jelenik meg. Ennek karakterhelyei a sprite pontjainak felelnek meg. A képernyőn a SPRITE NUMBER? kérdés jelenik meg. Erre az 1-8 billentyűk lenyomásával válaszolhatunk. Ezután a kiválasztott sprite megjelenik a képernyő jobb oldalán, s kinagyítva a képernyő elkerített részén. A megszokott villogó kurzor helyett egy + jel jelenik meg. Ezt a kurzor billentyűk segítségével ugyanúgy mozgathatjuk, mint az igazi kurzort. A sprite pontjainak színét az 1,2,3 és 4 billentyűk lenyomásával állíthatjuk be. A szerkesztést az alábbi karakterekkel befolyásolhatjuk:

A	az 1,2,3,4 billentyűk ismétlését kapcsolja ki/be;
<CLR>	törli a kiválasztott sprite-ot;
<CTRL n>	beállítja a sprite színét (n=1,...,8)
<C=-n>	beállítja a sprite színét (n=1,...,8)
X	a sprite X irányú nagyítása;
Y	a sprite Y irányú nagyítása;
M	többszínű üzemmód ki/be kapcsolása.

A sprite alakját a <SHIFT-RETURN> billentyűzéssel tárolhatjuk. Ezután lehetőség van további sprite-ok alakjának megadására. Ha már valamennyi alakját megadtuk, akkor a <RETURN> megnyomásával térhetünk vissza a BASIC-hez. előfordulhat, hogy a sprite alakját nem akarjuk tárolni. Ekkor a <STOP> billentyűt nyomjuk meg, majd pedig újból a <RETURN>-t.

A sprite-ok alakját az interpreter a 0-ás szelet $0E00-$0FFF (3584-4095) címeken tárolja. Ha a sprite-ok alakját lemezen akarjuk tárolni, akkor ezt a

```
BSAVE "SPRITEOK", B0, P3584 TO P4095
```

utasítással tehetjük meg. Ennek hatására a memória adott része a "SPRITEOK" nevű program file-ba mentődik. A sprite-ok alakját a

```
BLOAD "SPRITEOK"
```

utasítással tölthetjük vissza.

Hibalehetőségek: A sprite szerkesztő nem reagál az értelmetlen billentyűk lenyomására. Ha a sprite-okat már mozgattuk, előfordulhat, hogy a sprite kiválasztása után nem jelenik meg a képernyőn. Ekkor a <STOP-RESTORE> parancs kiadásával inicializáljuk a VICII chipet, s újból adjuk ki a SPRDEF parancsot.

SPRITE

Rövidítés: sP Token: $FE $07 (254,7)

Mód: mind parancs, mind program módban használható.

A sprite-ok paramétereinek beállítására szolgál.

Szintaxis:
SPRITE <sorszám> {,{<aktiv>}{,{<szín>}{,{<pri>}
{,{<Xnagy>}{,{<Ynagy>}{,{<mód>}}}}}}}

Az utasítás a <sorszám>-ú sprite paramétereit állítja be, így <sorszám> értéke 1-8 lehet csak. Az egyes paraméterek jelentése a következő:

<aktív>	1=a sprite látható, 0=nem
<szín>	a sprite színe (1-16)
<pri>	1=háttér látszik a sprite előtt, 0=nem
<Xnagy>	1=X irányban nagyított, 0=nem
<Ynagy>	1=Y irányban nagyított, 0=nem
<mód>	1=többszínű, 0=normál

Példák:

```
  (i)    SPRITE 5,1,2
 (ii)    10 I=0
         20 FOR K=1 TO 10
         30 I=1-I
         40 SPRITE 5,,,,I,I
         50 NEXT K
```

Az (i) példa az 5-ös sprite-ot bekapcsolja, és a színét fehérre állítja (színkód=2). A sprite nem biztos, hogy látszik: ez függ a sprite helyzetétől továbbá a többi sprite-tól és a háttértől is. (ii) 5-ször nagyítja majd újból kicsinyíti az 5. sprite-ot.

Hibalehetőségek: A paramétereknek a jelzett intervallumba kell esnie. Ha nem, akkor ?ILLEGAL QUANTITY ERROR hibajelzést kapunk.

SPRSAV

Rövidítés: sprS Token: $FE $16 (254,22)

Mód: mind parancs, mind program módban használható.

Adott sprite alakját egy sztringben tárolja, és megfordítva.

Szintaxis: **SPRSAV** <sorszám>,<sztring változó>
SPRSAV <sztring>,<sorszám>

Az utasítás első formája a <sorszám>-ú sprite alakját definiáló 64 byte-ot a <sztring> változóban tárolja. A második esetben a <sztring> 64 byte-ját az interpreter átírja a <sorszám>-ú sprite alakját definiáló memória részbe.

```
  (i)    SPRSAV 3,A$
```

(ii)
```
100 MOVSPR 2,90#20
110 SPRSAV A$,2
120 GOTO 130
130 SPRSAV B$,2
140 GOTO 110
```

Az (i) alatti példa az 3-as sprite alakját elmenti az A$ sztingbe. A (ii) programrész elején elkezdjük mozgatni a sprite-ot, majd az alakját a 110-140 ciklusban folyamatosan változtatjuk: hol A$, hol B$-nak megfelelő lesz a sprite alakja. így érhetjük el, hogy egy pók araszoljon, egy autón lobogjon a zászló stb.

Hibalehetőség: Ha a változók helyett kifejezést használunk ?SYNTAX ERROR hibajelzést kapunk.

SQR

C-64-es mód

Rövidítés: sQ Token: $BA (186)

Mód: mind parancs, mind program módban használható.

Függvény: a nem negatív argumentum négyzetgyökét számítja ki. (Lásd C-64 111. oldal!)

Szintaxis: **SQR**(<arit.kif.>)

SSHAPE

Rövidítés: sS Token: $E4(228)

Mód: mind parancs, mind program módban használható.

A nagyfelbontású képernyő egy téglalap alakú részét - a GSHAPE utasítás segítségével visszatölthető alakban - átmásolja egy sztringváltozóba.

Szintaxis: **SSHAPE** <sztring változó>,<x1>,<y1> {,<x2>,<y2>}

Az <x1>,<y1> az átmásolandó téglalap bal felső, míg <x2>,<y2> a jobb alsó csúcsának koordinátái. Ha ez utóbbi elmarad, akkor a grafikus kurzor helyzete határozza meg a bal alsó sarkot. <x1>,<y1>,<x2>,<y2> tetszőleges aritmetikai kifejezések lehetnek. A <sztring vált.> hossza normál, illetve több színű üzemmódban a következőképpen alakul:

```
INT((ABS(X1-X2)+1)/4+.99)*(ABS(Y1-Y2)+1)+4      (Normál)
INT((ABS(X1-X2)+1)/8+.99)*(ABS(Y1-Y2)+1)+4      (Több színű)
```

Példák: Lásd a GSHAPE-nél szereplő példát.

Hibalehetőségek A kifejezések kiértékelése közben számos hiba adódhat. Ha a téglalap koordinátáit úgy adjuk meg, hogy a sztring hossza nagyobb lenne, mint 255, akkor ?ILLEGAL QUANTITY ERROR hibajelzést kapunk.

ST

C-64-es mód

Rövidítés: nincs Token: nincs

Mód: mind parancs, mind program módban használható.

Fenntartott BASIC változó. értéke az utolsó adatátviteli művelet eredményéről tájékoztat. (Lásd C-64 111. oldal!)

STASH

Rövidítés: sT Token: $FE $1F (254,31)

Mód: mind parancs, mind program módban használható.

A BASIC munkaterület egy részét a memóriabővítés megadott lapjára másolja át.

Szintaxis: **STASH** <db.szám>,<b.cím>,<k.cím>,<szelet>

Az utasítás hatására a BANK utasításban megadott memória-szelet <b.cím>-étől kezdődően <db.byte> darab byte-ot másol át az interpreter a memóriabővítés <szelet>-nek megfelelő szeletére, a <k.cím>-től kezdődően.

Példák:

```
(i)     BANK 0
        STASH 1000,1024,40000,7
```

Egyetlen példánkban a képernyő memória a memória bővítés 7. szeletére másolódik a 40000-es címtől kezdve.

Hibalehetőségek: A címeknek a 0-65535 tartományba kell esniük. Ha nem használunk memóriabővítést, akkor arról nem kapunk hibajelzést.

STOP

C-64-es mód

Rövidítés: stO Token: $90 (144)

Mód: mind parancs, mind program módban használható.

Megállítja a futó BASIC programot. CONT-tal újra indítható. (Lásd C-64 112. oldal!)

Szintaxis: **STOP**

STR$ C-64-es mód

Rövidítés: stR Token: $C4 (196)

Mód: mind parancs, mind program módban használható.

Sztring függvény: a argumentumaként megadott szám karakteres alakjával tér vissza. (Lásd C-64 113. oldal!)

Szintaxis: **STR$**(<arit.kif.>)

SWAP

Rövidítés: sW Token: $FE $23 (254,35)

Mód: mind parancs, mind program módban használható.

A BASIC munkaterületnek és a memóriabővítés adott részeit felcseréli.

Szintaxis: **SWAP** <db.byte>,<b.cím>,<k.cím>,<szelet>

Az utasítás hatására a BANK utasításban megadott memória-szelet <b.cím>-étől kezdődően <db.byte> darab byte-ot cserél át az interpreter a memóriabővítés <szelet>-nek megfelelő szeletére, a <k.cím>-től kezdődő byte-jaival.

Példák:

```
(i)     BANK 0
        SWAP 512,DEC("E00"),35000,7
```

Az utsítás hatására a sprite-ok definíciói (DEC("E00")-tól 512 byte) helyet cserélnek a memóriabővítés 7. szeletének 35000-től kezdődő byte-okkal.

Hibalehetőségek: A címeknek a 0-65535 tartományba kell esnie . Ha nem használunk memóriabővítést, akkor arról nem kapunk hibajelzést.

SYS C-64-es mód

Rövidítés: sY Token: $9E (158)

Mód: mind parancs, mind program módban használható.

Gépi kódú alprogram meghívása. (Lásd C-64 114. oldal!) Szemben a C-64-es móddal az utasítás paraméterei közt megadhatjuk az A,X,Y és a PS (processzor státusz szó) regiszter tartalmát!

Szintaxis:
SYS <cím> {,<Areg> {,<Xreg> {,<Yreg> {,<PSreg>}}}}

TAB(C-64-es mód

Rövidítés: tA Token: $A3 (163)

Mód: mind parancs, mind program módban használható.

A PRINT utasításban adott pozícióig szóközöket nyomtat. (Lásd C-64 114. oldal!)

Szintaxis: **TAB(**<arit.kif.>**)**

TAN C-64-es mód

Rövidítés: nincs Token: $C0 (192)

Mód: mind parancs, mind program módban használható.

Függvény: a argumentum tangensét sámítja ki. (Lásd C-64 115. oldal!)

Szintaxis: **TAN(**<arit.kif.>**)**

TEMPO

Rövidítés: tE Token: $FE $05 (254,5)

Mód: mind parancs, mind program módban használható.

A PLAY utasítással lejátszandó dallamok tempóját állítja be.

Szintaxis: **TEMPO** <tempo>

<tempo> a dallam tempójával arányos szám, értéke a 0-255 intervallumba kell, hogy essen. A PLAY utasításban szereplő dallamot az interpreter úgy játsza le, hogy egy másodpercre 19.22/<tempo> ütem (felütés) jut.

Példák:
```
  (i)     TEMPO 12
 (ii)     X=120: TEMPO 120
```

(i)-ben egy nagyon lassú, (ii)-ben egy kicsit gyors tempót állítottunk be.

Hibalehetőségek: Ha a paraméter értéke nem esik az adott intervallumba, akkor ?ILLEGAL QUANTITY ERROR hibajelzést kapunk.

TI és TI$ C-64-es mód

Rövidítés: nincs Token: nincs
Mód: mind parancs, mind program módban használható.

Fenntartott BASIC változók. TI a gép bekapcsolása óta eltelt időt méri 1/60 másodpercben. TI$ az időt karakteres alakban tárolja. TI$ értéke megadható. (Lásd C-64 116. oldal!)

TRAP

Rövidítés: tR Token:$D7(215)

Mód: csak program módban használható.

Saját hibakezelő rutin kezdőcímét adja meg. Ha az utasítást követően hiba történik, akkor a vezérlés a TRAP-ben megadott címre kerül. A hibakezelő rutinból a RESUME utasítás kiadásával léphetünk ki.

Szintaxis: **TRAP** [<sorszám>]

A <sorszám> csak előjel nélküli egész szám lehet. Ha hiányzik, az a 0 értéknek felel meg.

Példák:
(i) TRAP 1200

Az utasítás végrehajtását követően bármely hiba után a vezérlés az 1200-as sorra kerül. A program folytatása a hibakezelő rutin által elsőnek végrehajtott RESUME-nak megfelelően történik.

Hibalehetőség: Ha a TRAP-ban szereplő sorszámú sor nem létezik, akkor nem kapunk hibajelzést. Ha ezek után történik hiba, akkor a program futása hibajelzés nélkül megszakad.

TROFF és TRON

Rövidítés: troF Token: TROFF $D9(217)
trO TRON $D8(216)

Mód: mind parancs, mind program módban használható.

A TRON bekapcsolja, a TROFF pedig kikapcsolja a nyomkövetési üzemmódot. Nyomkövetési üzemmódban a végrehajtott BASIC utasítások sorszámait az interpreter szögletes zárójelek közé téve folyamatosan kiírja a képernyőre. Ha programból használjuk, akkor lehetőség van arrra, hogy csak a program egyes részeit kövessük nyomon.

Szintaxis: **TRON**
TROFF

Hibalehetőségek: Nincs.

UNTIL

Rövidítés: uN Token: $FC(252)

Mód: mind parancs, mind program módban használható.

A DO vagy LOOP utasításban levő kilépési feltétel megadására használhatjuk. A ciklus akkor jár le, ha az UNTIL utáni logikai kifejezés igazzá válik.

Szintaxis:

```
  DO
     } UNTIL<logikai kifejezés>
  LOOP
```

Példák

```
(i)   DO UNTIL JOY(1)>0: LOOP
(ii)  1200 DO
      1210 INPUT "<CLR>X=";X
      1220 LOOP UNTIL X=INT(X)
```

Az (i) parancsban levő ciklus addig nem fejeződik be, míg az UNTIL-t követő kifejezés igaz nem lesz, azaz egészen addig, amíg az 1-es botkormánnyal nem csinálunk valamit. (ii) egy beolvasó ciklus. A ciklus az 1210-es INPUT utasítást addig ismétli, míg X-nek egész értéket nem adunk.

Hibalehetőségek: A logikai kifejezés kiértékelése közben számos hiba történhet. További hibát okozhat az UNTIL-t tartalmazó DO vagy LOOP nem megfelelő használata.

USR

C-64-es mód

Rövidítés: uS Token: $B7(183)

Mód: mind parancs, mind program módban használható.

Aritmetikai függvény, amelyik a felhasználó által definiált gépi-kódú programot hajtja végre.

Szintaxis: **USR(**<aritmetikai kifejezés>**)**

Az aritmetikai kifejezés értékének a 0-65535 intervallumba kell esnie (kerekítés után). Ezenkívül a $1218-121A (4632-4634) címen egy gépi-kódú utasításnak kell lennie (ez általában JMP ($xxxx) alakú). Az interpreter a függvény argumentumának értékét az 1. lebegőpontos akkumulátorba teszi, s utána egy JSR $1218 utasítást hajt végre. A gépi kódú programból való visszatérés után az 1. lebegőpontos akkumulátor értékét két byte-os egész számmá konvertálja, s ez lesz a USR függvény értéke. **Figyelem:** C-64-es üzemmódban a 784-786-os címeken kell egy gépi kódú utasításnak lennie! (Lásd C-64 117. oldal!)

Hibalehetőségek: A BASIC interpreter nem ellenőrzi a gépi kódú program futását. Rosszul megírt gépi kódú rutin lemerevítheti a rendszert, a <STOP> nem működik, stb.

VAL

c-64-es mód

Rövidítés: vA Token: $C5 (197)

Mód: mind parancs, mind program módban használható.

Függvény: az argumentum sztringet számnak tekinti, s annak értékével tér vissza. (Lásd C-64 117. oldal!)

Szintaxis: **VAL**(<arit.kif.>)

VERIFY

C-64-es mód

Rövidítés: vE Token: $95 (149)

Mód: mind parancs, mind program módban használható.

Adott file tartalmát összehasonlítja a BASIC programmal. (Lásd C-64 118. oldal!)

Szintaxis: **VERIFY** {<filenév> {,<egység> {,<mód>}}}

VOL

Rövidítés: vO Token: $DB(219)

Mód: mind parancs, mind program módban használható.

A hanggenerátor hangerejét a VOL-ban megadott értékre (0-15) állítja be. Ha ez 0, akkor a hanggenerátor egyáltalán nem szólal meg. Mind a PLAY, mind a SOUND utasítás előtt ki kell adni.

Szintaxis: **VOL** <hangerő>

Példák:

```
(i)     VOL 0
(ii)    10 VOL 15
        20 SOUND 2,600,50
(iii)   100 FOR I=0 TO 3
        110 VOL 7-I
        ....
        300 NEXT I
```

Az (i) példa kikapcsolja a C-128 hanggenerátorát. A (ii) zenélő programokban tipikus: maximumra kapcsoljuk a hanggenerátort. (iii) egy cikluson belül folyamatosan csökkenti a hangerőt.

Hibalehetőségek: Nincs.

Rövidítés: wA Token: $92 (146)

Mód: mind parancs, mind program módban használható.

A program addig várakozik, míg a

PEEK(<cím>) EOR <maszk2>) AND <maszk1>

kifejezés értéke Ø-val egyenlő. Mihelyt ez az érték először eltér Ø-tól, a program futása folytatódik. (Lásd C-64 119. oldal!)

Szintaxis: **WAIT** <cím>, <maszk1>, <maszk2>

WHILE

Rövidítés: wH Token: $FD(253)

Mód: mind parancs, mind program módban használható.

A DO vagy LOOP utasításban levő kilépési feltétel megadására használhatjuk. A ciklus akkor jár le, ha a WHILE utáni logikai kifejezés hamissá válik.

Szintaxis:

DO / **LOOP** } **WHILE** <logikai kifejezés>

Példák:

```
  (i)     DO WHILE JOY(1)>Ø: LOOP
 (ii)     2300 DO
          2310 INPUT "<CLEAR>X=";X
          2320 LOOP WHILE X<Ø
```

(i)-ben a ciklus addig jár, amíg az 1-es botkormány nem kerül alaphelyzetbe. A (ii) alatti beolvasó rutin ciklusa addig jár, amíg a 2320-as sorban a WHILE utáni feltétel igaz, azaz X<Ø. A ciklus tehát az első nem negatív érték beírásakor jár le.

Hibalehetőségek: A logikai kifejezés kiértékelése közben számos hiba történhet. További hibát okozhat az WHILE-t tartalmazó DO vagy LOOP nem megfelelő használata.

WIDTH

Rövidítés: wiD Token: $FE $1C (254,26)

Mód: mind parancs, mind program módban használható.

Beállítja a grafikus utasítások vonalvastagságát. Ez lehet normál,

vagy dupla széles rajzolás. Ennek megfelelően az utasítás egyetlen <mód> paramétere 0 vagy 1.

Szintaxis **WIDTH** <mód>

Példák
```
  (i)     10 GRAPHIC 1,1
          20 CIRCLE 1,100,100,40
          30 WIDTH 2
          40 CIRCLE 1,100,100,80
```

Az (i) példa 20. sorában egy kört rajzolunk normál vonalvastagsággal. A 30. sorban beállítjuk a dupla vonalvastagságot, s utána (40. sor) egy újabb kört rajzolunk. érzékelhető a két rajzolási mód közti eltérés.

Hibalehetőségek: Ha az utasítás paramétere nem 1 vagy 2, akkor ?ILLEGAL QUANTITY ERROR hibajelzést kapunk.

WINDOW

Rövidítés: wI Token: $FE $1A (254,26)

Mód: mind parancs, mind program módban használható.

Beállítja a képernyő ablakát.

Szintaxis:
WINDOW <Xtop>,<Ytop>,<Xbottom>,<Ybootom>{,<törlés>

<Xtop>,<Ytop> az ablak bal, felső; <Xbottom>, <Ybottom> pedig a jobb, alsó koordinátái. 40 oszlopos képernyő esetén az X koordináták értékének a 0-39, 80 oszlopos képernyő esetén pedig a 0-79 intervallumba kell esnie. Az Y koordináták értékének mindkét esetben a 0-24 intervallumba kell esnie. Ha <törlés> értéke 1, akkor az utasítás egyben törli is az újonnan definiált ablakot. Ha 0, akkor nem törli. Ha a paramétert nem adjuk meg, az 0-nak számít, azaz az ablakot nem törli az interpreter.

Példák:
```
  (i)     WINDOW 0,0,12,39,1
 (ii)     WINDOW 0,40,24,79
```
Az (i) példában az ablak 40 oszlopos képernyő felső fele lesz. (ii) csak akkor értelmes, ha a 80 oszlopos képernyőt használjuk. Az ablak a képernyő jobb fele lesz.

Hibalehetőségek: Ha az adott értékekből nem képezhető ablak, akkor ?ILLEGAL QUANTITY ERROR hibajelzést kapunk.

XOR

Rövidítés: xO Token: $CE $08 (206,8)

Mód: mind parancs, mind program módban használható.

Bitenkénti kizáró logikai vagy az egész számokon.

Szintaxis: **XOR(**<arit.kif1>,<arit.kif2>**)**

Az <arit.kif1> és <arit.kif2> aritmetikai kifejezéseket egész számmá alakítja, majd ezek 16 bites alakjára bitenként elvégzi a kizáró vagy logikai műveletet. Az ennek eredményeként előálló 16 bites szám értékével tér vissza.

Példák:

```
  (i)     PRINT XOR(24,30): REM = 60
          PRINT XOR(17,17): REM = 0: Miért?!
 (ii)     100 FOR I=1 TO 255
          110 PRINT XOR(I,255-I)
          120 NEXT I
```

Az (i) példa a XOR hatását szemlélteti. (ii) mindig 255-t nyomtat. (Miért?!)

Hibalehetőségek: Ha az aritemetikai kifejezés nem konvertálható át egész számmá, akkor ?ILLEGAL QUANTITY ERROR hibajelzést kapunk.

3.4 Felhasználói programok

A Commodore 128 személyi számítógép C-128-as üzemmódjára szinte az összes olyan programot átírták, ami a Commodore 64-en üzemel. Az alapszoftver körébe tartozó programok (szövegszerkesztők, nyelvi processzorok) esetében ez általában három előnnyel is jár. Egyrészt lehetőség van a 80 oszlopos képernyő használatára, másrészt a 128 Kbyte memória sokkal komolyabb feladatok megoldására teszi alkalmassá a rendszert. Harmadrészt részben a gyorsabb lemezegység, részben a nagyobb memóriának köszönhetően sokkal 'emberibb' programok írhatók. Általában bőséges HELP funkciók segítik a rendszerek működését. Maguk a programok funkciójukat tekintve azonban nem térnek el a C-64-es változattól. Akiknek ilyen C-64-ről C-128-ra átdolgozott változatú adatkezelési programjaik vannak, azok ezek leírását az LSI ATSZ gondozásában megjelent **Easy file-tól a Master 64-ig** című kiadványban találhatják meg. Játékprogramokat a C-128-as módra ritkán írnak, hiszen a C-64-nek a lehetőségei is kimeríthetetlenek. A Commodore 128-as gép játszásra szinte kizárólag C-64-es módban használatos. Ennek ellenére a játékok legtöbbjét már C-64/128 felirattal kínálják árusításra. Ez azonban csak annyit jelent, hogy a Commodore 128-as gép C-64-es módjában is futtathatók a programok. A játokprogamokról az **1001 játék...** című kiadványban szóltunk részletesen.

Ebben a fejezetben két, kifejezetten a C-128-as üzemmódra tervezett programot ismertetünk. Az egyik egy **BASIC fordító**, a másik egy **integrált szövegszerkesztő, táblázó és kartonozó program.**

BASIC 128 fordító

A BASIC 128 a Commodore 64-es gépen megismert BASIC 64 és AUSTROSPEED fordítók C-128-as megfelelője. A BASIC interpreter valamennyi tulajdonságával rendelkezik, s azonkívül további lehetőségeket kínál. A fordítót a DATA BECKER cég forgalmazza. Védelmi okokból a lemez 36-40. sávján is van információ, ezért csak a DATA BECKER COPY nevű program birtokában lehet másolni. Felhívjuk a figyelmet, hogy az illetéktelen másolást a hazai törvények is szankcionálják!

A fordító lemeze az alábbi programokat tartalmazza:

```
139   "BASIC 128"        prg
1     "START"            prg
6     "SYMBOL"           prg
```

A BASIC 128 maga a fordító, míg a START és a SYMBOL programok segítségével a lefordított program elhelyezkedéséről, illetve az eredeti program változóinak helyéről kaphatunk információt.

A fordító egy sor utasítást gyorsabban hajt végre, mint az interpreter. Ilyenek elsősorban a feltételes és feltétel nélküli vezérlésátadások, beleértve a ciklusutasításokat és szubrutin-hívásokat. A fordító a különböző valós függvényeket is lényegesen gyorsabban számítja ki. Ezek: TAN, ATN, SIN, COS, X↑Y, EXP, LOG, SQR. Ezek végrehajtása 2-11-szer gyorsabb, mint az interpreter esetében.

A fordító elsősorban a végrehajtási időre optimalizál, s nem az elfoglalt memória nagyságra. Ennek következtében ha pl. egy Commodore 64-re megírt BASIC programot a BASIC 64 fordítóval is és a BASIC 128 fordítóval is lefordítunk, akkor az utóbbi által előállított program mérete lényegesen nagyobb lesz. Az optimalizálásnak két szintje van. A 2. szinten a fordító saját egész aritmetikát használ, ami a ciklusutasításokban is használható.

Az interpreter alaphelyzetben egy ún. **P közbenső kódra** fordít, amit a lefordított program run-time modulja értelmez. Lehetőség van közvetlenül **gépi kódra** fordítani, ami gyorsabb, de még jobban megnöveli a program terjedelmét.

A fordítás menetét a BASIC programban elhelyezett **REM @** utasításokban levő paraméterekkel lehet vezérelni.

A fordító használata

A fordító a RUN "BASIC 128" parancs kiadásával indítható el. Addig ne vegyük ki a lemezt a lemezegységből, amíg be nem jelentkezik a fordító főmenüje. A főmenü 4 választási lehetőséget kínál:

```
        BASIC 128 COMPILER V1.00

(c) 1984,1985 DATA BECKER, THOMAS HELBIG

    1 = COMPILER/OPTIMIZER I

    2 = LOAD ENVIRONMENT

    3 = ADVANCED DEVELOPMENT PACKAGE

    4 = OVERLAY
```

Ezek a funkciók sorban a következők: fordítás megkezdése, fordítási paraméterek mentése/töltése, a fordítás paramétereinek beállítása, s végül átlapolt (overlay) programok fordítása. A menükből választani a megfelelő szám, illetve betű megnyomásával lehet.

Fordítani természetesen csak letesztelt programokat érdemes. Ha ennek ellenére azt akarjuk, hogy a nyomkövetéshez szükséges információk is beépüljenek a programba, akkor ezt a megfelelő fordítási opció beállításával a fordító tudomására kell hozni.

A fordítás megkezdése

Az első menüpont kiválasztása után a fordító megkérdezi a lefordítandó program nevét, majd elkezdi a fordítást. A lemezen annyi helynek kell lenni, hogy a fordítás közben keletkező segédfile-ok is elférjenek. A fordítás befejezésekor az alábbi - PRG típusú - file-ok maradhatnak a lemezen:

"Z-PROGNEV" (*)
"S-PROGNEV" (*)
"P-PROGNEV" vagy "M-PROGNEV"

"PROGNEV" a lefordítandó file neve. "Z-PROGNEV" a program sorainak megfelelő gépi kódú programrészek kezdőcímeit tartalmazza. Ha a "Z-PROGNEV" file-t a memóriába töltjük, akkor a LIST kiadása után <sorszám>=<cím> alakú sorokat látunk. Ezek mutatják, hogy melyik sor végrehajtása melyik címen kezdődik.

"S-PROGNEV" a szimbólumtáblát tartalmazza. A két file melletti (*) jelzi, hogy ezek csak akkor készülnek el, ha a fordítási paramétereket megfelelően beállítjuk. Alapértelmezésben a fordító nem készíti el ezt a két file-t.

Az utolsó két file a lefordított programot tartalmazza. A "P-", illetve "M-" betű jelzi, hogy P-kódról vagy gépi kódról van-e szó. A fordított program a RUN "P-PROGNEV", illetve a RUN "M-PROGNEV" utasítással indítható el.

A fordító kétmenetes. Megjelenik a **PASS 1** felirat, s megindul a fordítás. A képernyőn sorra jelennek meg a lefordított sorok számai. Ha egy sorban több utasítás is van, akkor az elválasztó kettőspontokat is kiírja a fordító. Ha olyan DIM utasításhoz ér, amelyik nem konstans hosszúságú tömböt definiál, akkor megkérdezi a tömb hosszát, s azután annyi elemnek foglal helyet a memóriában.

Az első menetben hozza létre a fordító a P-kódot, és a DATA-k adatait tartalmazó "D-PROGNEV" file-t, a második menetben történik meg az optimalizálás és a végleges P- vagy gépi kód előállítása. A második menet kezdetét a **PASS 2** felirat megjelenése jelzi. A fordítás előrehaladását a rendszer éppúgy követi, mint az első menetben. A fordítás befejezése után a READY felirat jelenik meg. Ha az <N> billentyűt nyomjuk meg, visszatérhetünk a C-128-as módba. Ha bármilyen más billentyűt, a fordító újra indul.

A fordítás során hibaüzeneteket és figyelmeztetéseket kapunk. Ezek között van olyan is, ami után a fordítás abbamarad. Mások esetén a fordítás folytatódik. A hibás sorok helyén általában olyan gépi kód generálódik, ami lemerevíti a rendszert.

A fordító - a fordítási idő alatt - az alábbi hibákat képes felismerni:

SYNTAX ERROR
REDIM'D ARRAY ERROR
TYPE MISMATCH ERROR
BAD SUBSCRIPT ERROR
UNDEF'D STATEMENT ERROR
OUT OF MEMORY ERROR
RUNTIME ERROR

A hibaüzenetek értelme ugyanaz, mint a BASIC interpreter esetén. Az utolsót a BASIC interpreter nem ismeri. Ez egy olyan hiba, ami a fordításban ugyan nem okoz hibát, de futtatáskor biztosan. Ilyen pl. a 10 X=0/0 sor fordítása.

A rendszer az alábbi fi gyelmeztetéseket küldi:

TRACE NOT FOUND
A fordítást nyomkövetési módban indítottuk el, de a programban a nyomkövetés nincs bekapcsolva.

TRACE NOT USED
A programban egy TRON utasítás szerepel, de a fordítás során a nyomkövetési üzemmód nem lett bekapcsolva.

BEGIN WITHOUT BEND
Nem találja a BEGIN végét jelző BEND-et.

ILLEGAL BEGIN
Olyan helyen szerepel a programban a BEGIN, ahol ez nem megengedett.

ILLEGAL ELSE
Olyan helyen szerepel a programban az ELSE, ahol ez nem megengedett.

ILLEGAL BEND
Olyan helyen szerepel a programban a BEND, ahol ez nem megengedett.

ILLEGAL OVERLY
Átlapolt programok használata esetén valamennyi tömböt a főprogramban (azaz az elsőként fordítottban) kell deklarálni, a deklaráció máshol nem használható.

LOAD ONLY IN OVERLY
A DLOAD és LOAD utasítások csak átlapolt üzemmódban elindított fordítás közben használhatók.

POINTER WITHOUT BANK
Az interpreter a BASIC változókat az 1. szeleten tárolja, így a POINTER mindig onnan veszi az értéket. A BASIC 128 lehetővé teszi a változók 0. szeleten történő tárolását is. Ezért az első POINTER előtt ki kell adni egy BANK utasítást is.

A fordító használja a lemezegységet, ezért az adatátvitel során is keletkezhetnek hibák. Ezekről SYSTEM ERROR x üzenetet kapunk, ahol x értéke 1-9 lehet.

A fordítás vezérlése
Mint utaltunk rá a tárgynyelvi programba elhelyezett REM ___ alakú parancsokkal befolyásolhatjuk a fordítás menetét.

REM@M
A fordító ettől kezdve gépi kódot generál.

REM@P
A fordító ettől kezdve P-kódot generál.

REM@E<sorszám>
A fordító a TRAP rutin kezdőcímének a <sorszám> sort tekinti és a fordítás nyomkövetési módban folyik tovább.

REM@I=<változó>,<változó>,...
A fordító a felsorolt változókat egésznek tekinti, függetlenül attól, hogy van-e %-jel a nevük után.

REM@R=<változó>,<változó>,...
A fordító a felsorolt változókat valósnak tekinti. Csak 2-es szintű optimalizálás esetén van hatása.

REM@O<szint>
Beállítja az optimalizálás szintjét. A <szint> paraméter vagy 1, vagy 2 lehet.

REM@S<cím>
Beállítja a tárgyprogram kezdőcímét.

REM@B
A fordító - szemben az interpreterrel - a változók tárolására a 0. szeletet használja, ez ugyanis nagy mértékben meggyorsítja a program futását. A fenti utasítás hatására a tömbök az 1. szeletre kerülnek.

REM@F
A fordító FAST módban kezd el dolgozni. A 40 oszlopos képernyőt a fordító ilyenkor kikapcsolja, s ezért a fordítás menetét nem lehet nyomonkövetni.

A nyomkövetéshez szükséges információk beépítését külön lehet vezérelni. Erre a következő parancsok szolgálnak:

REM@O
Nem épül be semmi.

REM@L
A sorszámok épülnek be.

REM@C
Az utasítás sorai is beépülnek.

REM@T
A fordítás nyomkövetési módban folytatódik.

Az egybetűs parancsok kombinálhatók, pl. REM@TLC hatására a fordítás nyomkövetési módban folytatódik, s mind a sorszámok, mind az utasítások beépülnek.

A fordítás kezdeti paramétereinek a kiválasztása
A harmadik menüpont lehetőséget biztosít a fordítás indítása előtt a REM-mel is vezérelhető opciók beállítására. A képernyőn egy almenü jelenik meg:

```
ADVANCED DEVELOPMENT PACKAGE


   A = CODE-GENERATOR:     P-CODE
   B = LOAD SYMBOL-TABLE:  OFF
   C = SAVE SYMBOL-TABLE:  OFF
   E = MEMORY
   F = COMPILER:           SLOW
   G = RUNTIME-MODUL       ON
   H = EXTENTION:          OFF
   I = ENVIRONMENT
   J = TRACE:              OFF
   K = TRAP:               OFF
   L = OVERLY:             OFF
   M = DISK-COMMAND
   N = DIRECTORY
   O = OPTIMIZER:          I
```

Az egyes menüpontokat a megfelelő betű lenyomásával választhatjuk ki:

A: kód generálás: lehetőség van annak eldöntésére, hogy a gépi kódra, vagy P-kódra fordítson a program.

B: szimbólum tábla betöltése: lehetőség van a PROFIASS által, vagy másik program fordításakor előállított szimbólum tábla betöltésére. A fordítás ennek megfelelően történik.

C: szimbólum tábla elmentése: a lefordított program szimbólum tábláját tárolja a lemezen. Ezt egy másik program fordításakor, vagy a a PROFIASS futtatásakor felhasználhatjuk.

D: Z-file generálása: megadhatjuk, hogy a rendszer generálja-e a "Z-PROGNEV" file-t, ami az egyes sorok kezdetét tartalmazza, vagy sem.

E: memória felhasználása: az opció segítségével megadhatjuk a fordítónak, hogy hogyan használja a memóriát. Az opció kiválasztásakor egy újabb almenü jelenik meg. Ennek egyes pontjai a következők:

1/ A 0. szeletben a változók hol kezdődhetnek.
2/ A program és adatok tárolása hol végződhet.
3/ Az 1. szeletben felhasználható memória eleje.
4/ Az 1. szeletben felhasználható memóriarész vége.
5/ Hol kezdődhet a generált kód.
6/ A rendszer használja-e a gyors valós műveleteket. Ha igen a generált kód kb. 3 Kbyte-tal nagyobb.

F: üzemmód: lehetőség van a 2MHz-es órajel használatára, s ezzel a fordítás meggyorsítására. Ekkor a 40 oszlopos képernyőn termeszetesen nem látunk semmit.

G: run-time modul generálása: A rendszer a fordított kódhoz mindig hozzáírja a 9 Kbyte-os run-time modult. Ezt le lehet tiltani.

H: bővítések Ha BASIC bővítést használ a rendszer, ezt a paramétert be kell kapcsolni. Ekkor a fordító azokat a kódrészeket is generálja, amik a BASIC bővítés végrehajtásához szükségesek. Ha a BASIC bővítés olyan, hogy a sorszámokra is szüksége van, akkor a TRACE paramétert is ON-ra kell állítani.

I: fordítási paraméterek mentése/töltése: lehetőség van a beállított paraméterek elmentésére, vagy visszatöltésére. Az adott név elé a fordító még egy "E-" toldalékot tesz.

J: sorszám és utasítás beépítése: Bizonyos BASIC parancsok helyes végrehajtásához (pl. TRON, RESUME, NEXT, COLLISION) és a nyomkövetéshez be kell építeni a tárgykódba a program sorszámait és utásításait. Ha a J opciót beállítjuk, ez megtörténik. Ha nagyobb programról van szó, akkor inkább egy REM@ utasításba érdemes ezt beállítani.

K: hibarutin: az opció lehetőséget biztosít a TRAP rutin kezdő sorszámának megadására. A rutin akkor is beépül, ha amúgy a programban egyáltalán nem szerepelt a TRAP utasítás.

L: átlapolt programok: ugyanaz a hatása, mint a 4-es főmenüpont kiválasztásánk.

M: parancs csatorna írása/olvasása: az opció kiválasztása után elküldhetünk a lemezegységnek egy DOS parancsot.

N: lemezkatalógus: a képernyőre listázza a lemez katalógusát.

O: optimalizálás szintje: beállíthatjuk, hogy az optimalizálás 1-es, vagy 2-es szintjét használjuk-e.

Átlapolt programok fordítása

Ha programjaink a LOAD vagy DLOAD segítségével felülírják magukat, azokat overlay üzemmódban kell lefordítani. Ezt a főmenü 4-es pontjának választásával érhetjük el. Valamennyi programrészre végrehajtjuk a fordítás első menetét, majd a másodikat. A fordító egyetlen szimbólumtáblát hoz létre és csak az első programhoz írja hozzá a run-time modult.

JANE

integrált szövegszerkesztő, táblázó és kartonozó program

A JANE az **Arktronics Corporation** terméke, amelyet 1985-ben a Commodore Electronics Limited dolgoztatott át Commodore 128-ra. A program lemezről betölthető formában a Commodore 4+ gép ROM-ban levő programjához hasonló feladatokat lát el. A JANE óriási előnye, hogy kihasználja a VIC IIa chip lehetőségeit és a 1570/71-es lemezegység gyorsaságát. Szemben a Commodore 4+ géppel a JANE az egér- és ikontechnikát alkalmazza, bár a tapasztalat szerint ezeknél valamivel gyorsabb a billentyűzet használata.

A JANE eszközszükséglete

A JANE programlemezből, a segítő képernyők szövegét tartalmazó lemezből, továbbá az adatok tárolására szolgáló egyéb lemezekből áll. Az eredeti termék esetén a programlemez színe szürke, a segítő lemezé sárga. A programmal adnak még egy fekete felhasználói lemezt is. Az első két lemez katalógusa az alábbi:

```
0  "jane              " 12 2a      Programlemez
16    "janegca"           prg
3     "janevect"          prg
6     "boot"              prg
6     "fastload"          prg
149   "janecalc"          prg
2     "jane.printers"     prg
153   "janegm"            prg
50    "buildprt"          prg
1     "jane.pref"         prg
102   "janewrite"         prg
96    "janelist"          prg
39    "c.budget"          prg
29    "c.grades"          prg
5  blocks free.
```

```
0  "help             " 12 2a      Segitő lemez
10    "jane01"           prg
4     "jane02"           prg
6     "jane03"           prg
8     "jane04"           prg
6     "jane05"           prg
8     "writ01"           prg
9     "writ02"           prg
8     "writ03"           prg
11    "writ04"           prg
9     "writ05"           prg
9     "writ06"           prg
6     "writ07"           prg
6     "writ08"           prg
5     "writ09"           prg
7     "writ10"           prg
7     "writ11"           prg
11    "writ12"           prg
7     "calc01"           prg
12    "calc02"           prg
8     "calc03"           prg
8     "calc04"           prg
11    "calc05"           prg
9     "calc06"           prg
5     "calc07"           prg
4     "calc08"           prg
5     "calc09"           prg
4     "calc10"           prg
3     "calc11"           prg
11    "list01"           prg
9     "list02"           prg
4     "list03"           prg
7     "list04"           prg
4     "list05"           prg
5     "list06"           prg
9     "list07"           prg
4     "list08"           prg
7     "list09"           prg
4     "list10"           prg
394  blocks free.
```

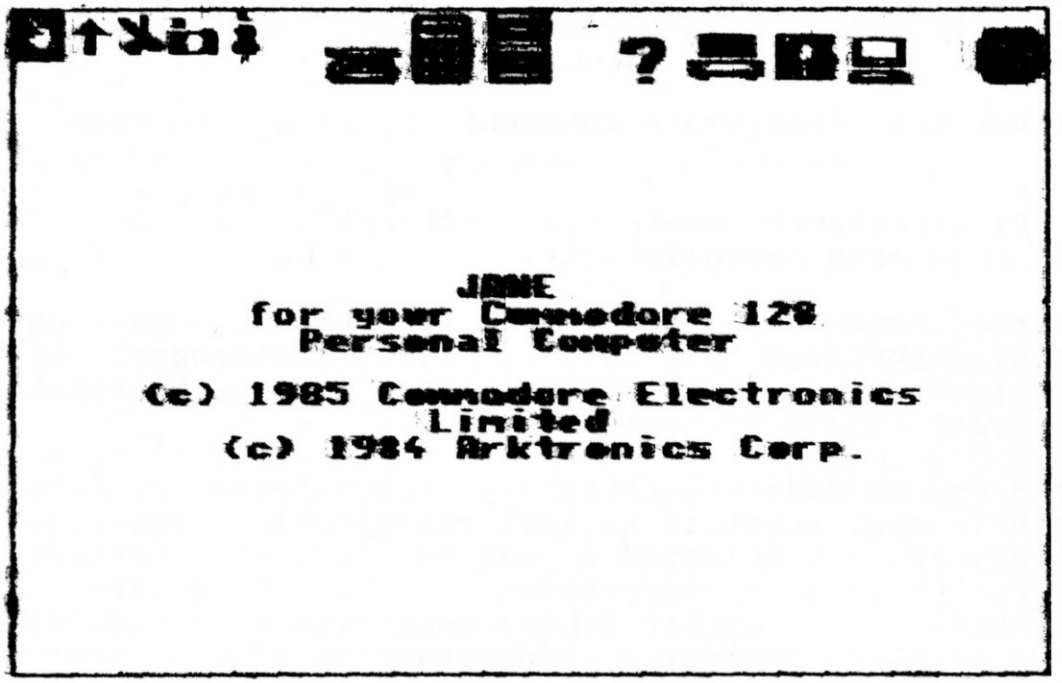
A JANE a **40 oszlopos képernyőt** használja, s nem árt ha vagy botkormányunk, vagy egerünk van. Mindkettőt a 2-es játék csatlakozóba kell bedugni.

A programlemez automatikus indítású, ezért csak teljes lemezes másolóval másolható. A segitő lemez file-jai külön-külön is másolhatók.

A JANE betöltése

A programlemez automatikus indítású. Ennek megfelelően a rendszer a következő képpen indítható el. A bekapcsolt lemezegységbe behelyezzük a lemezt. A gépet kikapcsoljuk, majd visszakapcsoljuk. Ez utóbbi helyett elég a RESET gomb megnyomása is.

Pár pillanat múlva megjelenik a képernyőn a JANE logo-ja, miközben a rendszer folyamatosan töltődik tovább. A teljes betöltés után egy majdnem üres képernyőt kapunk, amelynek a tetején különböző eszközöket látunk, s a képernyő bal felső sarkában egy kéz jelenik meg:



Az '**eszközök**' használata

A géppel való kommunikáció megkönnyítésére a képernyőn nem szöveg, hanem a programok funkciójára utaló ábrák láthatók. Ezek az **ikonok.** A végrehajtandó funkció kiválasztása nem valamilyen parancs begépelésével történik. (Bár erre is van lehetőség.) Ehelyett az egér vagy a botkormány segítségével a kezünket a megfelelő funkciót reprezentáló ikonra visszük. Elég, hogy az ikonnak és a kéznek egyetlen pontja érintkezzék! Ezután megnyomjuk a <TŰZ> gombot. Ezzel az adott funkciót kiválasztottuk.

A **kéz**, amit mozgatunk, nem mindig üres. Ha pl. ollót viszünk, akkor a funkciók módosulnak, s általában törléseket tudunk végezni. A funkció, amit a program végrehajt, így két dologtól függ: mi van a kezünkben (mi az amit mozgatni tudunk), s milyen ikonra visszük ezt rá.

A JANE bejelentkezésekor a képernyő tetején a legfontosabb eszközöket és funkciókat láthatjuk. Balról jobbra haladva ezek a következők:

Kéz: Ennek segítségével írhatunk be szavakat, számokat és képleteket, választhatjuk ki a használni kívánt funkciókat.

Ollő: Ha az ollót vesszük fel, akkor a szövegekből, adatokból részeket vághatunk ki. Ezeket azután máshová visszaragaszthatjuk, vagy a szemétbe dobhatjuk.

Fényképezőgép: A szövegek, adatok egyes részeit lefényképezhetjük, s ezeket tárolhatjuk, kinyomtathatjuk.

Ragasztós tubus: A ragasztós tubus segítségével a lefényképezett vagy kivágott szöveg és egyéb adatokat megismételhetjük, áthelyezhetjük.

Az eszközöket a választható funkciókat jelképező ikonok követik. Az

első három a JANE három program komponensére utal. Ezek a következők:

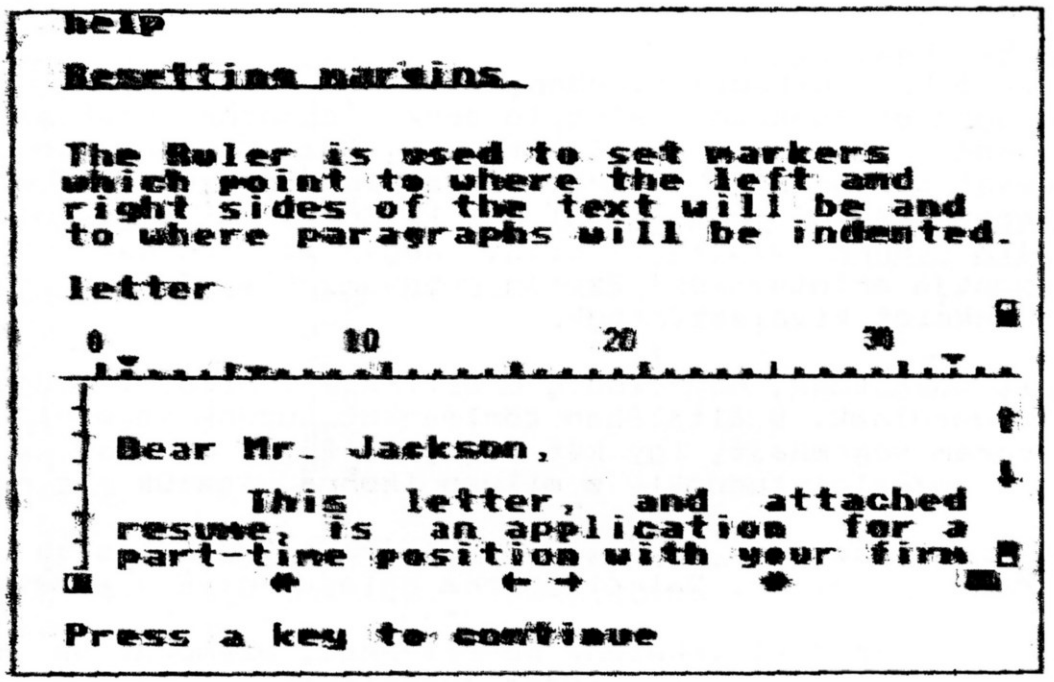
írógép: Ezt választva a JANEWRITER szövegszerkesztőt használhatjuk.

Zsebszámológép: Ezt választva a JANECALC táblázó programot indítjuk el.

Irattartó: Az irattartó szekrényt választva a JANELIST nevű kartonkezelő programot használhatjuk.

A fenti három komponenst egymástól függetlenül, de együtt is használhatjuk. Lehetőség van pl. a zsebszámológéppel előállított táblázat beillesztésére egy levélbe, majd a levél elküldésére a kartonos nyilvántartásba szereplőknek.

Rögtön ezután egy **Kérdőjel** következik. Ennek hatására az első segítő képernyő jelenik meg, angolul. Ki kell választani azt a témát, amire kiváncsiak vagyunk, s a rendszer a segítő lemezről betölti a kért információt és kiírja a képernyőre. A következő ábra egy, a szövegszerkesztés közben kapott segítő képernyőt mutat. A felső részen a magyarázat, az alsó részen a szövegszerkesztő képernyőjének egy darabja látszik. (Ez is a segítő képernyő része.)



A további három jel a legfontosabb 'háztartási' feladatokat végzi el. Ezek:

Nyomtató: már meglevő szövegek, táblázatok és kartonok nyomtatása.

Lemez: a lemez karbantartásának végzése.

Számítógép: a JANE működését befolyásoló paraméterek beállítása.

Legvégül egy **STOP táblát** látunk, ez a program, illetve valamely funkció befejezését jelenti.

Még néhány általános megjegyzés. A JANE az egyes feladatok végrehajtása során felváltva használja a program, a segítő és az adatlemezt. Ha nem a megfelelő lemezt találja a meghajtóban, akkor kéri, hogy azt helyezzük be, majd nyomjuk meg a <STOP> billentyűt vagy a botkormány <TŰZ> gombját.

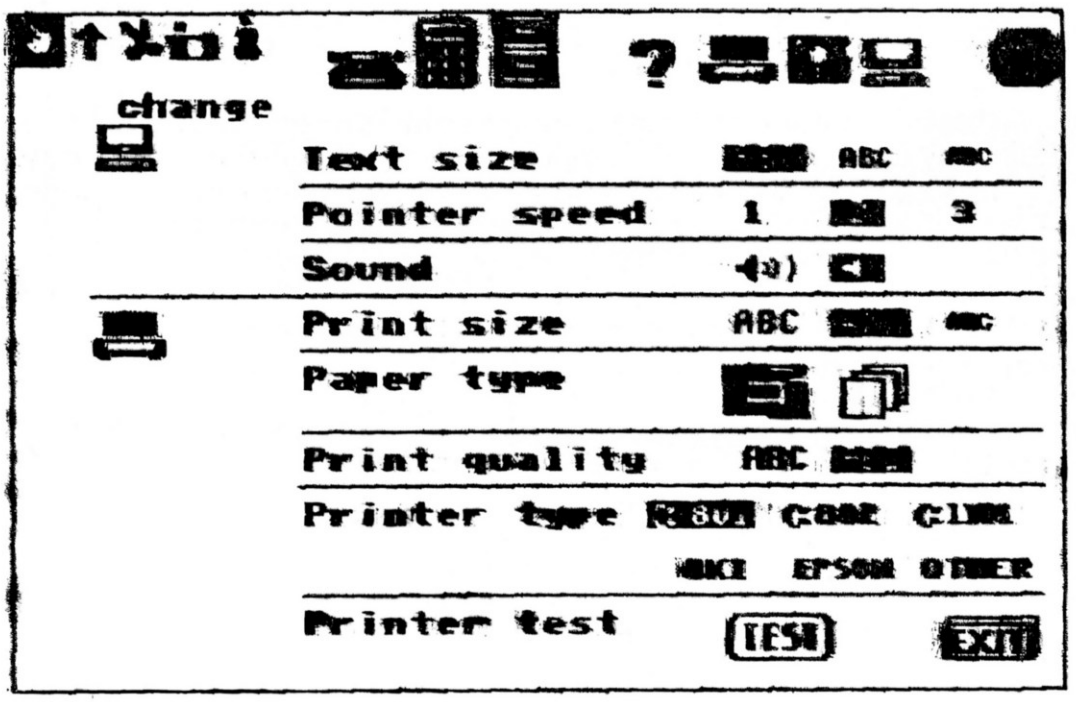
Az egyes lemezeket a színével azonosítja a rendszer:
grey (szürke) = programlemez
yellow (sárga) = segítő lemez
black (fekete) = adatlemez.

A három programkomponens használatához az adatlemezről kell betölteni az adatokat. Ilyenkor megjelenik a lemez katalógusa. A kezet a megfelelő file nevére kell pozicionálni, majd a <STOP> vagy a <TŰZ> gombot megnyomni. Ha új file-t akarunk létrehozni, akkor a NEW feliratra kell állnunk. Ezután a JANE megkérdezi az újonnan létrehozandó dokumentum, táblázat vagy kartondoboz nevét, s a lemezen létrehozza a szükséges file-t.

A JANE programjai előszeretettel használják az ablak technikát. Ha egy funkciónak paraméterei vannak, akkor pl. a szövegen egy ablak nyílik, és oda kéri a JANE a számára szükséges információt.

Sok esetben használt ikon még az **EXIT**, az **OK**, a **YES** és a **NO**. Az EXIT-et választva az illető funkció végrehajtása félbeszakad. Az OK-zás a funkció sikeres végrehajtását, illetve a lemezek cseréjének végét jelzi. Ha egy munkafolyamatot befejeztünk a JANE megkérdezi, hogy az eredményt lemezen is tárolni akarjuk-e. Ha YES-szel válaszolunk a lemezes file régi tartalma elvész, s a memóriában tárolt rész kerül oda. Ha NO-val válaszolunk, akkor a file tartalma nem változik.

A JANE működésének szabályozása
Lehetőség van a JANE bizonyos paramétereinek módosítására. Az erre szolgáló program állandóan a memóriában van, ezért bármikor végrehajthatjuk. A funkciót a felső sorban álló **számítógép** kiválasztásával érhetjük el. A képernyőt szinte teljes egészében betöltő ablak jelenik meg, a keretében a 'change' felirattal.



Az ablak jól láthatóan két részre oszlik: az elsőben a JANE paramétereit, a másodikban a nyomtatási jellemzőket állíthatjuk be. A paramétereket úgy módosíthatjuk, hogy a kijelzett értékre mozgatjuk a kezet és megnyomjuk a <TŰZ> gombot. Az egyes paraméterek jelentése a következő:

Text size: A szövegszerkesztő, táblázó és kartonozó által használt **betűméret**. Az üzenetek betűméretét ez nem befolyásolja. A választható betűméretek 40, 64 és 80 karakter/sor.

Pointer speed: A botkormánnyal vagy egérrel mozgatott eszköz **sebességét** adja meg. A három lehetséges érték közül az 1 a leglassabb, a 3 a leggyorsabb.

Sound: a JANE **hangjelzése** ki-, illetve bekapcsolható.

Print size: A kinyomtatott szövegrész **betűnagyságát** határozza meg. A lehetséges értékek: 10,12 és 15 karakter/inch.

Paper type: Lehetőség van perforált szélű leporelló, vagy különálló papírlapok használatára.

Print quality: JANE kétfajta betűtípust használ. Az egyik szebb, de lassabban nyomtatható, a másik kevésbé szép, de relatíve gyorsan lehet kinyomtatni.

Printer type: A felsorolt nyomtatók közül választhatunk.

Printer test: A funkció kiválasztásakor a JANE leteszteli a nyomtatót. Olyankor érdemes használni, ha nyomtatónk típusa nincs a feltüntetettek közt, de reméljük, hogy kompatibilis valamelyikkel.

Az EXIT kiválasztásával visszatérhetünk az előző funkcióra.

A lemezek karbantartása

A lemez ikonjának választásával a lemez karbantartó funkciót választhatjuk. A képernyő jobb oldalán nyílik egy ablak, a keretben a 'directory' felirattal. Az ablak bal szélén további ikonokat találunk. Ezek közül az EXIT jelentése a szokásos: visszatér az előző funkcióba.

Az első ikon **két egyforma cimkét** ábrázol. Ennek segítségével tudjuk a lemezen levő file-oknak - ugyanazon a lemezen - egy másolatát létrehozni. A másolás választásakor JANE megkérdezi a másolandó file nevét és a másolat nevét, majd végrehajtja a funkciót.

A második ikon **két cimkét ábrázol közte egy nyíllal.** Ennek segítségével nevezhetjük át a file-okat. JANE megkérdezi a file régi és új nevét, majd végrehajtja az átnevezést.

A **szemétkosár** file-ok törlésére szolgál. A file nevének megadása után a JANE letörli a lemezről a file-t.

Az utolsó ikon egy **NEW feliratú lemez.** Ennek segítségével formázhatunk meg új, még nem használt adatlemezeket. JANE kéri a lemez nevét és azonosítóját, majd megformázza a lemezt. Ha az azonosító helyét üresen hagytuk, akkor csak törli a lemezt.

Áttérés más programrészre
Valahányszor a szövegszerkesztő, a táblázó és a kartonozó program közt váltani akarunk, az adott funkció végrehajtását a **STOP tábla** választásaval be kell fejezni. Ugyanez a helyzet, ha pl. egy másik dokumentumot akarunk továbbszerkeszteni. A STOP tábla kiválasztása után a program megkérdezi, hogy a jelenlegi memória tartalmat visszaírja-e a lemezre. Ha igennel (YES-szel) válaszolunk, akkor ez megtörténik. Ezután van csak lehetőség új komponens választására.

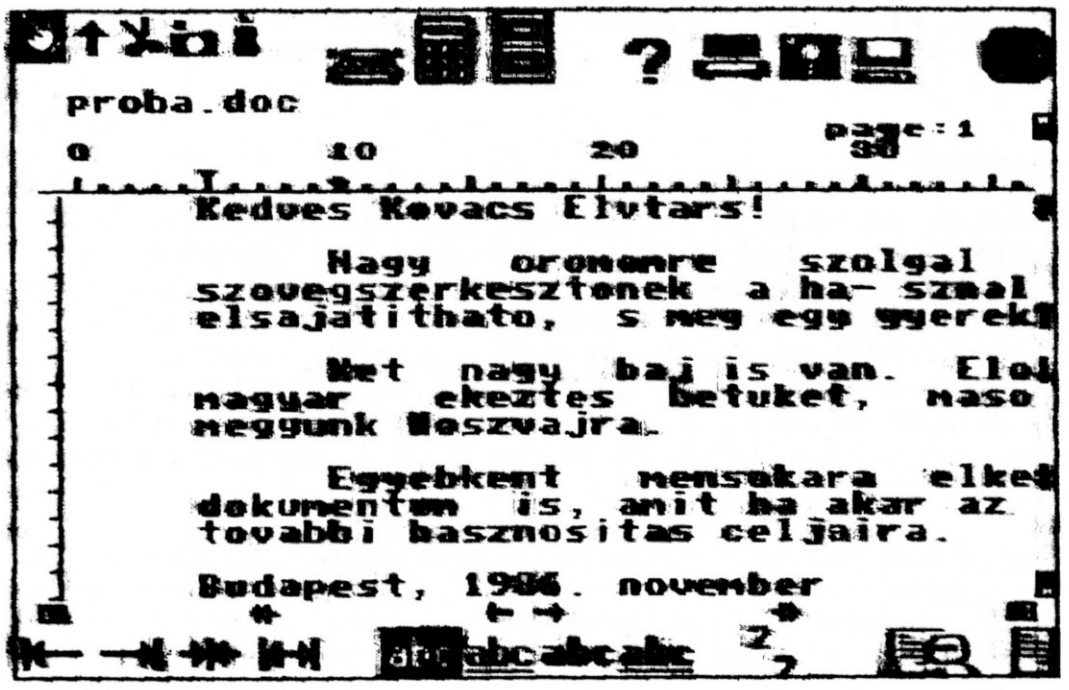
A JANE-ből nincs lehetőség a C-128-ba való visszatérésre. Ehhez a gépet ki kell kapcsolni, majd pedig újra bekapcsolni.

JANEWRITER

A JANEWRITER elindítása
Az írógép kiválasztásával a JANEWRITER szövegszerkesztő programját indítjuk el. A program bekéri az adatlemezt, majd kiválsztatja velünk, hogy melyik dokumentumot akarjuk szerkeszteni. Ha új dokumentumot készítünk, akkor a NEW ikonra kell pozicionálnunk. Előfordulhat, hogy már annyi dokumentumot írtunk, hogy valamennyi nem fér egyszerre a képernyőre. Ekkor a keret szélén levő nyilak segítségével mozgathatjuk a kijelzett neveket.

Ezután a rendszer kéri a programlemezt, s betölti a megfelelő programrészeket. A töltés ideje alatt egy **óra** látszik a képernyőn. Ezzel jelzi a rendszer, hogy dolgozik.

A programrendszer betöltése után a JANE újból kéri az adatlemezt, s betölti róla a kért dokumentumot. A képernyőn a JANEWRITE képernyő jelenik meg. Felül továbbra is láthatók a főmenü ikonjai, alul pedig további ikonok jelennek meg, amelyek a szövegszerkesztő működését vezérlik:



A középső rész - keretbe foglalva - az a terület, ahová írhatunk. A keret bal felső sarkában a dokumentum neve látható, míg a keret jobb és alsó részén nyilak és apró téglalapok láthatók. Ezek maguk is ikonok, amelyek segítségével a képernyő bekeretezett részét a szövegen mozgatni lehet.

A keret felső sorában egy skála látható, ami azt mutatja, hogy hányadik karakterpozícióba írunk. Két magasabb és egy alacsonyabb háromszög alakú jel a jobb és baloldali margót, illetve a bekezdés helyét jelöli. (A képernyőn egyszerre 40 ikon látható, ami 40 funkciót jelöl!)

A képernyő mozgatása
A képernyőt a szövegen a kereten levő nyilakkal mozgathatjuk. A szimpla nyilak segítségével 10-10 karakterrel mozgatjuk a szöveget, a dupla nyilakkal egy-egy képernyőnyit. A keret szélén levő téglalapok a szöveg jobb, bal, alsó, illetve felső szélere helyezik az ablakot.

A margók beállítása
A keret első sorában látható kis háromszögeket ugyancsak a kéz segítségével állíthatjuk át. Pozícionáljuk a kezet pl. a bal margóra, majd **nyomjuk meg a <TŰZ> billentyűt és tartsuk lenyomva**! A botkormány jobbra-balra mozgatásával a bal margót bárhová beállíthatjuk. Amikor a helyére vittük, akkor engedhetjük csak el a <TŰZ> gombot. Hasonlóan állíthajtuk be a jobb oldali margót, illetve a bekezdés helyét is.

Szövegírás
A kezet állítsuk arra a helyre, ahová írni szeretnénk. Kezdjünk el írni. A **kéz** a kurzorhoz hasonlóan kezd el mozogni. Egyetlen különbség: ha egy szó nem fér már ki a sorba, akkor a szövegszerkesztő automatikusan átviszi azt a következőbe.

Lehetőség van a betű típusának megválasztására is. Tetszés szerint írhatunk **normál, aláhúzott, vastagított, vastagított és aláhúzott** betűket. Lehetőség van **alsó és felső indexek** használatára. A használt betűtípust az alsó sorban középen látható 6 ikon közül a megfelelő kiválasztásával érhetjük el. A kiválasztott ikonnak megfelelő betűtípus kerül a szövegbe.

Törlés és másolás
Egy-egy karakter törlésére a <DEL> billentyűt használhatjuk. Nagyobb szövegrészeket az **olló** segítségével törölhetünk. Vegyük kezünkbe az ollót, s vigyük az első törlendő karakterre. Nyomjuk meg és **tartsuk lenyomva a <TŰZ> billentyűt**, s a botkormányt mozgassuk az utolsó törlendő karakterre. Azok a területrészek, amelyeken az olló áthalad **inverz alakúra változnak.** A törlés azonnal megtörténik, mihelyt a <TŰZ> gombot elengedjük. Ha mégsem akarunk törölni, az ollót a kezdő pozícióra vigyük vissza, s ott engedjük el a <TŰZ> billentyűt. A törölt részt - a következő törlésig - a ragasztós tubussal bárhová visszaragaszthatjuk!

Több karaktert a **felfelé nyíl**-lal szúrhatunk be. Cseréljük át a kezet a nyílra. Amit ezután beírunk a szövegbe, az nem felülírja a karaktereket, hanem a karakterek úgy íródnak a szövegbe, hogy a többi karakter egy hellyel jobbra tolódik. Ha már nem akarunk több karaktert beszúrni, akkor a felfelé nyilat cseréljük vissza a kézre.

A **fényképező gépet** szövegrészek megismétlésére használhatjuk. Első lépésben ez úgy történhet, hogy kezünkbe vesszük a fényképező gépet és

- hasonlóan ahhoz, ahogy a szöveget töröltük - beállítjuk a másolandó részt. A különség annyi, hogy a <TŰZ> billentyű elengedése után az inverz alakú rész visszaváltozik, ahelyett, hogy törlődne. Az így lefényképezett részt azután bárhová visszaragaszthatjuk. Ehhez felvesszük a **ragasztós tubust** és arra helyre visszük, ahová a lefényképezett rész bal-felső sarkát helyezzük majd. Ebben a helyzetben nyomjuk meg a <TŰZ> billentyűt. Megjelenik az óra, jelezve, hogy a JANE dolgozik. A másolás ezután megtörténik. Ha a másolás során valamit felül kellene írni, a másolás nem történik meg. Ha mégis szeretnénk másolni, akkor előbb törölni kell azokat a részeket. A **ragasztós tubust** a különböző programkomponensek közt is használhatjuk. Ha pl. a táblázó program használata közben lefényképeztük annak egy részét, azt a program cseréje után visszaragaszthatjuk, visszamásolhatjuk a szövegszerkesztő területére. Hasonlóan járhatunk el a kartonozóval is.

Szöveg formázása

A szövegszerkesztő segítségével tetszés szerint írhatunk be szövegrészeket a memóriába. Általában akkor tekintenek szépnek egy szöveget, ha bekezdésekben, a cimekkel esetleg középen 'formásan' jelenik meg. Az alsó sor jobb oldalán levő, nyílszerű ikonok erre szolgálnak. Az egyes jelek a következő formára hozzák a szöveget:

```
|  ******* ***** **          | formázás előtt
|***** ****          *****   |
|      *******   *           |
| ****** ******              |

|     ******* ***** **       | középre igazítás
|     ***** **** *****       |      ->|<-
|       *******   *          |
|      ****** ******         |

|           ******* ***** **| jobbra igazítás
|   ***** ****          *****|      -->|
|                ******* *  |
|              ****** ******|

|******* ***** **            | balra igazítás
|***** ****          *****   |      |<--
|*******   *                 |
|****** ******               |

|*******       *****      **| szélre igazítás
|*****      ****       *****|      |<-->|
|*******                   *|
|****** ******               |
```

A szöveget a következő eljárással tudjuk megformázni: A kézzel kiválasztjuk azt az ikont, aminek megfelelő módon akarjuk a szöveget megformázni. Ezután a kezet az első megformázandó sorra visszük, s megnyomjuk a <TŰZ> billentyűt. A teljes sor inverz alakúra változik. A <TŰZ> billentyű lenyomva tartása közben lefelé mozgatjuk a kezet. Mikor az utolsó megformázandó sorra értünk, elengedjük a billentyűt. A JANE a kiválasztott sorokat a kijelzésnek megfelelően formázza meg.

A fenti eljárással a szöveg különböző részeit más- és másféleképpen formázhatjuk meg.

Szöveg keresése és cseréje

A JANE lehetőséget biztosít adott karaktersorozatok megkeresésére és cseréjére. Ehhez a bal alsó sarok melletti **nagyító üveget** kell kiválasztanunk. Ezután egy ablak nyílik a képernyőn, s a program annak a karaktersorozatnak a beírását várja, amit meg akarunk keresni. Ezután kell beírni azt a szövegrészt, amire a megtalált karaktersorozatokat át akajuk cserélni. Ha a szövegrész beírásánál hibát vétünk, azt a kéz visszaállításával javíthatjuk. Használhatjuk a <DEL> billentyűt is. A keresendő szövegrésznél érdektelen, hogy milyen betűtípussal írtuk be, s hogy nagy vagy kis betűkből áll. A szöveg beírása után az OK-t válasszuk ki. Ha a 'Replace with' részbe nem írunk semmit, akkor csak keresés történik, de a megtalált karaktersorozatok változatlanul megmaradnak a szövegben.

Nyomtatás előkészítése

A jobb alsó sarokban álló **papírlap** ikon kiválasztása után egy szinte teljes képernyős ablakot kapunk, ahol beállíthatjuk a **nyomtatandó lap formátumát.** Az első sorban a lap fejlécének szövegét be kell írni. Az egyes paraméterek jelentése a következő:

Title= fejléc szöveg. A mellette levő ikonokkal lehet kiválasztani, hogy ez bal szélen, középen vagy jobb szélen jelenjen meg.

Page number= kezdő sorszám. A sorszámot be kell írni. A mellette levő ikonokkal lehet kiválasztani, hogy ez bal szélen, középen vagy jobb szélen jelenjen meg. A közvetlenül a szám mellett látható két ikonnal lehet kiválasztani, hogy a lapszám a lap tetején vagy alján jelenjen meg.

A következő értékeket a + és - ikont választva lehet növelni, illetve csökkenteni. Az egyes értékek a következőket jelentik:

Print margin = a nyomtatás hányadik karakterhelyen kezdődjék.

Line spacing = két sor között hány soremelés legyen.

Top margin = Lap tetején hány üres sort hagyjon.

Bottom margin = Hány üres sor maradjon a lap alján.

Page length = Az egy lapra írható sorok száma. JANE 66-ra állítja be, de ez nem jó. Általában 12 inch-es leporellót használunk, ezért ezt az értéket 72-re kell átállítani.

Printing at page = Hányadik lapnál kezdődjék a nyomtatás. értéke általában 1, de előfordulhat, hogy csak az 5. laptól akarjuk a szöveget kiíratni.

Number of copies = Példányszám.

A szöveg nyomtatása

Miután a szöveget beírtuk, megformáztuk, beállítottuk a nyomtatási paramétereket, kinyomtathatjuk a szöveget. Ehhez a felső sorban álló nyomtató ikonját kell csak választanunk, s a JANE azonnal kinyomtatja a szöveget.

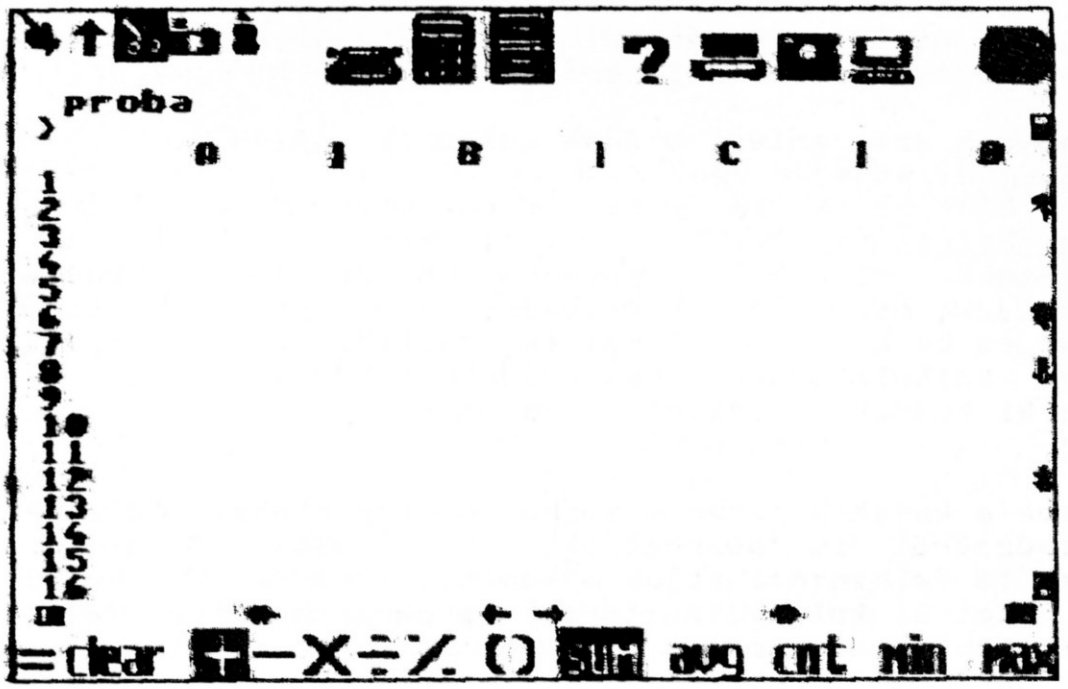
Kilépés a JANEWRITER-ből

Ha a szöveg módosítását, nyomtatását befejeztük, s egyéb feladatot akarunk végezni, akkor a **STOP tábla** kiválasztásával kell kilépnünk a szövegszerkesztőből. Rögtön ezután egy ablak nyílik a képernyőn, s a JANE megkérdezi, hogy el akarjuk-e menteni a memóriában levő szöveget a lemezre. Igen (YES) válasz esetén a memória tartalmát a program visszaírja a lemezre.

JANECALC

A JANECALC táblázó programot a **zsebszámológép** választásával érhetjük el. Hasonlóan történik a program betöltése, mint a JANEWRITE esetén: először az adatlemezt kéri, majd pedig a katalógus kiírása után azt, hogy válasszuk ki, melyik táblázattal szeretnénk dolgozni. Ha új táblázatot akarunk generálni, akkor a NEW ikont kell választanunk. Ezután a program betöltődik, s a rendszer beolvassa a kiválasztott táblázatot.

A JANEWRITE-nál megszokott képernyőformátum jelenik meg, az alsó sorban azonban egészen másfajta ikonok láthatók, mint ott:



Az ablak keretében most is a táblázat nevét látjuk, de alatta most egy > jellel kezdődő elkülönített sor látszik. Ezt parancs sornak hívjuk és formulák beírására használjuk.

A keretben vízszintes és függőleges sorszámozást látunk: a vízszintes az ABC betűivel, a függőleges számokkal történik. A betűk a táblázat egy-egy oszlopát, a számok egy-egy sorát jelentik. A táblázat elemeit a megfelelő oszlopra és sorra való hivatkozással jelöljük. Pl. C3 a C oszlop 3. sorában áll. A táblázat elemeit **celláknak** is hívjuk.

A táblázat az 1-50 sorokból és az A-Y oszlopokból áll. A képernyőt megint csak a **keret jobb, illetve alsó részén levő nyilak segítségével tudjuk mozgatni.**

Egy cella számot, címkét (szöveget) és képletet tartalmazhat. A képlet a táblázat bármely más cellájára, vagy celláira is utalhat. **Ha valamelyik érintett cella tartalma megváltozik, a cella értékét a rendszer automatikusan újraszámítja!**

Szöveg beírása

Címkék - azaz szövegek - beírása egyszerű. A kezet arra a cellára kell állítani, ahová írni szeretnénk. Ezután egyszerűen begépelhetjük a szöveget. Ha hibáztunk, a cella tartalmát a <DEL> billentyűvel javíthatjuk. Előfordulhat, hogy a szöveg számot is tartalmaz. Ekkor a szöveg beírása előtt meg kell nyomni a <CTRL-T> billentyűt. A számot a program ilyenkor karaktersorozatként tárolja. A cella tartalmát bármikor újraírhatjuk. A cimkéket általában a táblázat fejléceinek megadására használjuk.

Szám beírása

A szám beírása hasonlóan kezdődik, mint a szövegeké: a kezet a táblázat megfelelő helyére állítjuk és elkezdjük a számot írni. Az első számjegy leírása után a cella inverz alakú lesz, jelezve, hogy szám beírása kezdődött. A JANE ellenőrzi a szám alakját. A szám a keret tetején levő parancs sorban is megjelenik. Ha hibáztunk, azt a <DEL> billentyű segítségével javíthatjuk ki. A szám beírását a <RETURN> megnyomásával kell befejezni.

A legnagyobb szám amivel a JANE dolgozik 21474836.

Képletek beírása

A képleteket először a parancs sorba kell begépelni. Ezután eldönthetjük, hogy csak az értékére vagyunk-e kiváncsiak, vagy magát a képletet is be akarjuk valamelyik cellába írni. Az előbbi esetben egyszerű kalkulátorként használjuk a JANECALC-ot, míg a második esetben ki tudjuk használni az automatikus újraszámítási lehetőséget is.

Mozgassuk a kezet a parancs sorba. A képleteket teljes egészében a billentyűzetről is beírhatjuk, de a képernyő alsó sorában álló ikonokat is felhasználhatjuk a képlet írására. Ebben az esetben az illető jelet ki kell választani, s a parancs sorba felvinni. Ha egy cella fölött nyomjuk meg a <TŰZ> gombot, annak hatására a cella neve kerül a parancs sorba.

Az alsó sorban látható ikonok egyben azt is mutatják, milyen műveleteket végezhetünk. Balról jobbra haladva az egyes ikonok jelentése a következő:

összeadás jele = számok összeadása;
kivonás jele = az első számból kivonja a másodikat;
szorzás jele = számok szorzása;
osztás jele = az első számot elosztja a másodikkal;
százalék(%) jel = az első szám második számnyi százalékát számítja ki;
kezdő zárójel = először a zárójelben levő értéket számítja ki;
záró zárójel = a kezdő zárójel párja.

SUM = egy téglalapban álló elemek összegét számítja ki, pl. SUM(C3:C6), SUM(C5:F5), SUM(A1:B5). A SUM-ban kettősponttal elválasztva kell megadni a téglalap bal felső, illetve jobb alsó koordinátáit.

AVG = egy téglalapban levő elemek átlagát számítja ki. A téglalapot a SUM-hoz hasonlóan kell megadni. Pl. AVG(A1:B5).

CNT = megszámolja, hogy egy téglalap alakú részben hány szám van. A téglalapot ugyanúgy kell megadni, mint a SUM esetén: CNT(A1:B5), CNT(C1:C5).

MIN = egy téglalap alakú részben levő legkisebb szám értékét adja. A téglalapot ugyanúgy kell megadni, mint a SUM-ban.

MAX = egy téglalap alakú részben levő legnagyobb szám értékét adja. A téglalapot ugyanúgy kell megadni, mint a SUM-ban.

A képlet beírásának végén nyomjuk meg a **billentyűzeten az =** jelet. A rendszer a parancs sorba rögtön kiírja a képlet értékét.

Ha ezt a képletet valamelyik cellában tárolni akarjuk, akkor válasszuk ki az alsó sorban levő **ikonok közül az egyenlőségjelet**, s vigyük arra a cellára, ahová a képletet másolni akarjuk. Végül nyomjuk meg a <TŰZ> gombot. A cellában a képlet értéke jelenik meg, de **vastagított** formában. Ez jelzi, hogy nem számot látunk, hanem a cellába beírt kifejezés, képlet értékét.

Előfordulhat, hogy a későbbiekben kíváncsiak vagyunk arra, hogy milyen képlet szerepel abban a cellában. Válasszuk ki a **felfelé mutató** nyilat, vigyük a cella fölé, s nyomjuk meg a <TŰZ> gombot. A képlet megjelenik a parancs sorban, s akár módosítható is. Ebben az esetben a felfelé mutató nyilnak semmilyen 'beszúrási' funkciója sincs.

A táblázat részeinek törlése és másolása

A táblázat egy cellájának összesen négy értéke lehet:

nincs benne semmi;
szöveg van benne;
szám van benne;
képlet van benne.

Ha az első állapotot akarjuk elérni, akkor az alsó sorban levő ikonok közül a **CLEAR**-t az adott cellára kell vinni, s úgy megnyomni a <TŰZ> gombot.

Törölni teljes sorokat és oszlopokat, továbbá a táblázat téglalap alakú részeit lehet. Hasonlóan teljes sorok és oszlopok szúrhatók be, illetve másolhatók.

Sor vagy oszlop beszúrása

Vegyük kézbe a felfelé mutató nyilat s vigyük a sort (oszlopot) jelképező számra, (betűre). Ezután nyomjuk meg a <TŰZ> gombot. Az adott sortól (oszloptól) kezdve minden sor (oszlop) egy hellyel lejjebb (jobbra) csúszik, s így egy üres sor (oszlop) keltkezik.

Sor vagy oszlop törlése

Vegyük kézbe az ollót s vigyük a sort, illetve oszlopot jelképező

számra, illetve betűre. Ezután nyomjuk meg a <TŰZ> gombot. Az adott sortól (oszloptól) kezdve minden sor (oszlop) egy hellyel feljebb (balra) csúszik, s az adott sor (oszlop) törlődik.

Sor vagy oszlop áthelyezése
Az előző módon törölt sort (oszlopot) akármelyik sorra (oszlopba) átmásolhatjuk. Ehhez az ollót a ragasztós tubusra cseréljük, s a ragasztót a megfelelő sorra (oszlopra) visszük. A <TŰZ> gomb megnyomására a kivágott sor (oszlop) az adott sorba (oszlopba) másolódik.

Téglalap törlése és másolása
A fenti műveletek a sort, illetve oszlopot fizikai értelemben is törölték. Lehetőség van azonban arra, hogy csak a cellák tartalmát töröljük, ne magát a cellát. Ehhez vegyük kézbe az ollót, s vigyük a törlendő téglalap bal felső sarkába. Nyomjuk meg a <TŰZ> gombot és tartsuk lenyomva! A botkormánnyal az ollót a törlendő terület jobb, alsó sarkára visszük, s akkor elengedjük a <TŰZ> gombot.

Az utoljára törölt területet a ragasztós tubussal bárhová visszaragaszthatjuk.

Ha a táblázat egy részét meg akarjuk ismételni valahol máshol, akkor le kell fényképeznünk, majd a fényképet valahol visszaragasztanunk. Ehhez vegyük kezünkbe az fényképező gépet. Ha teljes sort (oszlopot) akarunk lefényképezni, akkor a sort (oszlopot) jelképező számra (betűre) kell a fényképező gépet állítani, ha téglalapot, akkor a téglalap jobb, felső sarkára. Nyomjuk le és tartsuk lenyomva a <TŰZ> billentyűt, majd a kurzor mozgatásával állítsuk be a lefényképezendő területet. A <TŰZ> billentyű felengedésével a fényképezés azonnal megtörténik.

A lefényképezett terület bárhová visszamásolható. Ezzel ugyanaz a rész kétszer jelenik meg a táblázatban. Vegyük kezünkbe a ragasztós tubust és mozgassuk a megfelelő sorra, oszlopra vagy cellára. Ezután nyomjuk meg a <TŰZ> gombot. A duplikálás azonnal megtörténik.

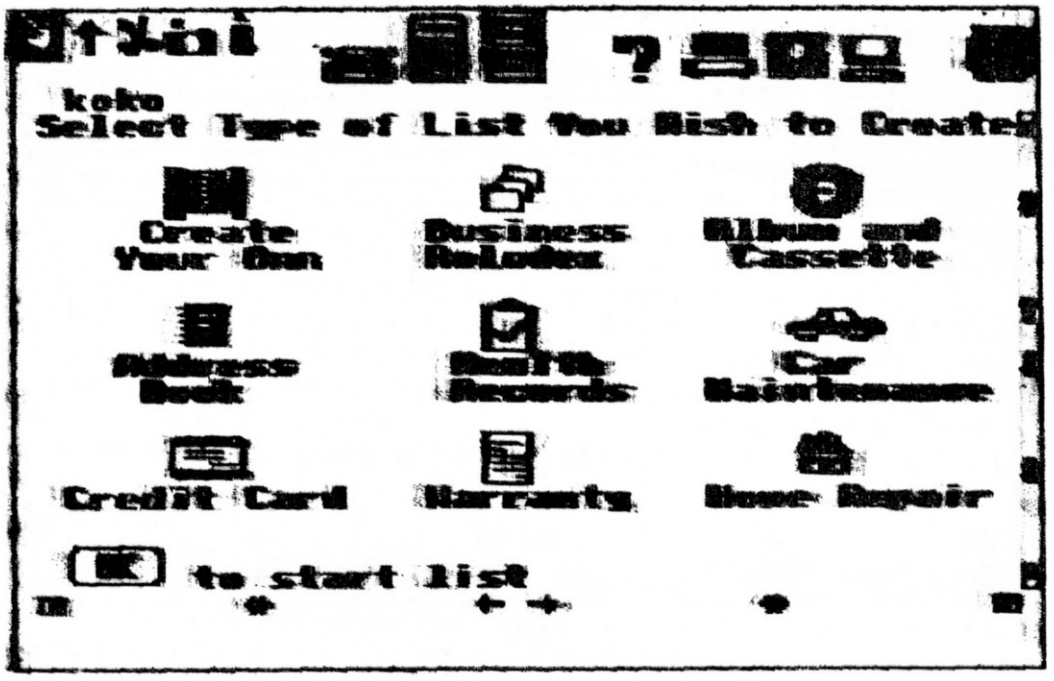
A lefényképezett részt a szövegszerkesztőbe is visszamásolhatjuk. Ehhez ki kell lépni a JANECALC-ból, majd elindítani a szövegszerkesztőt, s ott használni a ragasztót.

A táblázat kinyomtatása
A táblázat közvetlenül nem íratható ki. Ehhez a kinyomtatni kívánt részt először a fentebb vázolt módon át kell vinni a szövegszerkesztőbe, s a szövegszerkesztő segítségével kell kinyomtatni.

JANELIST

A JANELIST táblázó programot a **kartotékdoboz** választásával érhetjük el. Hasonlóan történik a program betöltése, mint a JANEWRITE esetén: először az adatlemezt kéri, majd pedig a katalógus kiírása után azt, hogy válasszuk ki, melyik kartotékdobozzal szeretnénk dolgozni. Ha új dobozt akarunk generálni, akkor a NEW ikont kell választanunk. Ezután a program betöltődik, s a rendszer beolvassa a kiválasztott kartotékdoboz tartalmát.

Mielőtt azonban ez megtörténne a JANE megkérdezi, hogy a kartotékon milyen adatok szerepelnek:



Ezeket magunk is megadhatjuk, azonban a JANE 8 fajta kartotéktípust előre kínál. Ezek a következő feladatokat oldják meg:

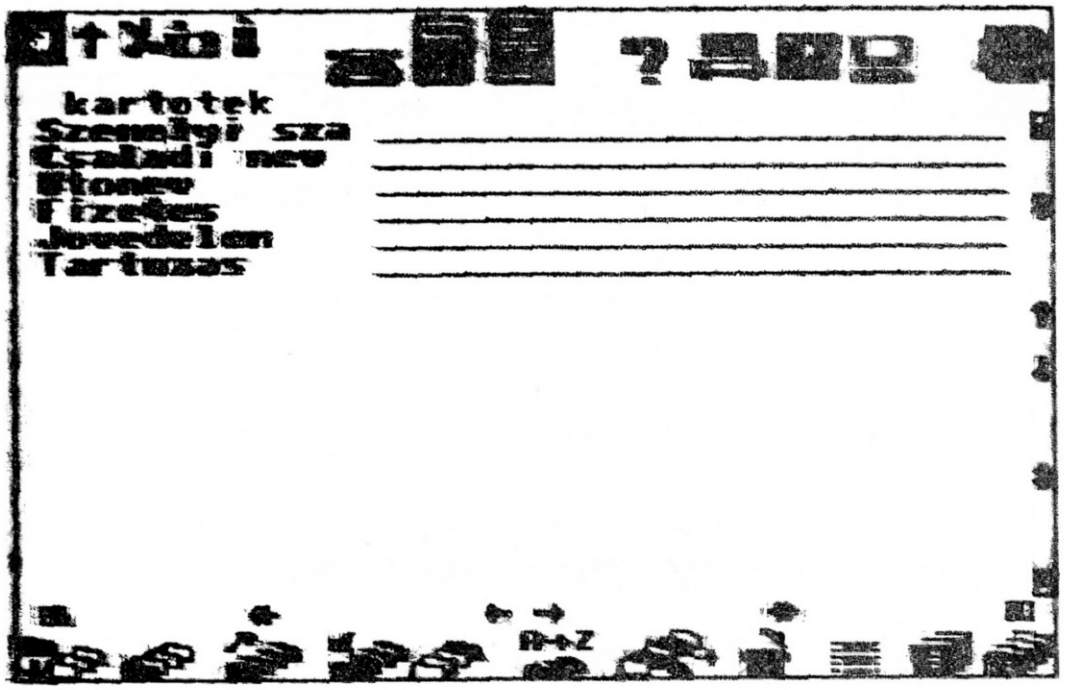
1/ lemez és kazetta nyilvántartás;
2/ hitelkártya nyilvántartás;
3/ egészségügyi adatok nyilvántartása;
4/ garancia jegyzék;
5/ telefonkönyv;
6/ üzleti adatok;
7/ kocsi karbantartási adatok;
8/ háztartás vezetéséhez szükséges adatok.

Meg kell adni, hogy saját definíciójú kartotékokat használunk, vagy valamelyik előre definiált típust. Ennek ismeretében tudja a rendszer a kartotékdoboz tartalmát betölteni.

Ha új kartotékdobozt definiáltunk és saját magunk adjuk meg a kartoték típusát, akkor a JANE egy ablakban kéri a kartotékon levő adatok megnevezését. Összesen 15 adatot tarthatunk nyilván a kartotékon, az egyes adatok megnevezése maximum 12 karakterből állhat.

Az egyes megnevezéseket úgy írhatjuk be, mint a JANEWRITE-nál, a kezet kell a megfelelő sorba vinni. Ha hibázunk, a <DEL> billentyűt használhatjuk törlésre. A megnevezések beírása után az OK funkciót kell kiválasztani.

Ezután a JANEWRITE-nál megszokott képernyőformátum jelenik meg, az alsó sorban azonban egészen másfajta ikonok láthatók, mint ott:



Ezek segítségével végezhetjük el a kartotékok karbantartását. Az ablak keretében most is a kartotékdoboz nevét látjuk. A keretben az első kartoték tartalma látszik, ami üres, ha új kartotékdobozt hoztunk létre.

A képernyőn látható kartoték tartalmát bármikor tetszés szerint módosíthatjuk. A kezet a megfelelő helyre mozgatva azonnal beírhatjuk az adatokat. Hiba esetén a <DEL> billentyűt használhatjuk törlésre. A beírás pillanatában módosul a szóbanforgó kartoték tartalma.

A képernyő alján látható ikonok segítségével tudunk a kartotékok közt lapozni, modósítani, válogatni, kartotékot megsemmisíteni.

Lapozás a kartotékok közt

Balról a második ikonon **a kartotéksorozat első kartotékját** ábrázolja. A hozzá tartozó funkció az első kartoték tartalmát írja ki a keretbe.

A mellette levő ikonon egy **kartotéksorozat előre mutató nyíllal** látható. Ennek hatására mindig a következő kartoték tartalma kerül a képernyőre.

Balról a negyedik ikonon egy **kartotéksorozat hátra mutató nyíllal** látható. Ennek hatására mindig az előző kartoték tartalma kerül a képernyőre.

Az ötödik ikonon egy **utolsó helyre rakott kartotékot** ábrázol. A hozzá tartozó funkció az utolsó kartoték tartalmát írja ki a keretbe.

Kartoték törlése

Mindig a képernyőn éppen látható kartoték törlésére van lehetőség. Ki kell választani a **szemétkosár** ikont, s a JANE rögtön törli a kartotékot.

Kartoték hozzáírása
További kartotékok felvétele a **kartoték hozzátétele** - balról az első - ikon kiválasztásával történik. A képernyőn egy üres kartoték jelenik meg, amit ki lehet tölteni. Az újonnan hozzáírt kartoték - az első rendezésig - az utolsó lesz.

Kartotékok rendezése
Néha szükség lehet a kartokékok rendberakására, valamilyen szempont szerinti rendezésére. Erre az **A-Z-s kártyasorozatot** ábrázoló ikon szolgál. Kiválasztása után a JANE kilistázza az adatok megnevezését. Ezek közül kell a rendezési szempontokat kiválogatni. Amelyik adat megnevezése előbbre kerül, azt a rendezés során előbb hasonlítja össze a rendszer. Ha pl. az első rendezési szempont a név, a második a fizetés, akkor az Agárdi János akkor is meg fogja előzni a Nagy Antalt, ha az utóbbi 5000 forinttal többet keres. Ha meggyőződtünk róla, hogy a rendezési szempontokat jól választottuk meg OK-zzuk le. Megjelenik az óra, utalva arra, hogy a program hosszabb ideig is dolgozhat, s a JANE átrendezi a kartotékokat.

Kartotékok keresése
Előfordulhat, hogy szükségünk van pl. az összes Nagy nevű ismerősünk adatait tartalmazó kartotékra. Ebben az esetben a **megjelölt kártyákat** ábrázoló ikont (jobbról az ötödik) kell választanunk. Rögtön ezután a JANE egy ablakot nyit, s a keresési szempontokat kéri. Azoknál az adatoknál kell csak valamit beírni, ahol válogatni akarunk. Ezután két lehetőség közül választhatunk. Vagy OK-zzuk a **'to select card(s)'** üzenetet, vagy OK-zzuk a **'to select all cards'** üzenetet. Az első esetben a keresés csak az első 'jó' (azaz a keresési szempontoknak megfelelő) kartotékig tart, míg a második esetben az összes 'jó' kartotékot kijelzi a program.

A kartotékok kiíratása
Szemben a szövegszerkesztővel és a táblázóval, a kartotékok kiírása némileg eltérő, ugyanis lehetőség van a nyomtatás speciális formáinak megadására. Erre szolgálnak az alsó sor jobb szélén álló ikonok.

A jobb sarokban egy **kartotéksorozat** látható. Ebben az esetben a kiírás abban a formában történik meg, ahogy a képernyőn látjuk a kartotékokat.

Mellette **papírlap sorozat** látható. Ha ezt a kiírási formátumot választjuk, a JANE megkérdezi, hogy mely adatokat írjon ki. Az **adatokat egyetlen sorban** fogja elrendezni.

Végül jobbról a harmadik ikon **öntapadós címkéket** ábrázol. Ha ezt választjuk, akkor a JANE újból megkérdezi, hogy milyen adatokat nyomtasson. Ezeknek az adatoknak az elrendezése most téglalap alakban történik meg.

Van még egy lehetőség a nyomtatásra. Ebben az esetben azonban nem az összes kartoték, pusztán egyetlenegy kerül nyomtatásra. Vegyük kézbe a fényképezőgépet és fényképezzük le a teljes kartotékot. Lépjünk ki a JANELIST-ből és térjünk vissza a szövegszerkesztőbe. A lefényképezett kartoték tartalmát a szövegbe bárhová beragaszthatjuk a ragasztós tubus segítségével.

A nyomtatás formátumának megválasztása után a nyomtatás maga a **nyomtató** ikon kiválasztásával történik.

A billentyűzet használata

Utaltunk rá, hogy az eszközök mozgatásán túl valamennyi parancsot a billentyűzetről is bevihetjük. Most felsoroljuk, hogy az egyes funkciók kiválasztása a billentyűzetről hogyan történhet.

A JANE-ben mindenütt használható:

<CTRL-F5> a JANEWRITER (írógép) kiválasztása
<CTRL-F3> a JANECALC (zsebszámológép) kiválasztása
<CTRL-F1> a JANELIST (kartonozó) kiválasztása
<C=> az éppen mutatott eszköz kiválasztása (ugyanaz, mint a <TŰZ> megnyomása)
<HOME> a kéz kiválasztása
<F1> a felfelé mutató nyil kiválasztása
<F3> az olló kiválasztása
<F5> a fényképező gép kiválasztása
<F7> a ragasztós tubus kiválasztása
<RETURN> a kiválasztott eszközt a képernyő következő logikai helyére állítja.
<FEL> az eszközt felfelé mozgatja
<LE> az eszközt lefelé mozgatja
<JOBBRA> az eszközt jobbra mozgatja
<BALRA> az eszközt balra mozgatja
<CTRL-FEL> fél képernyővel feljebb mozgatja a kijelzett részt
<CTRL-LE> fél képernyővel lejjebb mozgatja a kijelzett részt
<CTRL-JOBBRA> fél képernyővel jobbra mozgatja a kijelzett részt
<CTRL-BAL> fél képernyővel balra mozgatja a kijelzett részt
<TAB> a legközelebbi tabulálási pontra viszi az eszközt
<HELP> a kérdőjel kiválasztása
<CTRL-DEL> a számítógép kiválasztása
<CTRL-HOME> a lemez kiválasztása
<CTRL-F7> a nyomtató kiválasztása
<CTRL-RESTORE> a STOP tábla kiválasztása
<STOP> az EXIT kiválasztása
<DEL> törli az eszköz előtt levő karaktert

A JANEWRITE-ban használható:

<CTRL-visszanyíl> bal margóra igazítás (alsó sor első ikon)
<CTRL-1> jobb szélre igazítás (alsó sor második ikon)
<CTRL-2> középre igazítás (alsó sor harmadik ikon)
<CTRL-3> mindkét szélre igazítás (alsó sor negyedik ikon)
<CTRL-4> normál betűtípus kiválasztása (alsó sor ötödik ikon)
<CTRL-5> aláhúzott betűtípus (alsó sor hatodik ikon)
<CTRL-6> vastagított betűtípus (alsó sor hetedik ikon)
<CTRL-7> vastag és aláhúzott betűtípus (alsó sor nyolcadik ikon)
<CTRL-8> felső index (alsó sor kilencedik ikon)
<CTRL-9> alsó index (alsó sor tizedik ikon)
<CTRL-0> szövegrész keresése (nagyító) (alsó sor tizenegyedik ikon)
<CTRL - +> kiírás formátumának beállítása (alsó sor tizenkettedik ikon)
<CTRL-R> a kezet a következő háromszögre mozgatja (szöveg jobb és bal széle, bekezdés)
<C=> kiválasztja a háromszöget
<C=-JOBBRA> jobbra mozgatja a háromszöget
<C=-BALRA> balra mozgatja a háromszöget

A JANECALC-ban használható:

<CTRL-visszanyíl> egyenlőségjel kiválasztása
(alsó sor első ikon)
<CTRL-1> a törlés kiválasztása
(alsó sor második ikon)
<CTRL-2> az összeadás jel kiválasztása
(alsó sor harmadik ikon)
<CTRL-3> a kivonás jel kiválasztása
(alsó sor negyedik ikon)
<CTRL-4> a szorzás jel kiválasztása
(alsó sor ötödik ikon)
<CTRL-5> az osztás jel kiválasztása
(alsó sor hatodik ikon)
<CTRL-6> a százalék jel kiválasztása
(alsó sor hetedik ikon)
<CTRL-7> bal (kezdő) zárójel kiválaesztása
(alsó sor nyolcadik ikon)
<CTRL-8> jobb (záró) zárójel kiválasztása
(alsó sor kilencedik ikon)
<CTRL-9> a SUM jel kiválasztása
(alsó sor tizedik ikon)
<CTRL-0> az AVG jel kiválasztása
(alsó sor tizenegyedik ikon)
<CTRL - +> a CNT jel kiválasztása
(alsó sor tizenkettedik ikon)
<CTRL - -> a MIN jel kiválasztása
(alsó sor tizenharmadik ikon)
<CTRL-font> a MAX jel kiválasztása
(alsó sor tizennegyedik ikon)

<CTRL-R> Sor műveletek elvégzése. Ekvivalens a sort reprezentáló betűre pozicionálással. Hatása a kiválasztott eszköztől függ. Ha kéz, akkor egy üres sort szúr be; ha olló, akkor egy teljes sort vág ki; ha fényképezőgép, akkor lefényképezi a sort, ha ragasztó, akkor átmásol egy már kiválasztott részt a rendszer.

CTRL-C> Oszlop műveletek elvégzése. Ekvivalens az oszlopot reprezentáló számra pozícionálással. Hatása a kiválasztott eszköztől függ. Ha kéz, akkor egy üres oszlopot szúr be; ha olló, akkor egy teljes oszlopot vág ki; ha fényképezőgép, akkor lefényképezi az oszlopot, ha ragasztó, akkor átmásol egy már kiválasztott részt a rendszer.

<CTRL-D> A kezet a parancs sorra mozgatja

A JANELIST-ben használható:

<CTRL-visszanyíl> karton hozzáadása (alsó sor első ikon)
<CTRL-1> első karton kiválasztása (alsó sor második ikon)
<CTRL-2> a következő karton kiválasztása (alsó sor harmadik ikon)
<CTRL-3> az előző karton kiválasztása (alsó sor negyedik ikon)
<CTRL-4> az utolsó karton kiválasztása (alsó sor ötödik ikon)
<CTRL-5> kartonok rendezése (alsó sor hatodik ikon)
<CTRL-6> kartonok kiválogatása (alsó sor hetedik ikon)
<CTRL-7> karton szemétkosárba dobása (törlése)
(alsó sor nyolcadik ikon)
<CTRL-8> öntapadó cimkék nyomtatása (alsó sor kilencedik ikon)
<CTRL-9> kartonok listázása (alsó sor tizedik ikon)
<CTRL-0> kartonok eredeti forma szerinti kinyomtatása
(alsó sor tizenegyedik ikon)

4. fejezet

C-128 CP/M+ V3.0

4.1 A CP/M rendszer általános jellemzői

Bevezetés

A CP/M a 8-bites mikroszámítógépek legelterjedtebb operációs rendszere, az IBM PC megjelenése előtt világszabadalomnak számított. 16-bites változata, a CP/M86 ugyancsak elterjedt, sokan jobbnak tartják, mint az IBM DOS-t. A CP/M a **Digital Research** terméke.

A CP/M a Z80, illetve I8080 kompatibilis proceszorok számára készült. Ahhoz tehát, hogy a C-128-on (vagy a C-64-en) használni tudjuk, ahhoz a gépnek egy ilyen processzorral kell rendelkeznie. A Commodore 64 esetében ezt egy külön CP/M bővítés hozzáillesztésével érhetjük el. A C-128-nak azonban egy beépített Z80 típusú processzora is van. Ezt koprocesszornak hívjuk. Ez az elnevezés arra utal, hogy a C-128 CP/M üzemmódja nem pusztán a Z80 proceszort használja. Hogy miért, arra a későbbiekben visszatérünk.

A CP/M világsikerét rendkívül könnyű hordozhatóságának köszönhette. A programok egy gépről a másikra nem csak azért nem vihetők át könnyen, mert a processzoruk esetleg különböző, hanem azért is, mert az I/O műveleteket eltérően végzik: másképp kezelik a képernyőt és a billentyűzetet, a lemezegységeket. A CP/M -et úgy készítették el, hogy különválasztották a gépfüggő és gépfüggetlen részeket. A gépfüggő részek által ellátandó funkciókat pontosan definiálták, s meghatározták, hogy ezek a funkciók hogyan érhetők el. Ha valaki a CP/M-et használni akarta, csak meg kellett írnia a gépfüggő részeket, s máris használhatta valamennyi CP/M programot. Erről részleteket az LSI ATSZ gondozásában megjelent CP/M operációs rendszer című kiadvány 191-261. oldalain találhatunk.

Azóta persze a világ sokat fejlődött: az IBM PC-k megjelenésével a hardver is szabványos lett: a CP/M hordozhatósága így ma már érdektelen: az számít csak, hogy mint a számítógép operációs rendszere mire képes.

A CP/M-et nem 'játék' gépekhez szánták. Ez onnan is látszik, hogy normál működéséhez két lemezegység is szükséges. Ennek oka az, hogy eredetileg a központi memória igen kevés, 15-20 Kbye volt csak, s így az adatfeldolgozó programok futtatásához könnyen elérhető perifériákra volt szükség. Ezért a lemezegységek általában DMA segítségével dolgoztak, s bár nem olyan intelligensek, mint a VC 1541-es lemezegység, annál lényegesen gyorsabbak, és általában nagyobb kapacitásúak voltak.

A Commodore 128 a CP/M operációs rendszer gépfüggő részeit nem a Z80 segítségével hajtja végre. Mindössze egy 4K-s ROM van a gépben, ami a Z80-as kódokat tartalmazza. Valahányszor az operációs rendszer egy I/O utasítást végre akar hajtani beállít bizonyos paramétereket, majd a

vezérlést az M8510 processzornak adja át, s az végzi el a megfelelő műveleteket. Ennek nagy előnye, hogy így fel tudták használni a C-128 KERNAL már meglevő rutinjait. Ezért egy CP/M program futása közben a két processzor felváltva, kooperálva fut, innen a koprocesszor elnevezés. Ez a megoldás azt eredményezi, hogy a **CP/M programok számára 59K TPA** (tranziens program area) áll rendelkezésre. Az egyik legnépszerűbb adatbáziskezelő rendszer a dBASE II memóriaigénye 48K, így 11K szabad terület még marad. (Ezt a dBASE II a rendezésekhez használja.) Összeségében ez azt jelenti, hogy a legbonyolultabb CP/M-re írt programok is futtathatók a C-128-on.

Nem ilyen jó a helyzet a lemezegységekkel kapcsolatban. Mint már utaltunk rá, a CP/M folyamatosan használja a lemezegységet, nagyban él az **overlay** (**átlapolás**) technikájával. Ez azt jelenti, hogy a programnak egy ún. rezidens része mindig a tárban van, s a memória ún. tranziens (átmeneti) részére a programnak azon darabjai töltődnek, amire éppen szükség van. Helykimélés céljából az üzenetek is lemezegységen vannak, s onnan kell azokat betölteni, s a képernyőre kiírni. Ehhez gyors lemezegységekre van szükség. Ha a CP/M-et csak tanuljuk, akkor a 1541-es egység sebessége éppen hogy elfogadható, de adatfeldolgozásra szükségünk van a Commodore cég új, 1570-es és 1571-es típusú egységeire.

Az átlapolási technika alkalmazásával a programok mérete tetszőlegesen megnövelhető. Ezért azután általában két lemezegységet, vagy egy duál egységet használ a rendszer: az egyik meghajtóban a felhasználói progam, a másikban az adatfile-okat tartalmazó lemez van. Az ideális megoldás persze az, ha van egy Winchester (fix lemezes) egységünk.

A CP/M eszközigénye

A C-128 gépen kívül mindenképp szükség van egy lemezegységre. Célszerűbb a 80 oszlopos monitort használni, de lehetőség van a 40 oszlopos monitor használatára is.A CP/M rendszerlemezt - amelynek mindkét oldalán vannak programok - a Commodore-128-as géppel együtt adják.

A CP/M programokat 80 oszlopos képernyőre készítették, amelynek az alsó sorát a rendszer különféle üzenetek számára tartja fenn. Mivel a cél, hogy valamennyi CP/M program üzemeljen, ezért 'igazi 40 oszlopos' kijelzésre nincs lehetőség. Ezen a C-128 CP/M-je úgy segít, hogy a 40 oszlopos képernyő a 80 oszlopos képernyőn, mint egy ablak csúsztatható, s ilyen módon a teljes képernyő elolvasható. A képernyő mozgatására azonban csak akkor van lehetőség, ha az éppen futó program adatbevitelt vár a billentyűzetről. Más esetben a képernyő nem mozgatható. A DEVICE CP/M paranccsal lehetőség van a képernyő méretének megváltoztatására. Ekkor egyszerre látunk mindent, de a 80 oszlopra tervezett képernyő formátumok 'elromlanak'.

A CP/M betöltése

A CP/M üzemmódba kétféleképpen kerülhetünk. Az első lehetőség, hogy a lemezegység bekapcsolása után, de a gép bekapcsolása előtt a CP/M rendszerlemezt behelyezzük a meghajtóba, majd bekapcsoljuk a gépet. A C-128 automatikus indító funkciója betölti és elindítja a CP/M-et.

Második lehetőség, hogy C-128-as üzemmódban ugyancsak behelyezzük a CP/M lemezt a meghajtóba, s kiadjuk a **BOOT** BASIC parancsot.

Bármelyik lehetőséget választjuk, a rendszer betöltődik, s a képernyőn az alábbi felirat jelenik meg:

```
CP/M3.0 On the Commodore 128
        xx column display

A>
```

xx értéke 40 vagy 80, attól függően, hogy hány oszlopos kijelzést használunk. Ha a BOOT parancs kiadásakor a <40/80 DISPLAY> billentyű le volt nyomva, akkor 80, ellenkező esetben 40 oszlopos képernyőt kapunk. A CP/M használata közben a DEVICE paranccsal válthatunk képernyőt.

Az utolsó sorban a CP/M ún. promptja látható, ami egy betűből (A-P) esetleg egy számból (0-15) és egy > jelből áll. A betű az éppen kiválasztott lemezegység betűjele, a szám a felhasználó száma. A ">" jel után villog a kurzor.

A 40 oszlopos képernyő használata

Mint jeleztük, 40 oszlopos kijelzésre is lehetőség van. A sorok logikailag továbbra is 80 karakterből állnak, de a soroknak csak egy 40 karakteres része látható. A 80 oszlopos logikai képernyőn így egy 40 oszlopos ablakot kapunk. Az ablakot közvetlenül mozgathatjuk a <CTRL-JOBBRA>, illetve <CTRL-BALRA> billentyűkkel, amelyek 5-5 karakterpozícióval mozgatják a képernyőt jobbra, illetve balra. <JOBB>, <BAL> a funkcióbillentyűk között található kurzor mozgató billentyűket jelenti. (Ezek szürkék.) A <CRSR> feliratú kurzor billentyűk erre a célra nem használhatók.

Ezen túlmenően az ablakot automatikusan mozgatja a <RETURN> lenyomása, ha az ablak jobb (bal) szélén túl írunk (vagy törlünk).

Az ablak mozgatására csak akkor van lehetőség, ha a CP/M a billentyűzetről egy adat bevitelére vár.

A lemezegységek használata

A CP/M mind a VC1541, mind a 1571 és 1570 típusjelű lemezegységeket képes kezelni. Az utóbbi két lemezegységbe nem csak Commodore formátumú, hanem IBM kompatibilis lemezeket is helyezhetünk. Pontosabban a rendszer írja/olvassa a következő lemeztípusokat:

EPSON QX10 Valdocs
Kaypro II
Osborn

a CP/M80 operációs rendszer alatt;

IBM PC (Commodore PC10)

a CP/M86 alatt. A használt operációs rendszer típusa azért nem mindegy, mert pl. a CP/M86 katalógusa eltérő szervezésű az IBM DOS katalógusától.

Ha a rendszer felismer egy IBM típusú lemezt, akkor a jobb alsó sarokban megjelenik az IBM felirat. Elképzelhető, hogy a lemez fizikailag ugyan jó, de nem CP/M alatt írták fel. Ekkor a képernyő alsó sorában, középen a **MISSING** felirat jelenik meg.

Hasonló okokból nem használhatjuk a C-128-as üzemódban HEADER utasítással megformázott lemezeket. Ehelyett a FORMAT CP/M paranccsal kell ezeket megformázni.

A CP/M a lemezegységeket a többi perifériától teljesen eltérően kezeli. A csatolt lemezeket A,B,....,P betűkkel jelöli a rendszer, így is kell rájuk hivatkozni. A CP/M azt a dilemmát, hogy általában mindenkinek csak egy lemezegysége van úgy oldja meg, hogy az E lemezegységet az A lemezegységben szimulálja. Ha ezt a lemezegységet használjuk, akkor a képernyő alsó sorában megjelenik a felirat:

INSERT DISK E IN DRIVE A

A megfelelő lemez behelyezése után meg kell nyomnunk a <RETURN> billentyűt. Az E lemezegység használatával elérhető, hogy úgy dolgozhatunk, mintha két igazi lemezegységünk lenne.

Az A lemezegység a 8-as hardver számú egység Ø-ás meghajtója. B ugyanannak a lemezegységnek az 1-es meghajtója. A C lemezegység a 9-es egység Ø-ás meghajtója stb.

A CP/M rendszerben a lemezen tárolt minden file egyforma felépítésű. A különböző file-ok felismerését a nevüket követő háromkarakteres típuskód teszi lehetővé. Ez csak logikai megkülönböztetés: az egyes programok maguk "tudják", milyen típusú file-okkal dolgoznak.

A file-ok jelszóval védhetők. Ebben az esetben a file-ok nevét követően a jelszót is meg kell adni. Ha ezt elmulasztjuk, a CP/M nem nyitja meg a file-t.

Egy file nevét a következő formában adhatjuk meg:

[<drive>:]<file név>[.<tip>][;<jelszó>]

<file név> maximum nyolc, a <tip> típusjelző maximum három karakterből állhat. A <drive> annak a lemezegységnek a betűjele, amin a file található. <jelszó> a SET CP/M parancs segítségével megadott jelszó. Csak akkor kell megadni, ha a file jelszóvédett. Ha elhagyjuk a jelszót, de a file jelszóvédett, akkor a CP/M megkérdezi a jelszót. Ha rossz választ adunk, a program futása megszakad.

Az alábbi példák mutatják, hogyan lehet a file-okat meghatározni:

```
A:PROBA
B:PROBA.TXT
B:PROBA.TXT;5120
LECKE.TAN
```

Az első és második sorban álló file nem ugyanaz. Az első típusa 3 szóközből áll, míg a második TXT. Ezért célszerű a típust a filenév részének tekinteni.

Felhasználói szám

A nagy kapacitású, nem cserélhető háttértárakat több felhasználó is használhatja, hiszen ezek több száz "normál" lemeznyi adatot is képesek tárolni. Annak érdekében, hogy az egyes felhasználók file-jai ne keveredjenek, és a tárolóterületet ne kelljen felosztani,a felhasználó a CP/M használata közben megadhat egy **felhasználói számot.** Ennek értéke egy 0-15 közötti egységszám. Közvetlenül bármelyik felhasználó számára csak a saját számával jelzett file-ok, illetve a 0-ás felhasználói számmal jelzett file-ok érhetők el. Bizonyos parancsok lehetővé teszik a másik felhasználói számú file-ok használatát, de ekkor ezt külön fel kell tüntetni. A felhasználói szám a USER parancs segítségével adható meg. A CP/M promptja ekkor megváltozik. Például a USER 4 parancs kiadása után:

```
4A>
```

Ez azt jelenti, hogy a 4-es felhasználói számot használjuk. Valamennyi file ezt a számot kapja, függetlenül attól, hogy az A vagy valamely más lemezegységen van.

Logikai és fizikai perifériák

A lemezegységektől különböző perifériáknak logikai neveik vannak, s minden ilyenhez hozzá lehet rendelni egy fizikai perifériát. A Commodore-128 CP/M rendszere a következő logikai perifériákkal rendelkezik:

Logikai periféria	**CP/M jele**
Konzol bevitel	CONIN:
Konzol kivitel	CONOUT:
Külső bevitel	AUXIN:
Külső kivitel	AUXOUT:
Listázó periféria	LST:

A külső be- és kivitel eredetileg a lyukszalag olvasó és lyukasztó perifériákat kezelte, ma inkább külső adatátviteli vonal kezelésére használjuk.

A Commodre-128 CP/M-jével a következő fizikai perifériák használhatók:

Fizikai periféria	**CP/M jele**
Üres kiírás	NULL
Billentyűzet	KEYS
40 oszlopos képernyő	40COL
80 oszlopos képernyő	80COL
Nyomtató	PRT1
Nyomtató	PRT2
RS232 csatorna	6551

A DEVICE CP/M parancs segítségével a logikai-fizikai perifériák egymáshoz rendelését tetszés szerint elvégezhetjük.

A CP/M billentyűzet

A CP/M üzemmódnak saját szerkesztője van, ami elsősorban **sorok szerkesztésére** szolgál. Nincs lehetőség a kurzor közvetlen mozgatására sem. A Commodore gépekhez szokott felhasználót elsősorban az lepi meg, hogy a beírt sor szerkesztésére elsősorban a CTRL váltóval való billentyűkombinációkat használhatunk, s mellesleg a kurzor mozgató billentyűket. Ennek oka, hogy a régebbi gépeken nem voltak speciális és funkcionális billentyűk; éppen csak az ABC betűinek volt hely.

Figyelem: szemben a Commodore-128 128-as módjával, a CRSR billentyűk és a külön blokkban levő nyilas kurzor billentyűk hatása nem azonos. Kurzor műveletekre csak és kizárólag a szürke billentyűket használjuk! A CP/M beszúrási üzemmódban dolgozik. Ez azt jelenti, hogy egy billentyű leütésére a megfelelő karakter a képernyőre kerül, s a kurzor helyén levő karakterektől kezdve a sor egy karakterhellyel jobbra csúszik. Így nincs lehetőség karakter felülírására, legfeljebb törlésére.

Az egyes vezérlő karakterek a következők:

<CTRL-A>: a kurzort egy karakterhellyel balra mozgatja.
<CTRL-B>: a kurzort a sor elejére, ha a sor elején áll, a sor végére állítja.
<CTRL-C>: megszakítja a program futását és inicializálja a lemezegységet.
<CTRL-E>: a kurzor és a sor azt követő része a következő sor elejére kerül, de a sort nem küldi még el a rendszer. Hosszabb parancssorozatok begépelésénél használhatjuk.
<CTRL-F>: a kurzort egy karakterhellyel jobbra lépteti.
<CTRL-G>: a kurzor helyén levő karakter törlése. A sor kurzor utáni része egy karakterhellyel balra tolódik.
<CTRL-H>: a kurzor előtt levő karakter törlése. A kurzor és az utána álló karakterek egy karakterhellyel balra tolódnak.
<CTRL-I>: tabulálás. Kezdetben minden 8. karakterhelyre egy tabulálási pontot tesz a CP/M. Ugyanaz, mint a <TAB> lenyomása.
<CTRL-J>: a kurzor a következő sor elejére kerül, s a sor feldolgozásra átadódik a CP/M-nek. Ekvivalens a <CTRL-M> és <RETURN> billentyűzéssel.
<CTRL-K>: a kurzortól kezdődően valamennyi karaktert törli.
<CTRL-M>: a kurzor a következő sor elejére kerül, s a sor feldolgozásra átadódik a CP/M-nek. Ekvivalens a <CTRL-J> és <RETURN> billentyűzéssel.
<CTRL-P>: ki/be kapcsolja a nyomtatót. Ha a nyomtatót éppen bekapcsoltuk, akkor rövid hangjelzést kapunk. **Ezt követően valamennyi, a konzolon megjelenő felirat a nyomtatón is megjelenik.**
<CTRL-Q>: a képernyőre írást engedélyezi.
<CTRL-R>: kiírja az éppen begépelés alatt álló sort. A sorban levő vezérlő karaktereket feldolgozza.
<CTRL-S>: felfüggeszti a képernyőre történő kiírást. <CTRL-Q> oldja fel
<CTRL-U>: a sor végére egy # jelet tesz, s a kurzor a következő sor elejére kerül. A sor # előtti része nem lesz része a parancsnak.
<CTRL-W>: a kurzor helyétől kezdve az utoljára begépelt parancs karakereit ismétli meg. Ez olyankor hasznos, ha hibásan beírt, s a rendszer által végre nem hajtott parancsot akarunk javítani.
<CTRL-X>: törli a sor kurzor előtt álló részét, s a maradék sort az első karakterhelyig előre tolja.
<CTRL-Z>: adatbevitel vége.

Mint láttuk, a parancsok egyetlen sor szerkesztésére vonatkoznak. Sokkal több lehetőségünk van azonban, mint a Commodore 128 64-es üzemmódjában. (A gép C-128-as módja már az itt felsorolt szerkesztési funkciók legtöbbjével rendelkezik.)

A billentyűzet további érdekessége, hogy a billentyűk ASCII karaktereit tetszés szerint definiálhatjuk. Erre egy speciális KEYFIG program szolgál, de bármikor magunk is megtehetjük ezt. A billentyűzet használatának módosítását három meglehetősen szokatlan billentyűkombináció lenyomásával érhetjük el. Ennek oka, hogy a véletlen billentyűzés veszélyét minél jobban csökkentsék. Nevezetesen a **<CTRL> és a jobboldali <SHIFT>** billentyű lenyomva tartása közben az alábbi 3 billentyű valamelyikét nyomjuk meg: <BALRA>, <JOBBRA>, <ALT>. A két kurzor vezérlő billentyű a szürke funkciós billentyűk között található, s csak a megfelelő irányú nyíl van rárajzolva.

Az egyes billentyűzések hatása a következő:

<CTRL-SHIFT-BALRA>: billentyű átdefiniálása
<CTRL-SHIFT-JOBBRA>: szöveg hozzárendelése billentyűhöz
<CTRL-SHIFT-ALT>: bővített billentyűzet ki/be kapcsolása

Billentyű átdefiniálása

A <CTRL-SHIFT-BALRA> billentyű lenyomása után megjelenik a képernyő alsó sorában egy üres ablak. Nyomjuk le azt a billentyűt, amit át szeretnénk definiálni, ennek ASCII kódja hexadecimális alakban azonnal megjelenik az ablakban. A billentyű átdefiniálását az új ASCII karakterérték beírásával végezhetjük. A rendszer csak a 0-9,a-f karaktereket fogadja el. A második számjegy beírása után az átdefiniálás automatikusan megtörténik.

Az ASCII karakterek a következők:

00H	üres billentyű
01H-7FH	normál ASCII kód
80H-9FH	szöveges billentyűk
A0H-AFH	80 oszlopos képernyő: írás színe
B0H-BFH	80 oszlopos képernyő: háttér színe
C0H-CFH	40 oszlopos képernyő: írás színe
D0H-DFH	40 oszlopos képernyő: háttér színe
E0H-EFH	40 oszlopos képernyő: keret színe
F0H	lemezegység kijelzésének ki/be kapcsolása
F1H	megállás (PAUSE) ki/be kapcsolása
F2H	nem használt
F3H	40 oszlopos képernyő: jobbra görgetés
F4H	40 oszlopos képernyő: balra görgetés
F5H	MCM feloldása
F6H-FEH	nem használt
FFH	CP/M újratöltése

Szöveg hozzárendelése billentyűhöz

A fenti 80H-9FH ASCII kódú karakterekhez szövegeket kell hozzárendelni. A billentyű lenyomása a hozzá tartozó szöveg begépelésével ekvivalens. A szövegek definiálására a <CTRL-SHIFT-JOBBRA> billentyűzés után kerülhet sor. A képernyő alsó sarkában megjelenik egy hosszú, üres ablak. Ezután egy olyan billentyűt kell megnyomnunk, amelyik ASCII kódja a 80H-9FH tartományba esik. Azonnal megjelenik az ablakban a billentyűhöz tartozó szöveg. Ezt a szöveget módosíthatjuk. A szövegbe vezérlő karakterek is

kerülhetnek, így a beírt szövegen módosítani, illetve a szerkesztést befejezni ugyancsak a <CTRL-SHIFT-RETURN> billentyűkombinációval lehet:

<CTRL-SHIFT-RETURN> szöveg definiálásának vége
<CTRL-SHIFT-+> szóköz beszúrása
<CTRL-SHIFT--> törli a kurzor karakterét
<CTRL-SHIFT-BALRA> kurzor balra
<CTRL-SHIFT-JOBBRA> kurzor jobbra

Bővített billentyűzet ki/be kapcsolása
Lehetőség van a 7FH értéknél nagyobb ASCII kódú karakterek letiltására, illetve engedélyezésére. Ezt a <CTRL-SHIFT-ALT> billentyűzéssel érhetjük el. A CP/M betöltése után bővített billentyűzet módba kerülünk, s ezért valamennyi karaktert használhatjuk.

A képernyő használata

A CP/M C-128-on történő megvalósítása egy ADM3A típusú terminált szimulál, amelyen az ADM31 funkciók is végrehajthatók. Ennek megfelelően, ha egy CP/M programot konfigurálunk, s lehetőség van a terminál típusának megválasztására, akkor ezt a két típust választhatjuk. Bizonyos funkciók (például villogtatás) természetesen csak a 80 oszlopos képernyőn használhatóak.

Az ADM3A-n használható vezérlő karakterek a következők:

<CTRL-G>: hangjelzés adása.
<CTRL-H>: kurzor balra.
<CTRL-J>: kurzor le.
<CTRL-K>: kurzor fel.
<CTRL-L>: kurzor jobbra.
<CTRL-M>: kurzor a következő sor elejére.
<CTRL-Z>: képernyő törlése, a kurzor a home pozícióba.
<ESC> = rc: A kurzor pozíciójának beállítása. r a sor, c pedig az oszlop helyét megadó karakter. Ennek megfelelően r helyén tetszőleges karakter állhat a <szóköz>-től a <8> számjegyig (ASCII kódjaik 32-54). Hasonlóan c helyén tetszőleges karakter állhat a <szóköz>-től a <SHIFT-o> betűig (ASCII kódjaik 32-111).

Az ADM31 terminál további ESC sorozatok végrehajtására képes, amiket a C-128 CP/M-je ugyancsak szimulál:

<ESC> T vagy **<ESC> t:** a sor törlése a kurzortól a sor végéig.
<ESC> Y vagy **<ESC> y:** a képernyő törlése a kurzor pozíciótól a képernyő végéig.
<ESC> * vagy **<ESC> :** képernyő törlése és a kurzor home pozícióba való állítása.
<ESC> Q: karakter beszúrása.
<ESC> W: karakter törlése.
<ESC> E: sor beszúrása.
<ESC> R: sor törlése.
<ESC><ESC><ESC> C: A képernyő színeinek beállítása. C helyén a <szóköz>-től az <o>-ig tetszőleges karakter állhat. A színek a karakter ASCII kódjának értékétől függően a következőképpen állítódhatnak be:

20H-2FH az írás színe;
30H-3FH háttér színe;
40H-4FH a keret színe (csak 40 oszlopos képernyő esetén).

<ESC> >: csökkentett intenzitás. (nincs megvalósítva)
<ESC> <: teljes intenzitás. (nincs megvalósítva)
<ESC> G4: inverz kijelzés bekapcsolása.
<ESC> G3: aláhúzás bekapcsolása.
<ESC> G2: villogás bekapcsolása.
<ESC> G1: karakterkészlet váltás.
<ESC> G0: valamennyi <ESC>-G-vel kapcsolt funkció törlése.

4.2 CP/M parancsok

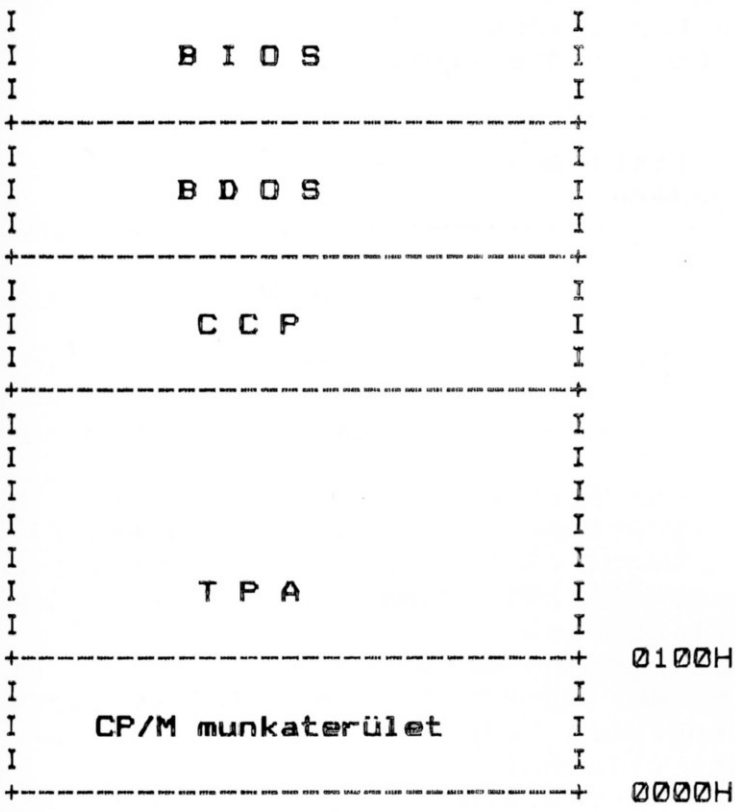
4.2.1 A CP/M parancsok felépítése

A CP/M rendszer - éppen a memóriaterület korlátozott volta miatt - maga is átlapolt rendszer. Ez azt jelenti, hogy egy-egy parancs a végrehajtásához szükséges program betöltésével kezdődik. Ha pl. egy lemezt akarunk megformázni, akkor először a FORMAT.COM nevű programfile töltődik be a memóriába, s csak azután kezdődhet meg magának a parancsnak a végrehajtása. Ez teljesen eltér a BASIC-ben megszokottól, ahol valamennyi parancsnak megfelelő program már a ROM-ban van, s pl. a HEADER utasítás kiadása után annak végrehajtása azonnal megkezdődhet.

Ennek a rendszernek kétségtelen előnyei vannak. Egyrészt a rendszer felfelé nyitott, tehát akárhány új parancs építhető be. (Ennek végül is a lemez tárolókapacitása szab csak határt). Másik előnye, hogy a felhasználói programok is parancsként írhatók meg, s így azok elindítása és a CP/M rendszer részét képező parancsok kiadása között nincs különbség.

A CP/M C-128-on való megvalósításának egyik leggyengébb pontja éppen a Commodore lemezegységek lassúsága. Ahhoz, hogy a parancsok végrehajtásának sebessége megközelítse a C-128-as módét, az új 1570 és 1571-es lemezegységeket kell használni.

A CP/M a memóriát a következőképpen használja:



A memória első 256 byte-ját a CP/M saját munkaterületének használja. A tranziens program area (TPA) a 0100H címen kezdődik. Ide töltődnek be a CP/M parancsok és a felhasználói programok, amelyek belépési pontja is 0100H. A TPA nagysága a C-128 esetén 59K, ami annak köszönhető, hogy a rendszer él a memórialapozási lehetőséggel, s a memóriának azon a szeletén, amelyen a CP/M fut, csak a periféria-műveletek belépési pontjai vannak. A másik 64K-s szeleten helyezkednek el a BIOS, illetve a BDOS 'igazi' részei.

A be/kiviteli műveletek kezelése két részre oszlik. A **logikai és fizikai perifériák** kezelését a **BIOS** (basic input-output system), a **lemezegységek kezelését pedig a BDOS** (basic disc operation system) nevű programrész végzi el. Ez utóbbi két komponens a CP/M memóriaterület végén helyezkedik el, s állandóan a memóriában van (rezidens). A CP/M utolsó eleme a **CCP** (command consol processor), ami a CP/M parancsok értelmezését, a szükséges programok betöltését és elindítását végzi.

A mondottak ellenére néhány parancs végrehajtásához a szükséges rutinokat a rendszer állandóan a memóriában tartja: ezeket **rezidens parancsok**nak hívjuk, szemben az ún. **tranziens** parancsokkal.

A tranziens parancsok alapszóval kezdődnek, az ugyanilyen nevű COM típusú file tartalmazza a parancs végrehajtásához szükséges programot. A CCP a parancsot követő paraméterek közül az elsőt mindig egy file-névnek tekinti, s az ennek megadásához szükséges információkat is előállítja. A parancs többi részét a CCP az input pufferben változatlanul hagyja.

A C-128 CP/M+ V3.0 rezidens parancsai a következők:

Parancs	Funkció
DIR	lemez katalógusának a képernyőre listázása
DIRSYS	A SYS típusú file-ok képernyőre listázása
ERASE	file-ok törlése
RENAME	file-ok átnevezése
TYPE	ASCII file képernyőre listázása
USER	felhasználói szám megadása

A tranziens parancsok a következők:

Parancs	Funkció
COPYSYS	a CP/M rendszer átmásolása
DATE	a dátum és pontos idő beállítása, lekérdezése
DEVICE	a fizikai és logikai perifériák egymáshoz rendelése
DIR	a katalógus paramétereinek lekérdezése
DUMP	hexadecimális file kilistázása
ED	CP/M sor-orientált szövegszerkesztője
ERASE	több file egyidejű törlése
FORMAT	CP/M rendszerben használható lemez megformázása
GET	konzol input egy adott file-ból
HELP	információkérés a CP/M használatáról
INITDIR	lemez katalógusának átalakítása
PATCH	CP/M rendszeradatok módosítása
PIP	file-ok másolása és összefűzése
PUT	konzol output átirányítása egy adott file-ba
RENAME	több file egyidejű átnevezése
SAVE	memória adott részének lemezre való másolása
SET	a lemez és file-jai védelmi paramétereinek beállítása
SETDEF	a rendszer keresési sorrendjének beállítása
SHOW	a file-ok attributumainak lekérdezése
SUBMIT	kötegelt feldolgozás végrehajtása
TYPE	több file egyidejű kilistázása

A felsorolásból látszik, hogy a DIR, ERASE, RENAME és TYPE rezidens parancsoknak tranziens változata is van. A CP/M a file-név és az utasítás paramétereinek ellenőrzésével eldönti, hogy a parancsot a rezidens résszel is végre tudja-e hajtani. Ha nem, akkor betölti és elindítja a tranziens változatot.

4.2.2 A CP/M működését befolyásoló parancsok

A CP/M parancsok egy része **magának a rendszernek a működését befolyásolja, míg a többiek igazából utility funkciókat** valósítanak meg. Először az előbbiekről lesz szó.

COPYSYS

A CP/M rendszer másolását végzi.

Szintaxis: **COPYSYS**

Utaltunk rá: a CP/M nagy sikerét könnyű hordozhatóságának köszönhette. Ezt azzal is támogatta a CP/M, hogy magának a CP/M-nek a részei voltak a rendszer másolásához szükséges programok.

A COPYSYS utasítás a C-128-on nincs implementálva. A parancs kiadásakor megjelenik a képernyőn az az üzenet, hogy a CP/M-et a PIP segítségével lehet másolni. Ennél azonban egyszerűbb és gyorsabb, ha valamilyen C-64 vagy C-128 másolót (pl. a DOS SHELL-t) használunk.

Ha mégis a PIP-pel másoljuk a rendszert, akkor csak a FORMAT segítségével már megformázott lemezre másolhatunk. Duál lemezegység esetén a másolást a

PIP B:=*.* [VO]

paranccsal indíthatjuk el. Egymeghajtós lemezegység esetén ugyanezt a

PIP E:=*.* [VO]

paranccsal indíthatjuk el. A rendszer felváltva kéri az új (logikai E:), s a régi (A:) lemezt a fizikai A: lemezegységbe.

DATE

A dátum és a pontos idő beállítása, illetve lekérdezése.

Szintaxis:

```
        {CONTINUOUS}
DATE    <időpont>
        SET
```

Az utasítás első alakjának használata esetén a gépben utoljára tárolt értéktől indul el az óra. A második esetben az <időpont> az új értéket mutatja. Ennek formája: **nn/hh/éé óó:pp:mm**, ahol nn a nap, hh a hónap éé az év, óó az óra, pp a perc, mm pedig a másodperc értékét adja. Ha valamelyik egy jegyű, pl. 3, azt 03 alakban kell beírni.

Az utasítás harmadik alakjában a dátum és a pontos idő megadására párbeszédes formában kerül sor. A rendszer a beírás formátumát is jelzi. Külön kérdés indítja magát az órát. A parancs ellenőrzi a dátum és az időpont helyességét is.

A DATE paraméter nélküli használata az időpont és óra lekérdezése. A rendszer ezenkívül kiírja a hét napját is.

Példák:
```
  (i)     DATE C
 (ii)     DATE 25/11/86 14:23:00
(iii)     DATE 31/11/86
```

Az (i) példa a DATE CONTINUOUS rövidítési lehetőségét mutatja. (ii) 1986 november 25-re állítja be a dátumot. (iii) kiadása után hibajelzést kapunk, mert a november hónap csak 30 napos.

DEVICE

Lehetővé teszi fizikai perifériák hozzárendelését a logikai perifériákhoz, továbbá a perifériák bizonyos paramétereinek beállítását.

Szintaxis:

(i)

DEVICE **NAMES** <fizper>
DEVICE **VALUES** <logper>

(ii) **DEVICE** <logper>=<fizper> {<param1>} {,<fizper> {<param2>},...}

(iii) **DEVICE** <logper>=**NULL**

(iv) **DEVICE CONSOLE** [<param>]

A parancs (i) alakja a rendszerben használható fizikai perifériák elnevezését (NAMES), az aktuális logikai-fizikai periféria egymáshoz rendelést (VALUES), továbbá az egyes fizikai és logikai perifériák paramétereit írja ki.

A (ii) alakban a rendszer egy logikai perifériájához egy, vagy több fizikai perifériát rendelünk hozzá. A fizikai perifériák bizonyos tulajdonságait is megadhatjuk. Tulajdonságokat csak az RS232 csatornára adhatunk meg. Ezek a következők:

XON = X-kapcsolt protokoll szerint kommunikál;
NOXON = kikapcsolja a XON paramétert;
<átvit>= átviteli sebesség. értéke az alábbiak egyike lehet: 50, 75, 110, 134, 150, 300, 600, 1200, 1800, 2400, 3600, 4800, 7200, 9600, 19200. Az érték baud-ban értendő.

A (iii) esetben egy logikai perifériát NULL-perifériának definiálunk. Ez azt jelenti, hogy a periféria csak logikailag létezik, de nem csinál semmit. A megfelelő be- és kiviteli műveletei egyetlen RET utasításból állnak. Elsősorban output esetén használjuk (pl. nincs szükség program-listára).

Az (iv) alakban a képernyő paramétereit állíthatjuk be. Az egymástól vesszővel (,) elválasztott paraméterek a következők lehetnek:

PAGE = 16 sor x 25 oszlopos képernyő;
LINES =<sor> <sor>-nyi sort használ a rendszer;
COLUMNS=<oszlop><oszlop>-nyi oszlopot használ a rendszer.

Példák:
```
   (i) DEVICE
  (ii) DEVICE NAMES
 (iii) DEVICE VALUES
  (iv) DEVICE LST:
   (v) DEVICE CONOUT:=40COL,PRT2
  (vi) DEVICE AUXIN:=6551[XON,1800]
 (vii) DEVICE LST:=NULL
(viii) DEVICE CON:=[COLUMNS=40,LINES=20]
```

Az (i) alatti parancs kiadása után a képernyőre íródik a fizikai perifériák neve, jellege (ki vagy beviteli művelet-e), és lehetséges paramétereik értéke. Rögtön ezután a rendszer kiírja a jelenlegi logikai-fizikai periféria hozzárendelést. (ii) hatására csak a létező fizikai perifériák, (iii) hatására pedig csak a hozzárendelések adatai kerülnek a képernyőre.

(iv) kiírja a LST: logikai perifériához hozzárendelt fizikai perifériát. Ha egyetlen fizikai perifériát írunk be paraméternek, akkor annak adatait kapjuk meg.

Az (v) példa a konzol outputot a 40 oszlopos képernyőre és a 2-es számú nyomtatóra irányítja. (A két eszközre egyszerre ír a rendszer.) A parancsnak ezt a formáját használhatjuk a 40, illetve 80 oszlopos kijelzés cseréjére. Ha például az (v) alatti parancs kiadásakor 80 oszlopos kijelzési módban voltunk, akkor a rendszer áttér a 40 oszlopos kijelzésre.

A (vi) példában a kiegészítő bemenetet az RS232-es csatornának definiáljuk, s beállítjuk a XON paramétert, s az átvitel sebességét 1800 baud-ban adjuk meg.

A (vii) példában a LST: perifériát NULL-nak definiáltuk, az esetleges listák nem készülnek el. A (viii)-ban a képernyő oszlopszámát 40-ben, sorainak számát 20-ban adtuk meg.

GET

Hatására a konzol input-ot a rendszer az utasításban megadott file-ból olvassa.

Szintaxis:
```
  (i)    GET {CONSOLE INPUT FROM} FILE <filespec> {[<param>]}
 (ii)    GET {CONSOLE INPUT FROM} CONSOLE
```

Az (i) alakban a konzol input nem a billentyűzetről, hanem a parancsban megadott file-ból jön. A <param> paraméterlista vesszővel elválasztva az alábbi paramétereket tartalmazhatja:

ECHO az input a képernyőn is megjelenik
NOECHO az input nem jelenik meg a képernyőn
SYSTEM a CP/M parancsokat is a file-ból olvassa a rendszer

Az utasítás (ii) alakjának végrehajtása után a konzol inputot újból a CONIN: logikai perifériáról olvassa a rendszer.

Példák:

```
  (i)     GET FILE MASOLAS.TXT [ECHO]
          PIP
          GET CONSOLE
 (ii)     GET FILE PROBA.TXT [SYSTEM]
```

Tegyük fel, hogy a MASOLAS.TXT file mindössze a **B:=A:*.*** szöveget tartalmazza. Az (i) példa első utasítására a konzol inputot a CP/M a MASOLAS.TXT nevű file-ból olvassa. A PIP parancs kiadása után megjelenik a *, a PIP promptja, majd mellé íródik a MASOLAS.TXT file-ból a B:=A:*.* szöveg. A GET utasításban használtuk az ECHO paramétert. A másolás végrehajtása után visszatérünk a CP/M-be.

(ii) esetén a rendszerparancsokat is a file-ból várja a rendszer. Ha például ugyanazt a hatást akarjuk elérni, mint az (i) példában, akkor a PROBA.TXT file-nak a következő szöveget kell tartalmaznia:

```
PIP
B:=A:*.*
GET CONSOL
```

Ha ez így van, akkor mind a rendszerparancsokat, mind a programok adatait a file-ból veszi a rendszer, egészen a GET CONSOL végrehajtásáig.

PATCH

Adott CP/M rendszerprogramot (általában a módosítás sorszámát jelző) azonosító számmal lát el.

Szintaxis: **PATCH** <filespec> {<sorszám>}

A <sorszám> előjel nélküli egész szám, az 1-32 intervallumba esik. <filespec> csak COM, PRL és SPR típusú file lehet. Ha a sorszám elmarad, akkor a CP/M kiírja a konzolra, hogy melyik 'patch'-et installálta.

Példa:

```
  (i)     PATCH SETUP 12
```

A program végrehajtása során a következő rendszerüzenetet kapjuk:

Do you want to indicate that Patch #12 has been installed for SETUP.COM?

(Jelezni akarja, hogy a SETUP.COM-ra a #12 sorszámó módosítást felvezette?)

Az Y válaszra az installálás végrehajtódik.

PUT

A konzol és a nyomtató outputot egy file-ba irányítja át.

Szintaxis:
(i) **PUT CONSOLE {OUTPUT TO} FILE** <filespec> **{[**<param>**]}**
PUT PRINTER {OUTPUT TO} FILE <filespec> **{[**<param>**]}**
(ii) **PUT CONSOLE {OUTPUT TO} CONSOLE**
PUT PRINTER {OUTPUT TO} PRINTER

A parancs (i) alatti alakjai lehetőséget biztosítanak a konzol és nyomtató kimenet átirányítására az utasításban megadott file-ba. A <param> paraméterlista vesszővel elválasztva az alábbi paramétereket tartalmazhatja:

ECHO a rendszer a képernyőre, illetve a nyomtatóra is elküldi a kiírandó adatokat;
NOECHO az adatok csak a lemezes file-ba kerülnek be;
FILTER a <CTRL-.> karakterek nem kerülnek a file-ba;
NOFILTER a <CTRL-.> karakterek is a file-ba kerülnek;
SYSTEM a rendszerüzenetek is a file-ba íródnak.

A (ii) alatti utasításokkal megszüntethetjük a file-ba való átirányítást.

Példák:
(i)
```
PUT CONSOLE FILE PROBA.TXT [ECHO]
DIR
PUT CONSOLE CONSOLE
TYPE PROBA.TXT
```

Az (i) példa első sorában a konzol outputot átirányítjuk a PROBA.TXT file-ba, majd kiadjuk a DIR parancsot. Az ECHO paraméter használata miatt mind a képernyőre, mind a file-ba kiíródik a katalógus. Ez utóbbiról úgy győződhetünk meg, hogy visszairányítjuk a konzolra a rendszer outputot, majd a TYPE-pal kiíratjuk a PROBA.TXT file tartalmát. (A PUT parancs a BASIC CMD utasításának felel meg.)

SETDEF

A CP/M bizonyos paramétereit állítja be.

Szintaxis:
(i) **SETDEF** {<lemezek>}**{[TEMPORARY=d:]}**
{[ORDER=(<tip1>**{,**<tip2>**})]**
(ii) **SETDEF [**<param>**]**

Mint utaltunk rá, a CP/M a parancsok végrehajtását a megfelelő file betöltésével kezdi. A SETDEF parancs segítségével megadhatjuk, mely lemezegységeken, milyen sorrendben keresse ezeket. Módunk van továbbá megadni, hogy először a COM vagy SUB típusú file-okat keressen a rendszer. Mindezeket az adatokat a SETDEF (i) alatti formájával érhetjük el. <lemezek> maximum négy, egymástól vesszővel elválasztott lemez betűjele. d annak a drive-nak a betűjele, ahol a rendszer a **$$$** típusú (programok ideiglenes munkafile-jai) file-okat tárolja. A

keresés sorrendjét az ORDER használatával adhatjuk meg. <tip1> és <tip2> csak COM vagy SUB lehet.

A <param> paraméterlista egymástól vesszővel elválasztott paramétereket tartalmaz. A lehetséges paraméterek a következők:

DISPLAY a parancsok kiadása után a rendszer kiírja a parancs nevét, verzió-, és felhasználói számát;
NODISPLAY a fenti kiírás elmarad;
PAGE a konzol PAGE üzemmódban dolgozik;
NOPAGE megszünteti a PAGE hatását.

Példák:

```
  (i)     SETDEF [ORDER=(SUB)]
 (ii)     SETDEF A:,*,E:
(iii)     SETDEF *
 (iv)     SETDEF [DISPLAY]
```

Az (i) parancs kiadása után a rendszer csak a SUB file-ok után végez keresést. (ii)-ben azt definiáljuk, mi legyen a keresés sorrendje: először az A: lemezegység, azután az éppen aktuális lemezegység (ezt jelzi a *, s ennek betűjele látható a CP/M > promptja előtt), legvégül az E: egység. (iii)-ban a keresést a mindenkori aktuális lemezegységre korlátozzuk.

(iv) hatására az egyes parancsok végrehajtása előtt a képernyőre íródik a program neve, verzió- és felhasználói száma.

SUBMIT

Kötegelt (batch) feldolgozási lehetőséget biztosít.

Szintaxis: {**SUBMIT**} {<filespec>} {<argumentumlista>}

<filespec> SUB típusú file, s a SUBMIT ennek, mint egy **parancs sorozatnak** (batch-nek) a feldolgozását végzi. Lehetőség van paraméterek használatára is, ezek jele $0,$1,...,$9. A $1,$2,...,$9 aktuális értékét az <argumentumlista>-ban lehet megadni, vesszővel elválasztva. A $0 paraméter értéke a <filespec>-cel lesz azonos. A CP/M a <filespec> file egyes rekordjait a CP/M-nek szóló parancsnak tekinti, s végrehajtja. Ha a parancsokban szereplő programoknak szeretnénk adatokat átadni, akkor azokat a < jellel kell bevezetni. Ha ezekre valamiért nincs szükség, akkor WARNING: Input data ignored! üzenetet kapunk. A <filespec> file-t tetszőleges szövegszerkesztővel, pl. az ED-vel is előállíthatjuk. A SUBMIT szót magát csak abban az esetben hagyhatjuk el, ha a SETDEF ORDER-ben a SUB paramétert megadtuk.

Példák:

```
  (i)     DATE SET
          SETUP
          <GU
          DBASE BOOT
```

```
(ii)    ED $1.FOR
        <AØB#T
        FORT $1
        FRUN $1

(iii)   DATE SET
        SETUP
        TYPE UZENET

(iv)    E:FORMAT A:
        E:INITDIR A:
        E:SET [NAME=$1,PASSWORD=$2, PROTECT=ON,
              CREATE=ON,UPDATE=ON]
```

Tegyük fel, hogy a DBRUN.SUB file az (i) alatti parancsokat tartalmazza. A SUBMIT DBRUN CP/M parancs kiadása ekvivalens az (i) alatti parancssorozat kiadásával. Először a DATE SET kerül végrehajtásra, majd a SETUP. A <GU hatására a SETUP programnak a GU karaktersorozat kerül átadásra. A billentyűzet DIN lesz (G=German) és a CBM nyomtató helyet a felhasználói kapura illesztett nyomtatót használhatunk (U=User port). Végül betöltődik a Dbase adatbáziskezelő rendszer, s elindul a BOOT nevű Dbase program. Valamennyi programnak az aktuális lemezen kell lennie. Ha először kiadjuk a SETDEF [ORDER=(SYS,SUB)] parancsot, akkor elég a DBRUN alak használata!

(ii) segítségével megszerkeszthetünk egy FORTRAN programot, lefordíthatjuk, majd elindíthatjuk. A megfelelő SUBMIT kiadásakor a $1 paraméter értékét is meg kell adni. Ha a (ii) alatti parancsokat a FORT.SUB file tartalmazza, akkor például a SUBMIT FORT PROBA CP/M parancs kiadására a szövegszerkesztő a PROBA.FOR file szerkesztését végzi, s az ebben tárolt tárgynyelvi szöveget fordítja le a FORTRAN fordító. A <AØB#T hatására az 'AØB#T' karaktersorozat inputként átadódik a szövegszerkesztőnek. Ennek hatására a forrásnyelvi program a Pufferbe töltődik, s annak tartalmat a rendszer kilistázza.

A (iii) az általam használt PROFILE.SUB file tartalma. Mint a bevezetésben utaltam rá, ha a CP/M lemez tartalmazza a SUBMIT.COM és a PROFILE.SUB file-okat, akkor a CP/M a bejelentkezés után automatikusan végrehajt egy SUBMIT PROFILE parancsot is. Ennek hatására a (iii)-ban látható parancsok hajtódnak végre: lehetőség van tehát a dátum/óra és a nyomtató beállítására. Az UZENET nevű file mindössze egyetlen rekordot tartalmaz, amivel a CP/M jó munkát kíván.

A (iv) példa jelszó védett lemezek formázására való. Egyben példa arra, hogyan használhatunk egy .SUB file-ban több lemezegységet. Az első sor megformázza az A: egységben levő lemezt, a második átalakítja annak katalógusát, míg a harmadik megadja a lemez nevét és jelszavát, illetve beállítja a védelmet. Ha a parancsokat a PROTECT.SUB file-ban tároltuk, akkor a SUBMIT PROTECT ADATBAZIS KUKORICA parancs kiadásával a lemezt ADATBAZIS névre formázzuk meg, s a jelszó KUKORICA lesz. A közben használt programoknak a logikai E: egységen kell lenniük.

4.2.3 CP/M utility programok

A CP/M parancsok legtöbbje a lemezegységgel és a lemezekkel kapcsolatos műveletek végzését támogatja. Ebben a részben ezeket, és az egyéb - eddig még nem tárgyalt - CP/M parancsokat soroljuk fel.

DIR

A képernyőre (pontosabban CONOUT: logikai perifériára) listázza a lemez katalógusának paramétereit.

Szintaxis: (i) **DIR** <meghajtó> [<param>]
DIR {<filespec>} {<filespec>}...[<param>]

(ii) **DIRS** <meghajtó>
DIRS {<filespec>}

A parancs rezidens parancsként hajtódik végre, ha nem használunk paramétereket, s az (i)-ben mindössze egyetlen <filespec>-t adunk meg. A <meghajtó> egy tetszőleges lemezegység betűjele a kettősponttal: **B:**. <filespec> a 4.1-ben megadott formátumú file leírás lehet, a file névben használhatjuk a * és ? karaktereket is. A <param> paraméterlista, az egyes paramétereket vesszővel kell elválasztani egymástól. A <param> paraméterlistát szögletes zárójelek közé kell tenni! A paraméterek sorrendje érdektelen.

A használható paraméterek a következők:

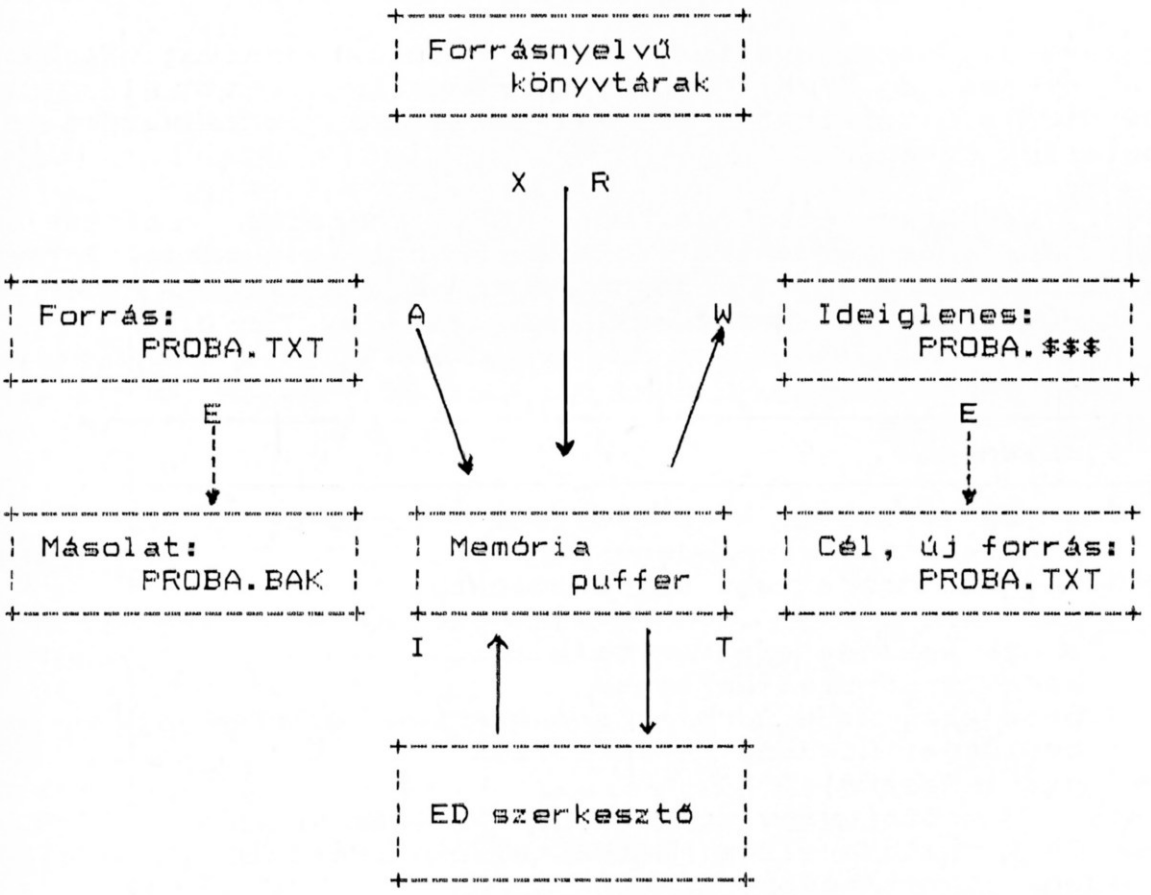
Paraméter	Jelentés
ATT	file-ok paramétereit is kiírja
DATE	file-ok időadatait is kiírja
DRIVE=ALL	valamennyi lemezegységen levő lemez file-jainak adatát kiírja
DRIVE=(A,B,...,P)	csak a zárójelben felsorolt egységeket veszi figyelembe
DRIVE=d	csak a d egységet veszi figyelembe
EXCLUDE	azon file-okat veszi csak figyelembe, amelyek alakja nem egyezik a <filespec>-ben megadottal
FF	ha a nyomtatót <CTRL-P>-vel aktivizáltuk, elküld egy form feed (lapemelés, 0CH) karaktert a listázás kezdetekor
FULL	a file-ok összes adatát kilistázza
LENGTH=n	a nyomtatón egy lapra n sort ír
MESSAGE	kiírja, hogy a file-t melyik egységen, s milyen felhasználói szám alatt találta
NOPAGE	nincs lapdobás
NOSORT	nem rendezi abc sorrendbe a file-okat
RO	csak az RO (read only) file-okat listázza
RW	csak az RW (read-write) file-okat listázza
SIZE	a file-ok méretét írja ki K-ban
SYS	csak a SYS típusú file-okat írja ki
USER=ALL	az összes felhasználói számon levő file-t kilistázza
USER=n	az n felhasználói számhoz tartozó file-okat írja ki
USER=(0,1,2...,15)	csak a felsorolt felhasználói számon levő file-okat listázza

DUMP

Hexadecimális file-t ASCII alakban a konzolra listáz.

Szintaxis: **DUMP** <filespec>

Példák:
(i) DUMP A:DIR.COM
(ii) DUMP E:HELP.TXT



ED

Az ED paranccsal indíthatjuk el a CP/M egysoros szövegszerkesztőjét.

Szintaxis: ED <forrás> {<cél>}

<forrás> tetszőleges file lehet, ami a szerkesztendő szöveget tartalmazza. <cél> pedig az a filenév, ami alatt a szövegszerkesztő az átszerkesztett szöveget elhelyezi. Ha ez elmarad, akkor az ED a régi néven írja vissza az átszerkesztett szöveget. Ha a <forrás> file még nem létezik, akkor az ED úgy értelmezi, hogy most akarunk egy új szöveget írni, s létrehozza a file-t.

A <cél> file nem létezhet a lemezen. Ha mégis, akkor az ED-től Output File Exists, Erase It üzenetet kapunk.

Az adatok biztonságos megőrzése és a memória korlátozott mérete miatt az ED a szövegszerkesztést egy pufferen keresztül biztosítja. A <forrás> file-ból lehet sorokat a pufferbe tölteni, ott szerkeszteni, majd a már kész sorokat kiírni a <cél> file-ba. Mivel a <cél> file azonos lehet a <forrás> file-lal, ezért az ED először egy ideiglenes file-ba írja az adatokat. Pl. az ED PROBA.TXT parancs hatására az ábra szerinti file-ok jönnek létre.

A pufferből kiíródó sorok először a PROBA.$$$ nevű ideiglenes file-ba íródnak. Ha a szövegszerkesztést sikerrel befejeztük, akkor az ED a PROBA.TXT <forrás> file-t átnevezi PROBA.BAK-re, majd a PROBA.$$$-t PROBA.TXT-re. Az új forrásfile mellett tehát a régi file is megmaradt. Ha tehát elrontottunk valamit, lehetőség van a hiba korrigálására.

Az ED a szövegszerkesztésére egyetlen betűből álló parancsokat kínál. Ezek részletes leírása a **CP/M Operációs rendszer** című LSI ATSZ kiadvány 85-104 oldalain található, itt mindössze a legszükségesebb ismereteket foglaljuk össze.

Az Ed-nek egy ún. karakter mutatója van (CP, character pointer), amelyik a memória pufferben az aktuális karakter helyét mutatja. Ennek a mozgatásával tudunk a szövegben mozogni, s ez határozza meg, hogy a szövegen végzett műveletek hol kezdődnek.

Az ED egybetűs parancsai

Parancs	Jelentés
a append	a forrásfileból sorok töltése a pufferbe
b bottom	a CP-t a szöveg elejére állítja
c character	a CP-t jobbra vagy balra mozgatja
d delete	a CP előtt vagy után karaktereket töröl
e end	a szerkesztés sikeres befejezése
f find	karaktersorozat keresése
h head	befejezi, majd újrakezdi a szerkesztést
i insert	beszúrási üzemmód bekapcsolása
j juxtapose	csere és törlés
k kill lines	a CP előtti vagy utáni sorok törlése
l move lines	CP mozgatása előre, hátra teljes sorokkal
m macro	egy utasításcsoport megismétlése
n next	keresés a puffer töltésével
o original	sikertelen vég; újraindul az eredetivel
p page	a CP-t 23 sorral mozgatja előre vagy hátra
q quit	kilépés a szerkesztőből, az eredeti megmarad
r read	LIB típusú file beolvasása
s substitute	karaktersorozatok cseréje
t type	teljes sor kijelzése
u upper case	minden karakter nagybetű lesz
v verify	puffer kapacitása
w write	a pufferből sorokat ír az ideiglenes file-ba
x transfer	sorokat ír/olvas egy segédfile-ból
z sleep	felfüggeszti a szerkesztőt

ERASE

File-okat töröl.

Szintaxis: **ERASE** <filespec> {[**CONFIRM**]}

Az ERASE a <filespec>-nek megfelelő file-okat (logikailag) törli a lemezről. A <filespec> tartalmazhatja a ? és * karaktereket is. Ilyenkor egyszerre több file-t is törölhetünk. Ha használjuk a CONFIRM paramétert, akkor minden file törlése előtt megkérdezi, hogy valóban törölni akarjuk-e. Ha a <filespec> nem tartalmaz * vagy ? karaktert, akkor az ERASE rezidens parancsként hajtódik végre. Ha tartalmaz, akkor az ERASE.COM file-t a rendszer betölti. Az ERASE szót elég ERA alakban beírni.

Példák:

```
  (i)     ERASE PROBA.TXT
 (ii)     ERASE E:PROBA.TXT
(iii)     ERASE GRID*.*
 (iv)     ERASE *.* [CONFIRM]
```

Az (i) parancs törli a PROBA.TXT nevű file-t. (ii) ugyanezt a file-t törli, csak az E: lemezről. (iii) törli a lemezről az összes GRID-del kezdődő, tetszőleges típusú file-t. (iv) - ha nem használnánk a [CONFIRM] paramétert - az összes file-t törölné. Így minden egyes file után visszakérdez, hogy törölje-e le.

HELP

A képernyőre listázza a kiválasztott témához kapcsolódó segítő információt. Az információ angol nyelvű.

Szintaxis: **HELP** {<téma>} {<résztéma1>,...} {[<param>]}

A <téma> az alábbi témák egyike lehet: COMMANDS, CNTRLCHARS, COPYSYS, DATE, DEVICE, DIR, DUMP, ED, ERASE, FILESPEC, GENCOM, GET, HELP, HEXCOM, INITDIR, LIB, LINK, MAC, PATCH, PIP(COPY), PUT, RENAME, RMAC, SAVE, SET, SETDEF, SHOW, SID, SUBMIT, TYPE, USER, XREF. Az egyes résztémákat a témák kérésekor tudhatjuk meg. Ha nem adjuk meg a témát, akkor egy témaválasztó képernyőt kapunk.

A <param> értéke vagy **LIST** vagy **NOPAGE** lehet. Az előbbi esetben 24 sor kiírása után a kiírás felfüggesztődik, s csak RETURN-re folytatódik. A NOPAGE használatakor a kiírás - míg van információ - folyamatos.

HEXCOM

HEX típusú file-ból közvetlenül betölthető COM típusú file-t készít.

Szintaxis: **HEXCOM** <filenév>

A <filenév> HEX típusú, szabvány INTEL formátumú file neve kell hogy legyen. éppen ezért a típust nem is kell megadni. Az eredményül kapott file neve ugyanaz, de COM típusú lesz.

Példák:
(i) HEXCOM B:MASOLO

A HEXCOM hibátlan működéséhez az kell, hogy a B: lemezen egy MASOLO.HEX típusú file legyen. Ebből hoz létre egy közvetlen betölthető MASOLO.COM file-t, ugyancsak a B: lemezen.

INITDIR

Átalakítja a lemez katalógusát, hogy az az idő paramétereket is tárolni tudja.

Szintaxis: **INITDIR** {<lemez>}

<lemez> annak a lemezegységnek a betűjele, amelyik a szóbanforgó lemezt tartalmazza. Ha elmarad, a rendszer maga kéri be. A program ellenőrzi, hogy a lemez katalógusa nincs-e már átalakítva. Ha igen, akkor kérésre visszaalakítja. Ha nem, akkor kérésre átalakítja.

Példa:
(i) INITDIR A:

PIP

File-ok és különböző logikai perifériák közti adatátvitelre szolgáló program. A régebbi rendszerekben még COPY volt a neve. A PIP a peripheral interchange program rövidítéséből származik.

Szintaxis:
PIP { <cél>=<forrás1> {[<param>]} {,<forrás2> {[<param2>]}},...}

A <forrás>-ok és a <cél> is tetszőleges file vagy logikai eszközök lehetnek. Ha logikai eszközt adunk meg, az természetesen olyan kell hogy legyen, hogy a feladat egyáltalán végrehajtható legyen rajta. Ha több forrást adunk meg, akkor a PIP ezekből egyetlen <cél>-t hoz létre, úgy, hogy összefűzi a specifikált file-okat. Ha egyetlen paramétert sem adunk meg, akkor a PIP egy * promptot ad, s úgy várja a paraméterek beírását.

A <param> paraméterlista az alábbiakat tartalmazhatja:

A = Csak azokat a file-okat másolja a rendszer, amelyek módosítva lettek.

C = Minden egyes másolás előtt megkérdezi a rendszer, hogy tényleg másolni akarjuk-e. Főleg * és ? használatakor érdekes.

Dn = Az n-nél hosszabb rekordok végét levágja.

E = Echo. Az átmásolt file minden egyes karaktere megjelenik a CONOUT:-on is. Csak karakteres file esetén használható.

F = kiszűri a lapdobás (ØCH) karaktereket. Ezek a karakterek nem kerülnek bele a <cél> file-ba.

Gn = a <cél> file felhasználói száma n lesz.

H = szabvány INTEL formájú HEX adatok másolása. A program ellenőrzi a formátumot, s ha eltérést talál, azt a képernyőn jelzi.

I = kihagyja a :ØØ rekordokat a HEX típusú file-ból.
Automatikusan beállítja a H paramétert is.

L = a forrásban levő nagybetűket kisbetűkre átalakítva írja a célra. Logikai eszközök esetén is használható.

N = a forrásból beolvasott sorokhoz sorszámokat ad a program, amiket a célra ki is ír. A sorszámozás 1-gyel kezdődik és egyesével nő. A sorszámot egy kettőspont is követi. Ha N2-t adunk meg, akkor a sorszám elé Ø-k is íródnak és a sorszám után egy TAB-ot ír a program. Ha a T paramétert használjuk, akkor a TAB-okat szóközökké alakítja a rendszer.

O = ún. object file másolása. A másolandó file olyan hexadecimális karaktereket is tartalmazhat, amelyek másképpen vezérlő karakterek (pl. file vége jel). Ennek ellenére a CP/M a file fizikai végéig átmásolódik. Az összefűzésnél az esetleges <CTRL-Z> karakterek kimaradnak.

Pn = lapdobás beállítása. Ha a P paramétert használjuk, akkor a PIP minden n-ik sor után egy lapdobás karaktert másol át.

Q<sztring> = a másolás a <sztring> karaktersorozat megtalálásakor végetér. A paraméter beírásánál a <sztringet> <CTRL-Z>-vel kell befejezni.

R = SYS típusú file másolása. Normál körülmények közt a PIP nem másolja a SYS típusú file-okat. A paraméter hatására azonban mégis átmásolja.

S<sztring> = a másolás a forrás file <sztring> karaktersorozatától kezdődik. A parancs felírásakor a <sztring>-et egy <CTRL-Z> karakterrel kell befejezni.

Tn = a paraméter megadása után a PIP a TAB karaktereket szóközökkel egészíti ki. A rendszer úgy tekinti, hogy minden n-ik helyen van tabulátor pont.

U = minden betűt nagybetűként másol át.

V = ellenőrzi, hogy a másolás helyesen történt-e meg. A <cél> eszköz csak lemezegység lehet.

W = Ha a <cél> egy már létező és R/O attributumú file, a PIP megkérdezi, hogy felül akarjuk-e írni. Ha W paramétert használjuk, a kérdés elmarad, s a másolás megtörténik.

Példák:

```
  (i)     PIP E:PROBA.TXT=A:PROBA.TXT
          PIP E:=A:PROBA.TXT
          PIP E:=PROBA.TXT
          PIP E:[G4]=A:PROBA.TXT
 (ii)     PIP VEGSO.TXT=PROBA.TXT
          PIP E:VEGSO.TXT=PROBA.TXT
(iii)     PIP B:=A:*.* [C]
          PIP B:=*.* [G6 C]
          PIP B:[G4]=*.TXT [G6 C]
 (iv)     PIP LST:=PROBA.TXT [N2 T8]
  (v)     PIP LST:=CON:
 (vi)     PIP FILE.$$$=CON:
```

Az (i) a PIP leggyakoribb használatát mutatja be. Az A: lemezen levő file-t átmásoljuk az E: lemezre. A második sor mutatja, hogy ha a név nem változik, akkor a <cél>-ből el is maradhat. A harmadik formát csak akkor használhajuk, ha az aktuális lemezegység éppen az A:. A negyedik formában az átmásolt file a 4-es felhasználói számot kapja.

A (ii) példa azt mutatja, hogy a másolt file neve nem kell, hogy azonos legyen a másolandóval. Ha nem ugyanarra a lemezre akarjuk másolni, akkor a <cél>-nál meg kell jelölni az új lemezegységet.

A (iii) alatti parancsok a * karakter használatát mutatják. Az első alakban használtuk a C paramétert. A lemezen levő file-ok közül csak az másolódik át, amire Yes-szel válaszolunk. A második alakban csak a 6-os felhasználói számhoz tartozó file-okat másolja át a PIP. A harmadik alak ugyanez, azzal a különbséggel, hogy az átmásolt file-ok a 4-es felhasználói számot kapják.

A (iv)-(vi) példák azt mutatják, hogy logikai perifériák és/vagy lemezes file-ok közt is képes a PIP 'másolni'.(iv)-ben a PROBA.TXT nevű file-t a listázó perifériára másolja a rendszer. Az N2 paraméter hatására előnullázott sorszámok kerülnek minden sor elé, s T8 miatt a TAB-okat szóközökre cseréli a rendszer.

Az (v) példában a konzolon beírt karaktereket másolja a PIP a listázó perifériára. Az adatbevitelt <CTRL-Z>-vel kell befejezni. A (vi) példa a konzolról bevitt adatokat a FILE.$$$ file-ba viszi. Az adatbevitelt ugyancsak <CTRL-Z>-vel kell befejezni.

RENAME

File-ok átnevezésére használható.

Szintaxis: **RENAME** {<filespec-új>=<filespec-régi>}

A <filespec-régi> nevű file-t <filespec-új>-ra nevezi át. A filenév tartalmazhat * karaktert is.

Példák:

```
  (i)     RENAME PROBA.BAS=SZOVEG.BAS
          RENAME B:PROBA.BAS=SZOVEG.BAS
 (ii)     RENAME GRID*.TXT=KUK*.TXT
(iii)     RENAME
```

Az (i) példában az aktuális lemezegységen léteznie kell a SZOVEG.BAS file-nak. Az utasítás végrehajtása után ennek a neve PROBA.BAS lesz. A második sor mutatja, hogy ha nem az aktuális lemezen van a file, akkor a lemez azonosítóját - esetünkben B: - meg kell adni.

A (ii) példa a * karakter hatását mutatja. Az összes KUK-kal kezdődő, TXT típusú file-t átnevezi a program. KUKAC.TXT új neve GRIDAC.TXT, KUKI.TXT-é pedig GRIDI.TXT lesz.

A (iii) a RENAME paraméter nélküli használatát mutatja. A program megkérdezi:

Enter New Name:PROBA.BAS
Enter Old Name:SZOVEG.BAS

majd átnevezi a file-t.

SAVE

A memória adott részének kiírása lemezes file-ba.

Szintaxis: **SAVE**

A memóriában levő program elmentéséhez előbb ki kell adni a SAVE parancsot, majd futtatni a programot. A program megállása, vagy abortálása után a SAVE elindul, s kéri a file nevét, ahová a memória tartalmát el kell menteni. Ezután a memória kezdő és végcímét kérdezi, majd megtörténik a memória adott részének elmentése:

Enter file(type RETURN to exit): proba.com
Beginning hex address 100
Ending hex address 2000

Ha a fenti válaszokat adtuk, akkor a memória 100H-2000H közti része íródik ki a PROBA.COM file-ba.

SET

A lemez és a file-ok attributumait állítja be, vagy mutatja meg.

Szintaxis:
SET {<lemez>} [<param>]
SET <filespec> [<param>]

A parancs első alakjában a lemez paramétereit állíthajuk be, a második esetben a lemez egyes file-jainak attributumait.

Lemez attribútumok

RO = csak olvasható lesz a lemezegység.
RW = a lemezegység újra írhatóvá válik.
NAME=<név> = a lemez neve <név> lesz
PASSWORD=<jelszó> = a lemez neve a <jelszó> jelszóval védett lesz. Ha a <jelszó> egyetlen karaktert sem tartalmaz, akkor a védelem megszűnik.
PROTECT=ON = bekapcsolja a lemez adatvédelmi rendszerét. Az egyes file-ok csak ezután védhetők.
PROTECT=OFF = kikapcsolja a lemez adatvédelmi rendszerét. Ezzel valamennyi file védelme megszűnik!
DEFAULT=<jelszó> = Valamennyi jelszó kéréskor a rendszer először ezt a jelszót vizsgálja meg, s csak ha ez nem jó, akkor kér jelszót. Az üres sztinget beírva hatástalaníthatjuk.
CREATE=ON = a lemezes file-ok létrehozásának dátumát beírja a katalógusba a CP/M. Ehhez előbb az INITDIR-rel újra kell formázni a lemez katalógusát.
ACCESS=ON = a lemezes file-ok utolsó olvasásának dátumát beírja a katalógusba a CP/M. Ehhez előbb az INITDIR-rel újra kell formázni a lemez katalógusát.
UPDATE=ON a lemezes file-ok utolsó írásának dátumát beírja a katalógusba a CP/M. Ehhez előbb az INITDIR-rel újra kell formázni a lemez katalógusát.

Megjegyezzük, hogy a CREATE és a másik két időjelző (ACCESS és UPDATE) nem használható egyszerre.

File attribútumok

RO = a file csak olvasható.
RW = a file írható és olvasható.
SYS = rendszer file
DIR = felhasználói file
ARCHIVE=ON = beállítja a file archiválási jelzőjét.
ARCHIVE=OFF = kikapcsolja a file archiválási jelzőjét.
Fi=ON = bekapcsolja a file i-ik felhasználói attributum jelzőjét. i csak 1-4 lehet.
Fi=OFF = kikapcsolja a file i-ik felhasználói attributum jelzőjét. i csak 1-4 lehet.
PASSWORD=<jelszó> = a file jelszava <jelszó> lesz. Ez önmagában még nem kapcsolja be az adatvédelmet.
PROTECT = READ = a jelszót a CP/M olvasáskor, íráskor, másoláskor, törlésnél és átnevezésnél egyaránt kéri.
PROTECT = WRITE = a jelszót a CP/M íráskor, törlésnél és átnevezésnél kéri.
PROTECT = DELETE = a jelszót csak törlésnél és átnevezésnél kell megadni.
PROTECT = NONE = a file többé nem védett.

Példák:

```
  (i)     SET A: [NAME=KONYV.TXT]
          SET A: [PASSWORD=ENIS, PROTECT=ON]
 (ii)     SET BEVEZETES.TXT [PASSWORD=VIGYAZAT, PROTECT=WRITE]
(iii)     SET *.* [PROTECT=DELETE]
          SET [DEFAULT=NEMHAGYOM]
 (iv)     INITDIR A:
          SET A: [CREATE=ON]
```

Az (i) példa két parancsának a hatására az A: lemez neve KONYV.TXT lesz, s bekapcsoljuk az adatvédelmét. A lemez jelszava ENIS lesz.

A (ii) példában a BEVEZETES.TXT file-ra bekapcsoljuk az adatvédelem WRITE fokozatát. A jelszó VIGYAZAT lesz.

(iii)-ban valamennyi file védve lesz a véletlen törlés ellen. A default jelszót NEMHAGYOM-nak adjuk meg. Ezután az összes olyan file törölhetö - a rendszer visszakérdezése nélkül - aminek NEMHAGYOM a jelszava.

A (iv) példa átformázza a lemez katalógusát, majd jelzi, hogy a létrehozási dátumokat be kell írni a katalógusba.

SHOW

A lemez egyes paramétereit írja ki a konzolra.

Szintaxis: **SHOW** {<lemez>}{[<param>]}

<lemez> valamilyen lemez azonosító betű, a kettősponttal együtt. A <param> egymástól vesszővel elválasztott paraméterek listája. A következő paraméterek használhatók:

SPACE = kiírja a lemez szabad területének nagyságát.
LABEL = kiírja a lemez nevét, s hogy miyen védelmet és időparaméterezést használ a lemez.
USERS = kiírja az érvényes felhasználói számot, s hogy ahhoz hány file tartozik.
DIR = kiírja hány szabad katalógus hely van még.
DRIVE = kiírja a használt meghajtó fizikai paramétereit.

TYPE

Egy szöveges file-t a képernyőre listáz.

Szintaxis: **TYPE** <filespec> {[<param>]}

<filespec> a kilistázandó file, tartalmazhat * és ? karaktereket is. Ekkor az összes megfelelő nevű file-t kilistázza a program. <param>-nak összesen két értéke lehet:

PAGE = a kiírás 24 soronként (pontosabban a DEVICE-ban definiált sorszámnak megfelelően) félbeszakad. A listázás a <RETURN> megnyomásával folytatható.
NOPAGE = a listázás folyamatos.

Példa: (i) TYPE SZOVEG.TXT [NOPAGE]

5. fejezet

A lemezegység használata

5.1 A lemezegység felépítése

Ebben a fejezetben a C-128 számítógéphez csatlakoztatható lemezegységek használatát ismertetjük. A C-128-hoz többfajta lemezegység is csatlakoztatható, ezek sorban a következők:

1541	egymeghajtós	egyoldalas	GCR
1570	egymeghajtós	egyoldalas	GCR, MFM
1571	egymeghajtós	kétoldalas	GCR, MFM
4040	kétmeghajtós	egyoldalas	GCR

A legtöbb lemezegység - lemezoldalanként és meghajtónként - egy író/olvasó, illetve egy törlő fejjel rendelkezik, amelyeket egy léptető motor segítségével lehet a lemez megfelelő sávja fölé pozicionálni. A motor mindig sugárirányban mozgatja a fejeket, egy-egy lépés nagysága megközelítőleg 1/40 inch. A lemezen annak a sávnak a szélessége, amelyen az írás végül is történik, körülbelül 1/80 inch.

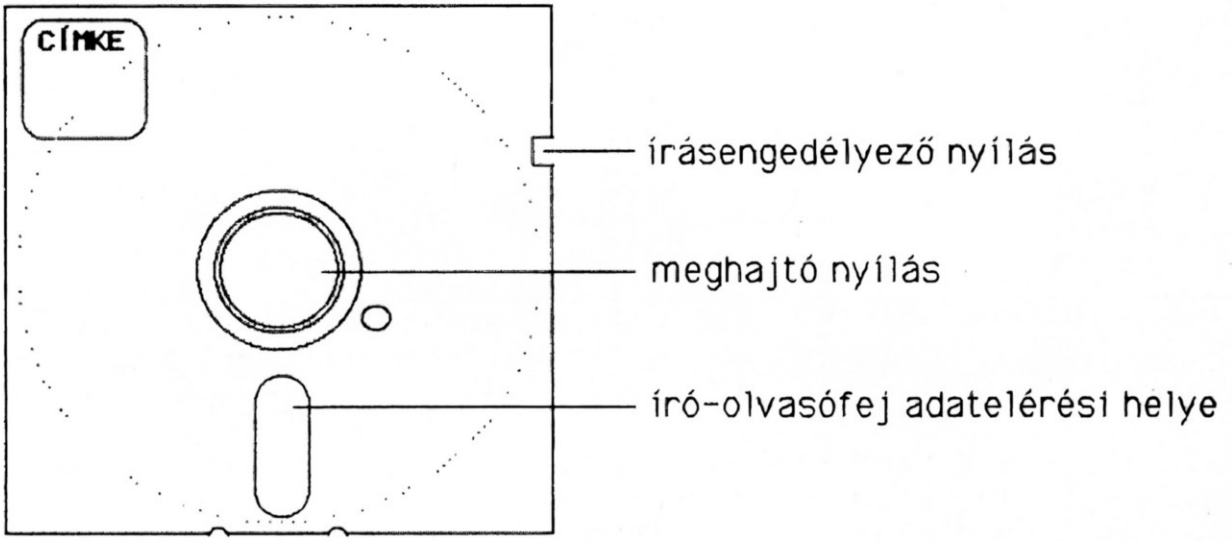
A 1570/71-es lemezegységek két újdonsága az MFM formátumú lemezek kezelése és egy, az eredetinél **nyolcszor gyorsabb** átviteli lehetőség a számítógép és a lemezegység között. Ez utóbbi lehetőséget azonban csak 128-as üzemmódban tudjuk használni.

A lemezegység számos áramkört, érzékelőt tartalmaz, amelyek lehetővé teszik a léptetőmotor vezérlését, az írást engedélyező rés meglétének ellenőrzését, a lemezegység adott sávjának olvasását stb. A lemezegység egy önálló **operációs rendszert (DOS)** tartalmaz, amelyik a lemezegység kezelésén túlmenően a C-128-cal való kommunikációt is elvégzi. Ennek nagysága a 1541-es egység esetén 16K, míg a 1570/71-es lemezegységek DOS-a 32K nagyságú. A DOS használata nagyfokú önállóságot eredményez, ami lehetővé teszi, hogy a központi egységtől függetlenül is dolgozhasson a lemezegység. Elérhető például, hogy a lemezegység egy file-t a sornyomtatón listázzon, miközben a központi egység valami más feladatot végez. A számítógép csatlakozóját úgy tervezvezték, hogy egyszerre négy lemezegység és egy nyomtató legyen rácsatlakoztatható. (Ebben az esetben természetesen a második, illetve a további lemezegységek hardver számát - ami általában 8 - módosítani kell.)

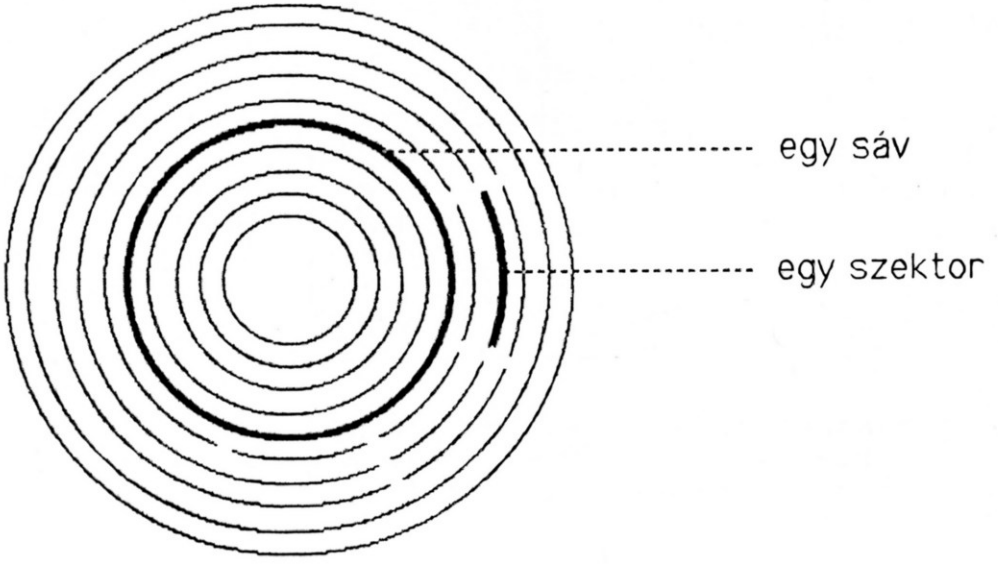
A lemezegységek és a számítógép csatlakoztatására az ún. IEC buszt használja a rendszer. Ez meglehetősen lassú, ezért az újabb fejlesztésű 1570/71-es lemezegységek egy gyors átviteli lehetőséggel is rendelkeznek, ami azonban csak teljes blokkok írására és olvasására alkalmas. Ezért a C-128-as mód a gyors átvitelt csak a programfile-ok töltésére/ellenőrzésére használja. Igazi előnyeit a CP/M haszálja, amelyik adatátvitele eleve blokk orientált. Ha a 1541-es helyett egy 1570-es meghajtót használunk az mintegy hét-nyolcszorosára gyorsítja a CP/M-et. C-64-es üzemmódban nem, (s ezért Commodore 64-en sem) használhatjuk a gyors átvitelt, mert az ehhez szükséges hardver hiányzik.

A C-128 BASIC V7.0 interpreter a korábbi Commodore gépekhez képest számos, a lemezegység kezelését megkönnyítő BASIC utasítást tartalmaz. Természetesen ezeket célszerű használni a bonyolultabb DOS parancsok helyett. Ha azonban olyan programot szeretnénk írni, amelyik valamennyi Commodore gépen fut, akkor nem használhatjuk ezeket az utasításokat.

Az alábbi ábra egy hajlékonylemez (floppy) vázlata. A négyzet a lemezt magába foglaló borítékot szemlélteti, a kör alakú rész a mágneses adathordozó valódi (de nem látható) szélét mutatja. Az írásvédő rés a lemezek fizikai írásvédelmét szolgálja. Amennyiben az írásvédő rést **kivágtuk**, a DOS nem akadályozza meg a lemezre való **írást**. Ha a rés **hiányzik** (pl. leragasztottuk), a DOS **nem engedi** meg a lemezre való írást.



A DOS (normál, azaz GCR üzemmódban) a lemezre **sávonként** (track) írja az információt. A sávok koncentrikus körgyűrűk. A lemezen 35 sáv helyezkedik el, amelyeket kívülről 1-gyel kezdődően számoz meg a rendszer. A sávra szektoronként (sector) egy blokknyi adat (256 byte) információ kerül felírásra úgy, ahogy ezt az alábbi ábra szemlélteti:



Egy szektorba a következő információk kerülnek:

SYNC	08	ID1	ID2	TRACK	SECTOR	ELL.OSSZEG	GAP1
SYNC	07	256 byte adat				ELL.OSSZEG	GAP2

A DOS rendszer automatikusan gondoskodik a lemezre való írásról/olvasásról. A Commodore cég lemezegységei - eltérően a legtöbb gyártótól - a külső (alacsonyabb sorszámú) sávokban sűrűbben írnak, és így ott több szektor fér el:

sáv(track) sorszáma	szektorok indexe	szektorok száma
1-17	0-20	21
18-24	0-18	19
25-30	0-17	18
31-35	0-16	17
36-52	0-20	21
53-59	0-18	19
60-65	0-17	18
66-70	0-16	17

Az 1-35 sávok a lemez egyik, míg a 36-70 sávok a másik oldalán találhatók. Ezért ez utóbbiak használatára csak a 1571-es lemezegység esetén van lehetőség. A fenti adatokból is látható, hogy a lemez egy oldalára összesen 683 blokknyi információ fér el. Ezek közül a DOS oldalanként 19 blokkot szervezési célokra használ, ezért egy lemezoldal tárolókapacitása 168566 byte (664 blokk). Kétoldalas lemezegység esetén 349696 byte (1328 blokk).

A lemezegység és a lemezek megfelelő karbantartása esetén egy bit meghibásodásának a valószínűsége igen kicsiny (kb. 1E-10). A DOS rendszerhibák ennél gyakoribbak. A DOS, ha hibát észlel, az író/olvasó fejet a lemez széléig lépteti, majd újból megkísérli az olvasást. Ezt az eljárást tízszer ismétli, és csak azután ad hibajelzést.

1570/71-es lemezegység használatakor lehetőség van MFM formátumú lemezek írására és olvasására is. Ebben az esetben a tárolókapacitás megváltozik:

Szektorméret	Szektor/sáv	Teljes tárolókapacitás
128 byte	26	133120 byte/oldal
256	16	163840 byte/oldal
512	9	184320 byte/oldal
1024	5	204800 byte/oldal

Az MFM formátum használatára közvetlenül csak a CP/M alatt nyílik lehetőség. Ha C-128-as üzemmódban szeretnénk használni, akkor magunknak kell megírni a szükséges - gépi kódú - rutinokat.

5.2 Tárolási alapelvek

A DOS a lemezen tárolt adatokról nyilvántartást (katalógus) vezet, ami rögziti, hogy a lemez mely sávjai és szektorai tartalmaznak információt; milyen programokat, adatfile-okat tárolunk a lemezen stb. A DOS két módot kínál a lemezre való írásra/olvasásra. Az első esetben a lemez **katalógus**ában (**directory**) szereplő file-okat használjuk, a második esetben közvetlenül az általunk kiválasztott blokkba írunk (vagy onnan olvasunk ki) információt.

A DOS a lemezen tárolt file-ok kezeléséhez szükséges információt a 18. (illetve az 53.) sáv 0. szektorában kezdődő katalógusban tárolja. Ezek az információk a következők:

a/ a lemez neve és azonosító száma;
b/ a lemez formátuma;
c/ minden egyes, a lemezen levő file
 neve
 típusa
 blokkjainak a száma
 törölt-e vagy sem?
 védett-e?
 a file első blokkjának sáv/szektor száma

A katalógusra a különböző utasításokban a $ jellel lehet hivatkozni. A katalógus egy speciális program file-nak is tekinthető, amelyik például a LOAD"$",8 utasítással betölthető a memóriába, majd a LIST paranccsal kilistázható. A CATALOG, vagy DIRECTORY BASIC utasításnak ugyanez a hatása, azzal a különbséggel, hogy a memóriában levő program nem megy tönkre. A blokkok foglaltságának jelzésére 140 (kétoldalas lemezegység esetén 2x140) byte szolgál a katalógus elején. A katalógusnak ezt a részét **BAM**-nak hívjuk az angol elnevezés (**block availability map**) alapján.

A file-ok azonosítására a nevük szolgál. Normál körülmények közt egy lemezen két azonos nevű file akkor sem lehet, ha különböző a típusuk. A file-ok nevét a BASIC utasításokban rövidíthetjük. Ha egy file név *-ra végződik, akkor azt a lemezegység az azzal a névvel kezdődő első file-lal azonosítja. A névben szereplő kérdőjelek helyén tetszőleges karakter állhat.

Az ugyanahhoz a file-hoz tartozó blokkokat a DOS összeláncolja. Ez azt jelenti, hogy a blokk első két byte-ja mindig a file következő blokkjának sáv és szektor számát adja meg. A file-hoz tartozó első blokk adatait a katalógusban találjuk meg. Mivel 0. sorszámú sáv nincs, ezért a 0. sorszámú sáv a file utolsó blokkját jelenti. Ebben az esetben a 2. byte az adott blokkon belül az utolsó, még a file-hoz tartozó byte sorszámát jelenti.

Az ugyanahhoz a sávhoz tartozó blokkok foglaltságát 4 byte jelzi. Az első byte a sáv (track) szabad byte-jainak számát adja meg, az azt követő 24 bit pedig az egyes blokkok foglaltságát jelzi. A 0 bit foglaltat, az 1 bit szabadot jelent. A nem létező szektoroknak megfelelő utolsó bitek mindig nullák.

5.3 A lemezegység vezérlése

A lemezzel történő adatcseréhez a filenév megadásán túl két további adatot kell megadnunk, amelyek mindegyike technikai jellegű, és az adatcserét nem befolyásolja. A BASIC kiíró/beolvasó utasításai az egyes file-okra egy-egy számmal hivatkoznak, amelyeket a file megnyitásakor kell megadnunk. Ezek a **logikai file számok**, amelyek értéke az 1-255 intervallumba kell hogy essen. A 0-ás file szám használatát a BASIC szintaktikus hibának tekinti. A másik ilyen – a ki/beviteli műveletet – szabályozó technikai adat a **csatornaszám**, aminek az értéke a 0-15 intervallumba kell hogy essen. A csatornaszámot a lemezegység használja a file-ok azonosítására. A file nevére csak a megnyitásakor van szükség, attól kezdve a logikai file számot, illetve a csatornaszámot használhatjuk az azonosítására (meghatározott és a későbbiekben ismertetett szabályok szerint).

A DOS a 0., 1. és 15. csatornákat speciális célokra használja. A 0. és 1. csatornákat a SAVE és LOAD utasítások használják. Adatok továbbítására a 2.-14. csatornákat használhatjuk. A 15. csatornát **hiba** vagy **parancs csatorná**nak hívjuk. Ezen keresztül adhatunk parancsokat a DOS-nak, illetve kaphatunk információt a lemez állapotáról. Ha a lemezegység hibát észlel, azt az előlapon látható **piros lámpa villogása** jelzi. Előfordulhat, hogy a BASIC **nem** ad hibajelzést. Ilyenkor csak a hibacsatorna olvasásával kaphatunk információt arról, milyen hiba is történt.

A 15. csatornát általában az **OPEN 15,8,15** utasítással nyitjuk meg. Ennek hatására a BASIC nyilvántartásba vesz egy 15-ös file számú file-t, amelyik a lemezegység 15. csatornáját használja. Ha a PRINT# utasítással ebbe a file-ba írunk, az nem íródik a lemezre, hanem a DOS parancsnak tekinti és végrehajtja. Ha ebből a csatornából olvasunk, akkkor a lemezegység állapotáról szóló információt kapunk.

A C-128-as mód BASIC V7.0-ás utasításai közt szerepelnek a 15. csatorna parancsai is. Ebben az esetben nem kell megnyitni a 15. csatornát, a BASIC ezt maga elvégzi.

Ha duál lemezegységet használunk, akkor lehetőség van a meghajtó jelölésére is, amit a 0: és 1: használatával valósíthatunk meg, ha a parancsban file nevet adtunk meg! Ha nem (pl. NEW DOS-parancs), akkor a parancs után egyszerűen be kell írni a 0-t vagy az 1-et:

```
LOAD "1:PROGRAM",8
PRINT#15,"NEW0"
```

Lemezek formázása

Valahányszor egy új, még eddig nem használt lemezt akarunk használni, a lemezt először formázni kell. A DOS ilyenkor elhelyezi a megfelelő szinkronizációs jeleket, felírja az üres katalógust stb. Az utasítás alakja:

```
PRINT#15,"NEW:<név>,id"   vagy rövidítve:
PRINT#15,"N:<név>,id"
```

A <név> karaktersorozat a lemez neve lesz. A név csak a katalógusba kerül beírásra, míg az id **kétkarakteres azonosító** valamennyi blokkba. Ha két kiíró utasítás között kicseréljük a lemezt, a DOS érzékeli, hogy új id azonosítójú lemezzel dolgozik, és hibát jelez. A fenti utasítás PRINT#15,"N:<név>" alakja **csak** a katalógust törli, de nem formázza újra a lemezt, és csak a nevet változtatja meg.

A NEW DOS-parancs természetesen használt lemezekre is kiadható, de ebben az esetben a rajta levő információ teljes egészében elvész. Használata előtt tehát mindig győződjünk meg róla, hogy a rajta levő adatokra már tényleg nincs-e szükségünk.

Ugyanezt a feladatot elvégezhetjük a HEADER BASIC utasítás segítségével is.

A lemezegység inicializálása
Elsősorban DOS rendszerhiba vagy BASIC programhiba után előfordulhat, hogy a lemezegység pufferében tárolt BAM nem egyezik meg a lemezen levő BAM-mal. Ez meglevő file-ok felülírását eredményezheti. Ilyen esetben célszerű a lemezegységet inicializálni, melynek hatására a DOS lezárja a megnyitott file-okat és újra olvassa a BAM-ot.

PRINT#15,"INITIALIZE" vagy rövidítve:
PRINT#15,"I"

Ugyanezt a feladatot a DCLEAR BASIC parancs segítségével is elvégezhetjük.

File-ok törlése
Lehetőség van egy vagy több file törlésére. A törlés nem jelenti a fizikai törlést, csupán a katalógusban jelzi egy bit, hogy a szóban forgó file már nem létezik. Amikor legközelebb egy új file-t hozunk létre, az írja felül a lemez megfelelő részeit. Az utasítás alakja:

PRINT#15, "SCRATCH:<név>" vagy
PRINT#15, "S:<név>"

A névben szereplő * illetve ? karakterek jelentése speciális. A * karakter csak a név utolsó karaktere lehet. Ebben az esetben **az összes, azzal a névvel kezdődő file törlődik.** Ha a

PRINT#15,"S:GRID*"

utasítást adjuk ki, akkor törlődnek a következő programok: GRID, GRIDBOOT, GRIDRUNNER stb. A kérdőjel azt jelenti, lényegtelen, hogy azon a helyen milyen betű áll. Ha a katalógus tartalmazza a PEK, POK, PIK nevű programokat, akkor a PRINT#15, "S:P?K" valamennyit törli.

A törlésre külön BASIC parancs szolgál, amit ugyancsak SCRATCH-nek hívnak. Használatához ugyancsak nem kell megnyitni a 15. csatornát, a BASIC interpreter ezt maga elvégzi. A fenti DOS parancsokat az alábbi BASIC utasításokkal lehet helyettesíteni:

SCRATCH "GRID*"
SCRATCH "P?K"

A lemez újraszervezése
Ha egy lemezt már sokat használtunk, akkor az ugyanahhoz a file-hoz tartozó blokkok össze-vissza helyezkednek el a lemezen, és a DOS igen lassan tudja csak olvasni őket. Lehetnek a lemezen az OPEN utasítással megnyitott, de szabályosan le nem zárt file-ok is. A lemez rendbetételére a

PRINT#15,"VALIDATE" vagy rövidítve a
PRINT#15,"V"

DOS parancsot használhatjuk. Az utasítás hatására a DOS újraszervezi a lemezeken levő adatokat, anélkül, hogy azok tartalmát megváltoztatná. A VALIDATE DOS-parancs törli a közvetlen elérésű file-ok blokkjait, ezért olyan lemez esetén, amelyik tartalmaz ilyent is, semmiképp se használjuk.

Ugyanez a feladat oldható meg a COLLECT BASIC utasítás segítségével.

File-ok átnevezése
Néha szükség lehet arra, hogy egy file-nak megváltoztassuk a nevét. Ezt a következő utasítással érhetjük el:

PRINT#15,"RENAME:<új név>=<régi név>"
PRINT#15,"R:<új név>=<régi név>"

Ugyanezt a feladatot oldja meg a RENAME BASIC parancs. Vigyázzunk, ebben az esetben előbb jön a régi, s csak azután az új név:

PRINT#15,"RENAME:LISTA=PROBA", illetve
RENAME "PROBA" TO "LISTA"

ugyanaz! Az utóbbi esetben természetsen nem kell megnyitni a 15. csatornát.

File-ok másolása és összefűzése
Lehetőségünk van egy file másolatát ugyanazon a lemezen, de más név alatt létrehozni:

PRINT#15,"COPY:<új név>=<régi név>" vagy
PRINT#15,"C:<új név>=<régi név>"

A COPY DOS-parancsot maximum négy file összefűzésére is használhatjuk. A COPY ebben az esetben csak szekvenciális file-okat tud összefűzni:

PRINT#15,"C:<új név>=<r.név1>,<r.név2>,<r.név3>,<r.név4>"

Ugyanezt a feladatot a COPY és az APPEND BASIC utasítások oldják meg. Az előbbi file-ok másolására, az utóbbi egy adott file-hoz egy másik file hozzáfűzésére szolgál.

Ha duál lemezegységet használunk, akkor lehetőség van arra, hogy a file-ok más-más meghajtóban legyenek. Ezt a tényt a parancsban a nevet megelőző 0: vagy 1: megadásával jelölni kell:

PRINT#15,"C1:LISTA=0:PROBA"
COPY "PROBA",D0 TO "LISTA", D1

A hibacsatorna olvasása

A 15. csatorna az INPUT#15,A$,B$,C$,D$ utasítással olvasható. Az A$, B$, C$, D$ értékei a DOS hibaüzenetét tartalmazzák. Az egyes változók jelentése:

A$=a hiba számkódja;
B$=a hiba megnevezése;
C$=a sáv,
D$=a szektor száma, amihez a hiba kapcsolódik.

Az A$, C$, D$ csak számjegyeket tartalmaz, ezért az INPUT utasításban aritmetikai változókkal helyettesíthetjük őket:

```
INPUT#15, E1,E2$,E3,E4
```

Ugyanezt végzi el a BASIC interpreter, amikor a DS vagy DS$ fenntartott változókra hivatkozunk. Kiolvassa a hibacsatornát, majd annak megfelelően beállítja DS, illetve DS$ értékét. Éppen ezért a C-128-as módban a fenti utasítások helyett csak a DS, illetve DS$ változókat használjuk.

5.4 Programok tárolása

Programok lemezre írására, illetve visszaolvasására speciális utasítások állnak rendelkezésre. A DOS a tárolásra és betöltésre a lemezegység 0., illetve 1. csatornáját használja, ezért ezeket adatcsatornaként nem lehet üzemeltetni.

A programfile-ok első két byte-ja a **töltési cím**, a program első karakterének a helye a memóriában. A file többi byte-ja (karaktere) a program része.

Programok betöltése

A lemezen tárolt programoknak a memóriába való betöltésére a LOAD utasítás szolgál. Alakja:

```
LOAD <név>,<egységszám> {,<mód>}
```

A <név> a **program nevét** definiáló sztringkifejezés; az <egységszám> a lemezegység sorszáma, ami általában 8. A <mód> értéke 0 vagy 1 lehet, (ha nem írjuk ki, az 0-nak felel meg). <mód>=1 esetén a program a memóriába pontosan oda íródik vissza, ahonnan a SAVE utasítással elmentettük, vagyis a töltési címtől kezdődően töltődik be a memóriába. <mód>=0 esetén a program a BASIC munkaterület első poziciójától kezdődően töltődik be.

A katalógus a LOAD"$",8 utasítással tölthető be, majd LIST utasítással írható ki a képernyőre.

A LOAD iménti alakja valamennyi Commodore gépen használható. A C-128-as módban azonban további utasítások szolgálnak a programok betöltésére:

DLOAD <név> {,U<egység>}
BLOAD <név> {,U<egység>} {,ON B<szeletszám>} {,P<cím>}

RUN <név> {,U<egység>}
BOOT <név> {,U<egység>}

A DLOAD utasítás BASIC programok betöltésére szolgál. Ez azt jelenti, hogy függetlenül attól, hogy a <név> file-ban milyen töltési cím van, a program a BASIC munkaterület elejére töltődik be. Ezzel ekvivalens a LOAD <név>,<egység> BASIC V2.0 utasítás.

A BLOAD utasítás a <név> nevű file-t a memória tetszőleges helyére betölti. Ha a P<cím> paramétert nem adjuk meg, akkor a file első két byte-ját használja töltési címként. Ez utóbbi ekvivalens a LOAD név>,<egység>,1 BASIC V2.0 utasítással. A <cím> megadására a V2.0-ban nincs lehetőség.

A RUN, illetve a BOOT BASIC, illetve gépi kódú programok betöltésére és elindítására szolgál. A BASIC program az első utasításától indul el RUN-nal. A gépi kódú program a betöltési címétől indul SYS-szel.

Programok tárolása

A memóriában tárolt programokat a SAVE utasítás segítségével menthetjük ki lemezre. Az utasítás formája:

SAVE <név>, <egységszám>

A <név> és <egységszám> jelentése ugyanaz, mint a LOAD utasításban. Az utasítás a BASIC programot másolja ki a lemezre, ahonnan azután a LOAD utasítás segítségével visszatölthető. A BASIC V7.0 további két utasítást biztosít a memória adott részének az elmentésére:

DSAVE <név> {,U<egység>}
BSAVE <név> {,U<egység>} {,ON B<szeletszám>}{,P<cím> TO P<cím2>}

Mindhárom utasítás - feltéve, hogy <név> nevű file még nincs a lemezen -, létrehoz egy új, <név> nevű, PRG típusú file-t, és a memóriában tárolt program tartalmát elmenti a file-ba. A SAVE és a DSAVE a BASIC programot, a BSAVE pedig a memória adott részét menti el. Ha a file már létezik, a mentés nem történik meg. Hibajelzést nem kapunk, csak a lemezegység piros lámpája villog. Ezt elkerülendő használni kell a '@' karaktert a név megadásában:

```
SAVE "@:LISTA",8
DSAVE "@LISTA"
BSAVE "@KEP", ON B0, P1024 TO P2047
```

Figyeljük meg, hogy a SAVE alkalmazásakor a '@' után kell még egy kettőspont is! A '@' hatására a DOS először törli a meglevő file-t, majd az elmentés végrehajtódik.

Programok ellenőrzése

A **VERIFY <név>, <egységszám>** utasítás ellenőrzi, hogy a <név> nevű file tartalma megegyezik-e a C-128 memóriájában tárolt programmal. Ha igen, akkor OK választ, ha nem, akkor ?VERIFY ERROR hibajelzést kapunk. A program elmentésének helyességét a DVERIFY BASIC utasítással is ellenőrizhetjük.

Program file-ok írása/olvasása

A programfile-ok a szekvenciális file-okhoz hasonlóan írhatók/olvashatók. Az egyes utasítások alakja és hatása megegyezik a szekvenciális file-oknál használhatókkal. Ha program file-t magunk írunk, akkor természetesen nekünk kell gondoskodni, hogy az olyan alakú legyen, amit a BASIC interpreter a töltő utasítások elvárnak.

Példák

Szükségünk lehet egy lemezen tárolt program kezdőcímének ismeretére. Mint már említettük, a kezdőcím a megfelelő file első két byte-ja. Ennek megfelelően a következő program egy tetszőleges program file kezdőcímét állítja elő:

```
10 INPUT"FILE NEV";N$
20 OPEN 2,8,2,N$+",PRG,READ"
30 GET#2,X$: IF X$="" THEN X$=CHR$(0)
40 GET#2,Y$: IF Y$="" THEN Y$=CHR$(0)
50 X=ASC(X$): Y=ASC(Y$)
60 PRINT "KEZDOCIM=";  X+256*Y
70 CLOSE 2
```

Program file-ok közvetlen írásával és olvasásával elérhetjük programrészek egymáshoz fűzését. A következő rövid program ezt a feladatot végzi el. A programunk a nagyobb gépeken meglevő APPEND programszerkesztő utasítást szimulálja. (Ne tévesszük össze a COPY DOS paranccsal! Az is összefűzi a két program file-t, **de nem módosítja a megfelelő mutatókat**!)

```
5 INPUT "ELSO FILE NEVE=";E$
6 INPUT "MASODIK FILE NEVE=";M$
7 INPUT "EREDO FILE NEVE=";F$
10 OPEN 2,8,2,E$+",P,R":OPEN 3,8,3,F$+",P,W"
20 GOSUB 900,ME=NE:GOSUB 1100:EL=0:MUT=NE
30 PRINT"KEZDOCIM=";NE:GOSUB 2000
40 CLOSE 2:OPEN 2,8,2,M$+",P,R"
50 GOSUB 900:EL=MUT-NE:MUT=NE:GOSUB 2000
60 PRINT#3,CHR$(0);CHR$(0);:CLOSE2:CLOSE 3:END
900 GOSUB 1000:Y$=X$:GOSUB 1000:NE=256*ASC(X$)+ASC(Y$):RETURN
1000 GET#2,X$:IF X$="" THEN X$=CHR$(0)
1010 RETURN
1100 Y$=CHR$(ME/256):X$=CHR$(ME AND 255):PRINT#3,Y$;X$;:RETURN
1200 FOR I=1 TO DB:GOSUB1000:PRINT#3,X$;:NEXT I:RETURN
2000 GOSUB 900:IF NE=0 THEN RETURN
2010 DB=NE-MUT-2:MUT=NE:ME=MUT+EL
2020 GOSUB 1100:GOSUB 1200:GOTO 2000
```

5.5 Adatok tárolása

Adatok tárolására két file-típust használhatunk. Ezek egyike a **szekvenciális** (SEQ) file-típus, amelyik a kazettás egységen levő SEQ file típussal egyezik meg. Ennél a típusnál lényegesen jobban használhatók a **relatív file**-ok (REL). Ez a file-típus lehetővé teszi a rekordok közvetlen írását/olvasását.

A DOS még egy ún. felhasználói file-típus használatát is biztosítja. A felhasználói file-okat a DOS 1.X verziójú változatai használták, és többé-kevésbé ugyanazt a feladatot látták el, mint a DOS 2.X verziójú változataiban a relatív file-ok. A DOS 2.6 a felhasználói file-okat gyakorlatilag nem különbözteti meg a szekvenciális file-októl.

5.5.1 Szekvenciális file-ok

A szekvenciális file-ok az adattárolás egyik legegyszerűbb formáját jelentik. A file-ba írt adatokat csak a felírás sorrendjében tudjuk visszaolvasni. Nincs mód a file adatainak módosítására. A szekvenciális file-t legegyszerűbben karakterek egymás utáni sorozatának képzelhetjük el:

```
K U T Y A F U L E , M A C S K A. . . . . ..
                        ↑

                  karakter számláló
```

A sorozat valamelyik helyére mutat a karakter számláló. Az írás és olvasás hatására a számláló előre mozog. A file elejére csak úgy tudunk visszatérni, ha lezárjuk, és újra olvassuk.

A szekvenciális file-okkal a következő műveleteket végezhetjük:

a/ nyitás/zárás;
b/ írás vagy olvasás (kizáró értelemben).

Szekvenciális file megnyitása
Mielőtt a file-t használnánk, a megfelelő OPEN utasítás segítségével meg kell nyitnunk. Ennek alakja:

BASIC V2.0:
OPEN lf,<egység>,<csatorna>,"<név>,SEQ,<mód>"

BASIC V7.0:
DOPEN#lf,"<név>" {,D<egység>} (olvasásra)
DOPEN#lf,"<név>",W {,D<egység>} (írásra)
APPEND#lf,"<név> {,D<egység>} (továbbírásra)
lf a file logikai file száma. A továbbiakban ezzel az értékkel hivatkozhatunk a file-ra. Az <egység> a lemezegység **hardver száma**, általában 8. A <csatorna> a lemezegység 2-14 sorszámú **adatcsatornáinak** valamelyike. A <név> a **file neve**, SEQ pedig a típusa. <mód> a file megnyitási **módja**, lehetséges értéke: READ, WRITE illetve APPEND. A fenti szavak első betűjükkel rövidíthetők.

A <mód> értékétől függően a szekvenciális file-t

i/ **írásra** (WRITE);
ii/ **olvasásra** (READ);
iii/ **továbbírásra** (APPEND)

nyithatjuk meg. Az i/ esetben a lemezen az adott nevű file nem létezhet. Ha létezik, és egy ugyanolyan nevű új file-t szeretnénk létrehozni, akkor a "@:<név>,SEQ,WRITE", vagy a DOPEN#1f,"@<név>" alakot kell használnunk. Ez törli a régi file-t és egy újat hoz létre ugyancsak <név> névvel.

Az ii/ esetben a lemezen értelemszerűen léteznie kell egy <név> nevű szekvenciális file-nak, amit olvasásra megnyitunk. Ha nem létezik, hibajelzést kapunk.

A iii/ esetben egy már létező szekvenciális file **írását folytatjuk.** Ennek megfelelően a file-nak léteznie kell a lemezen.

Az alábbi program bemutatja az egyes megnyitási módok hatását:

```
100 REM ************************
105 REM * DSK.SEQV2            *
110 REM ************************
115 REM * SZEKVENCIALIS FILE *
120 REM *  MEGNYITASI MODOK  *
123 REM *     BASIC V2.0     *
125 REM ************************
130 PRINT"<CLR>";
135 :
140 REM IRAS
145 REM ----
150 OPEN 2,8,2,"`:PROBA1,S,W" : PRINT "W";
155 FOR I=1 TO 3
160 PRINT#2,I : PRINT TAB(2);I
165 NEXT I
170 CLOSE2
175 :
180 REM HOZZAIRAS
185 REM ---------
190 OPEN 2,8,2,"PROBA1,S,A" : PRINT "A";
195 FOR I=4 TO 6
200 PRINT#2,I : PRINT TAB(2);I
205 NEXT I
210 CLOSE2
215 :
220 REM OLVASAS
225 REM -------
230 OPEN 2,8,2,"PROBA1,S,R": PRINT"S"; TAB(8);"R";
235 INPUT#2,X : PRINT TAB(10);X
240 IF ST=0 THEN GOTO 235
245 CLOSE2
```

```
100 REM **********************
105 REM * DSK.SEQV7           *
110 REM **********************
115 REM * SZEKVENCIALIS FILE *
120 REM * MEGNYITASI MODOK   *
123 REM *     BASIC V7.0     *
125 REM **********************
130 PRINT"<CLR>";
135 :
140 REM IRAS
145 REM ----
150 DOPEN#2,"`PROBA" : PRINT "W";
155 FOR I=1 TO 3
160 PRINT#2,I : PRINT TAB(2);I
165 NEXT I
170 DCLOSE#2
175 :
180 REM HOZZAIRAS
185 REM ---------
190 APPEND#2,"PROBA1" : PRINT "A";
195 FOR I=4 TO 6
200 PRINT#2,I : PRINT TAB(2);I
205 NEXT I
210 DCLOSE#2
215 :
220 REM OLVASAS
225 REM -------
230 DOPEN#2,"PROBA1": PRINT"S"; TAB(8);"R";
235 INPUT#2,X : PRINT TAB(10);X
240 IF ST=0 THEN GOTO 235
245 DCLOSE#2
```

Szekvenciális file-ok lezárása

Szekvenciális file-okat a CLOSE lf, illetve a DCLOSE#lf utasítások segítségével kell lezárni, ahol lf a szóban forgó file logikai file száma.

Szekvenciális file-ok írása/olvasása

A szekvenciális file-okat a PRINT#,INPUT#,GET# utasításokkal használhatjuk. Az egyes utasítások alakja a következő:

PRINT#lf {,<nyomtatási kép>}

Az utasítás a <nyomtatási kép>-ben szereplő értékeket a megadott formában kiírja az lf logikai file számu file-ba.

GET#lf,<változólista>

Az utasítás az lf logikai file számú file következő karaktereit beolvassa és a változólista elemeihez rendeli. Ismeretlen struktúrájú file-okat a legegyszerűbb a GET# utasítás segítségével olvasni.

INPUT#lf,<változólista>

Az utasítás beolvassa az lf logikai file számu file karaktereit az első kocsivissza-soremelés (kódja 13) karakterig. Az így kapott input-sor elemeit rendeli hozzá a <változólista> elemeihez úgy, mintha a billentyűzetről gépeltük volna be. A különböző mennyiségeket tehát elválasztó jelekkel (vessző, pontosvessző, kettőspont, kocsivissza-soremelés) kell elválasztani. Ha tehát különböző mennyiségeket INPUT#-al akarunk visszaolvasni, akkor a kiíráskor nekünk kell az elválasztó jelekről gondoskodnunk. Ha tehát egy file első három elemének az A$,B,C% változók értékét szeretnénk kiírni úgy, hogy az INPUT#-kal visszaolvasható legyen, akkor a következő két módon járhatunk el:

```
i/ 10 OPEN 2,8,2,"`:PROBA,S,W"
   20 INPUT A$,B,C%
   30 PRINT# 2,A$;",";B;",";C%
   40 CLOSE 2
```

Ebben az esetben a 30. sorban a program gondoskodik az elválasztó jel (,) kiírásáról.

```
ii/ 10 OPEN 2,8,2,"`:PROBA,S,W"
    20 INPUT A$,B,C%
    30 PRINT# 2,A$
    40 PRINT# 2,B
    50 PRINT# 2,C%
    60 CLOSE 2
```

Ebben az esetben a 30.-50. sorokban a PRINT# utasítások maguk gondoskodnak egy CHR(13) karakter kiírásáról.

Mindkét esetben a három értéket a következő programmal olvashatjuk vissza:

```
    100 OPEN 2,8,2,"PROBA,SEQ,READ"
    110 INPUT# 2,A$,B,C%
    120 PRINT A$,B,C%
    130 CLOSE 2
```

Az i/ használatakor a következő karakterek íródnak a PROBA nevű file-ba (A$="ILDIKO",B=2500,C%=3 esetén):

```
    ILDIKO, 2500 , 3 CHR(13)EOF
```

Ugyanez a ii/ esetben:

```
    ILDIKOCHR(13) 2500 CHR(13) 3 CHR(13)EOF
```

Az ST változó

A BASIC és a DOS külön is figyeli, hogy mikor érünk egy szekvenciális file végére. Ha egy szekvenciális file utolsó karakterét olvastuk be, az ST változó értéke 64 lesz. így lehetőség van olyan szekvenciális file olvasására is, amelyiknek nem ismerjük sem a hosszát, sem a szerkezetét. A következőkben egy olyan programot írunk, amelyik egy szekvenciális file-t karakterenként beolvas, és a karaktereket kiírja a képernyőre. Egyetlen módosítást végez, a CHR$(13) karaktert inverz < jelre cseréli. (Vizsgáljuk meg programunkkal az előző programokkal kapott file-jainkat !)

```
100 REM **********************
105 REM * DSK.SEQLISTV2      *
110 REM **********************
115 REM * SEQ TIPUSU, ASCII *
120 REM * FILE-OK LISTAZASA *
125 REM **********************
130 :
135 INPUT "<CLR>FILE NEVE=";A$
140 OPEN 2,8,2,A$+",S,R"
145 :
150 IF ST<>0 THEN CLOSE2: END
155 GET#2,A$
160 IF A$=CHR$(13) THEN PRINT "<CTRL-RVSON><<CTRL-RVSOFF>";:GOTO 150
165 PRINT A$;
170 GOTO 150
```

```
100 REM **********************
105 REM * DSK.SEQLISTV7      *
110 REM **********************
115 REM * SEQ TIPUSU, ASCII *
120 REM * FILE-OK LISTAZASA *
125 REM **********************
130 :
135 INPUT "<CLR>FILE NEVE=";A$
140 DOPEN#2,(A$)
145 :
150 IF ST<>0 THEN DCLOSE#2: END
155 GET#2,A$
160 IF A$=CHR$(13) THEN PRINT "<CTRL-RVSON><<CTRL-RVSOFF>";:GOTO 150
165 PRINT A$;
170 GOTO 150
```

5.5.2 Relatív file-ok használata

A relatív file-ok, hasonlóan a szekvenciális file-okhoz, rekordokból állnak, de ezek a rekordok ugyanolyan hosszúak. (éppen ezért nincs szükség a rekord végét jelölő speciális karakterek használatára.) Szemben azonban a szekvenciális file-okkal, ezekre a rekordokra sorszámuk alapján lehet hivatkozni. Ahhoz, hogy a DOS gyorsan megtaláljon egy-egy adott rekordot, külön blokkokra van szükség, amelyek megadják, hogy egy-egy rekord melyik blokkon található.

Relatív file-ok megnyitása/lezárása

Egy még nem létező relatív file-t a következő utasítással nyithatunk meg:

BASIC V2.0
OPEN 1f,<egység>,<csatorna>,"<név>,L,"+CHR$(<hossz>)

BASIC V7.0
DOPEN#1f,<név>, L<hossz> {,U<egység>}

lf a logikai file szám, <név> a relatív file neve, <csatorna> a programban még eddig nem használt, 2-14 adatcsatornák bármelyike lehet. Az "L" betű a relatív file-t jelenti. <egység> a lemezes egység száma, általában 8. <hossz> a relatív file **rekordhosszát** adja meg, és legfeljebb 254 lehet. A file egész 'élete' alatt ez lesz a rekordhossz. Az utasítás fenti formájában hiába használjuk a '@' alakot, a file - ha esetleg létezett - nem fog törlődni. A relatív file-ok csak a SCRATCH paranccsal törölhetők. Már meglevő relatív file-t az

BASIC V2.0
OPEN lf,<egységszám>,<csatorna>,"<név>"

BASIC V7.0
DOPEN#lf,<név>

utasítással lehet megnyitni. A READ és WRITE megjelölésre nincs külön szükség; lévén, hogy a relatív file-ba írni, olvasni egyszerre lehet.

A megnyitott file-t a CLOSE lf, illetve DCLOSE#lf utasítással zárhatjuk le.

A rekordszám beállítása

Relatív file-okat ugyanúgy írhatunk, olvashatunk mint szekvenciális file-okat, azzal a lényeges különbséggel, hogy **megadhatjuk hányadik rekord hányadik pozíciójától kezdődjék az I/O művelet.** Erre szolgál a P DOS parancs, melynek alakja:

BASIC V2.0
PRINT#hf,"P"+CHR$(<csatorna>)+CHR$(L)+CHR$(H)+CHR$(P)

BASIC V7.0
RECORD#lf,<rekord>,<P>

hf a 15. csatorna megnyitásakor használt logikai file szám, lf a relatív file logikai file száma, <csatorna> pedig a csatornaszáma (secondary adress). L és H a <rekord> rekord sorszámának alsó és felső byte-ja. A valódi sorszám tehát <rekord>=256*H+L. Végül P a rekordon belüli kezdőpozíció. Az utasítás második alakja lényegesen egyszerűbb. Nincs szükség a csatornaszám ismeretére, helyette a logikai file szám használható. Nem kell a rekord sorszámát alsó és felső byte-ra szétszedni.

Azt, hogy egy relatív file-nak hány rekordja van, a file 'élete' során a P parancsban kiadott legnagyobb rekordszám dönti el. Ha a file hosszánál nagyobb rekordszámra hivatkozik a P parancs, a DOS helyet foglal a lemezen a további rekordoknak. A még nem használt rekordok első byte-ja $FF, a többi 0. Ha egy rekordot nem írunk tele, a maradék rész 0-kal lesz feltöltve.

Relatív file-ok írása/olvasása

A GET#, INPUT# illetve PRINT# utasításokat használhatjuk a relatív file-ba való olvasásra és írásra. Az I/O művelet a legutolsó P DOS vagy RECORD BASIC V7.0 parancsban megadott pozíciótól kezdi a file-t feldolgozni. GET# az azon a pozíción levő karaktert adja vissza,

INPUT# a legközelebbi CHR$(13)-ig olvassa a file-t (akár a rekord végén túl is!), PRINT# az adott pozíciótól kezdve ír, de nem lépi át a rekordhatárt. A rekordba be nem férő karakterek elvesznek. Kivétel természetesen a file utolsó rekordja, amelyen ha túl olvasunk vagy túl írunk, akkor hibajelzést kapunk (ST=64). Mivel azonban a rekord vége általában logikai határ is, célszerűbb elkerülni az olyan I/O műveletet, amelyik átlépi a rekord határát.

Példánkban egy számokat tartalmazó file-t nyitunk meg, majd beleírunk 30 számot. A következő programrész segítségével tetszőleges számot visszaolvashatunk a lemezről, és ha akarjuk, módosíthatjuk. Egy-egy számot a relatív file egy rekordja tartalmaz.

```
100 REM ****************************
102 REM *                          *
104 REM *                          *
110 REM * PELDA RELATIVE FILE-RA   *
120 REM *                          *
130 REM * A PROGRAM 30 SZAMOT IR   *
132 REM * EGY RELATIV FILE-BA.     *
134 REM * A SZAMOK AZUTAN TETSZES*
136 REM * LEKERDEZHETOK ES HA      *
138 REM * MODOSITHATOK.            *
140 REM *                          *
142 REM ****************************
150 :
153 REM A RELATIVE FILE MEGNYITASA
200 OPEN 15,8,15:REM  PRINT#15,"S:PROBA"
210 OPEN 1,8,2,"PROBA,L,"+CHR$(14)
212 :
214 REM 30 SZAM FELIRASA
215 REM A SZAMOK MEGEGYEZNEK A REKORD SORSZAMAVAL
216 REM HA MAGUNK AKARJUK MEGADNI OKET
217 REM A SORBAN ALLO MEGJEGYZESBEN
218 REM LEVO UTASITAST KELL HASZNALNI
220 FOR I=1 TO 30
230 SZAM=I:REM INPUT "KOV.SZAM";SZAM
232 :
234 REM A REKORDSZAM BEALLITASA
240 PRINT#15,"P"+CHR$(2)+CHR$(I)+CHR$(0)+CHR$(0)
242 REM A HIBA ELLENORZESE
250 INPUT#15,E1,E2$,E3,E4:IF(E1<20) OR E1=50 THEN GOTO 270
260 PRINT E1,E2$,E3,E4:CLOSE 1: CLOSE 15:END
262 :
263 REM A SZAM KIIRASA
270 PRINT#1,SZAM;
280 INPUT#15,E1,E2$,E3,E4:IF(E1<20) OR E1=50 THEN GOTO 300
290 GOTO 260
292 :
294 REM A CSATORNAK LEZARASA
300 NEXT: CLOSE 1: CLOSE 15
302 :
304 REM MODOSITO RESZ. HA A PROBA NEVU
305 REM FILE A 30 SZAMMAL MAR LETEZIK
306 REM INNEN KELL A PROGRAMOT INDITANI
310 OPEN 15,8,15: OPEN 1,8,2,"PROBA"
312 :
314 REM MODOSITO RESZ
```

```
320 INPUT"MELYIK SZAM KELL";I
330 I=INT(I):IF (I<0) OR (I>29) THEN GOTO 320
340 PRINT# 15,"P"+CHR$(2)+CHR$(I)+CHR$(0)+CHR$(0)
345 REM A SZAM BEOLVASAS
350 INPUT#1,SZAM
360 PRINT SZAM
370 PRINT "AKARJA-E MODOSITANI(I/N)"
380 GET A$: IFA$="" THEN GOTO 380
390 IF A$="N" THEN GOTO 430
400 INPUT "UJ SZAM ERTEKE";SZAM
410 PRINT#15,"P"+CHR$(2)+CHR$(I)+CHR$(0)+CHR$(0)
420 PRINT#1,SZAM
430 PRINT"ELEG VOLT-E (I/N)"
440 GET A$: IF A$="" THEN GOTO 440
450 IF A$="N" THEN GOTO 320
452 :
454 REM A CSATORNAK LEZARASA
460 CLOSE 1: CLOSE 15: END
```

5.6 Direkt elérési mód

A bevezetőben már említettük, hogy lehetőség nyílik az egyes blokkok közvetlen írására, illetve olvasására. Ahhoz, hogy a DOS ebben az üzemmódban dolgozzon, egy # nevű pszeudo-file-t kell megnyitnunk, ami valójában a lemezegység valamelyik pufferét jelenti. (Ezenkívül mindenképp szükségünk van a 15. csatorna megnyitására is):

OPEN lf, <lemezegység>, <csatorna>, "#{<sorszám>}"

lf a file logikai file-száma, amire az I/O utasításokban hivatkoznunk kell, <lemezegység> a lemezegység hardver száma (ez általában 8); <csatorna> azt az adatcsatornát jelenti, amin keresztül az adatátvitel zajlik. értékének a 2-14 tartományba kell esnie. Az opcionális <sorszám> a puffer sorszámát adja meg. Ha elmarad, a DOS maga választja meg, melyik puffert használja. Például

```
OPEN 15,8,15:OPEN 1,8,2,"#"
```

utasítja a DOS-t a közvetlen elérésű üzemmód használatára. A lemez és a puffer közti adatátvitelt a 15. csatornán DOS-parancsokkal szabályozhatjuk. A puffert az INPUT# és a GET# utasításokkal, mint 1-es file-t olvashatjuk, illetve a PRINT# utasítással írhatunk bele.

A következőkben összefoglaljuk a közvetlen elérésű üzemmód DOS parancsait.

B-R

BLOCK-READ utasítás

A DOS-parancs lehetővé teszi egy tetszőleges blokk olvasását.

Szintaxis:
PRINT#hf,"BLOCK-READ:";<csatorna>;<meghajtó>;<sáv>;<szektor> vagy
PRINT#hf, "BLOCK-READ:<sztring>"

Az utasítás fenti alakjában a hf a 15. csatornának megfelelő logikai file-szám (ami általában 15). <csatorna> a # pszeudo-file megnyitásában szereplő csatornaszám. <meghajtó> jelen esetben mindig 0. (Olyan lemezegység esetén, amelyik két lemezmeghajtót is tartalmaz, <meghajtó> értéke 0 vagy 1 lehet.) A <sáv> és <szektor> annak a blokknak a paraméterei, amit a pufferbe szeretnénk beolvasni.Az utasítás második formájában a <sztring> négy karakterének ASCII kódja felel meg a fenti négy értéknek. A "BLOCK-READ:" szövegrész az utasításban a "B-R:" és "U1:" szöveggel helyettesíthető. (Lásd a USER utasítást.)

Következő példánk a lemez megadott blokkját olvassa be a C-16 memóriájába és írja ki a képernyőre:

```
10 OPEN 15,8,15:OPEN 1,8,2,"#"
20 INPUT "<CLR>SAV,SZEKTOR";S,SZ
30 PRINT#15,"B-R:";2;0;S;SZ
40 GET#1,X$:IF ST=64 GOTO 20
50 PRINT X$;:GOTO 40
```

A szekvenciális file-okról szóló részben említettük, hogy a szekvenciális file-ok blokkjainak első 2 byte-ja a file következő blokkjának paramétereit adja meg. Az alábbi program lehetővé teszi az egymáshoz láncolt blokkok követését:

```
10 OPEN 15,8,15:OPEN 1,8,2,"#"
20 INPUT "<CLR>SAV,SZEKTOR";S,SZ
30 PRINT#15,"B-R:";2;0;S;SZ
35 PRINT#15,"B-P:";2;0
40 GET#1,X$:IF X$="" THEN X$=CHR$(0)
50 GET#1,Y$:IF Y$="" THEN Y$=CHR$(0)
55 S=ASC(X$):SZ=ASC(Y$)
60 PRINT "SAV=";S,"SZEKTOR=";SZ
65 GET A$:IF A$="" THEN GOTO 65
70 IF S=0 THEN CLOSE1:CLOSE15:END
75 GOTO 30
```

B-W

BLOCK-WRITE utasítás

Ez a DOS-parancs lehetővé teszi egy tetszőleges blokk írását.

Szintaxis:
PRINT#hf,"BLOCK-WRITE:";<csatorna>;<meghajtó>;<sáv>;<szektor>
PRINT# hf, "BLOCK-WRITE:"+<sztring>

Az utasítás fenti alakjában hf a 15. csatornának megfelelő logikai file szám (ami általában ugyancsak 15). A csatorna a # pszeudo-file

megnyitásában szereplő csatornaszám. A <meghajtó> jelen esetben mindig 0. (Olyan lemezegység esetében, amelyik két lemezmeghajtót is tartalmaz, a <meghajtó> értéke 0 vagy 1 lehet.) A <sáv> és <szektor> annak a blokknak a paraméterei, ahová a puffer tartalmát ki akarjuk írni. Az utasítás második formájában a <sztring> négy karakterének ASCII kódja felel meg a fenti négy paraméternek. A "BLOCK-WRITE:" szövegrész az utasításban a "B-R:" és "U2:" szöveggel is helyettesíthető. (Lásd a USER utasítást.)

Első példánk az 1. sáv valamennyi (0-20 sorszámú) szektorába beírja ugyanazt az üzenetet. A szektorok sorrendjének változtatásával ki lehet próbálni, melyik esetben a leggyorsabb a DOS.

```
10 OPEN 2,8,2,"#":OPEN 15,8,15
20 DATA 0,1,2,3,4,5,6,7,8,9,10
30 DATA 11,12,13,14,15,16,17,18,19,20
40 INPUT "UZENET";M$
45 FOR I=1 TO 21:READ SZ
50 PRINT#15,"B-R:";2;0;1;SZ
60 PRINT#15,"B-P:";2;0
70 PRINT#2,M$
80 PRINT#15,"B-W:";2;0;1;SZ
85 NEXT I
90 CLOSE2:CLOSE15:END
```

(A programot csak üres, de megformázott lemezen próbáljuk ki, hogy meg ne sértsünk egyetlen programot se!)

Második példánk egy adott blokkot részben (adott pozíciótól kezdődően) módosít:

```
10 OPEN 1,8,2,"#":OPEN 15,8,15
20 INPUT "SAV,SZEKTOR";S,SZ
30 INPUT "KEZDOPOZICIO";P
40 INPUT "UZENET";M$
50 PRINT#15,"B-R:";2;0;S;SZ
60 PRINT#15,"B-P:";2;P
70 PRINT#1,M$;
80 PRINT#15,"B-W:";2;0;S;SZ
90 CLOSE1:CLOSE15:END
```

(A programot csak üres lemezen próbáljuk ki, hogy meg ne sértsünk egyetlen programot se!)

B-E

BLOCK-EXECUTE utasítás

Az utasítás hatása hasonló egy program betöltéséhez, majd futtatásához. A lemeznek az utasításban specifikált blokkja betöltődik a pufferba és a program végrehajtása a puffer elején folytatódik. Amikor a gépi kódú program egy RTS-hez ér, az a BASIC programba való visszatérést eredményezi. Ritkán használjuk, mert ehhez a DOS ROM részletes ismeretére van szükség.

Szintaxis:

```
PRINT#hf, "BLOCK-EXECUTE:";<csatorna>;<meghajtó>;<sáv>;<szektor> vagy
PRINT#hf, "BLOCK-EXECUTE:"+<sztring>
```

Az egyes paraméterek jelentése megegyezik a B-W és B-R parancsoknál leírtakkal. A "B-E:" rövidítés használata megengedett.

B-A

BLOCK-ALLOCATE utasítás

Az utasításban specifikált blokknak a BAM-ban megfelelő bit alacsony lesz, jelezve a DOS-nak, hogy azt más célokra már nem használhatja. Amennyiben a blokk már foglalt 65, NO BLOCK, S, SZ hibaüzenetet kapunk, ahol S, SZ a következő, még nem foglalt blokk sáv- és szektorszámát adja meg.

Szintaxis:
PRINT#hf, "BLOCK-ALLOCATE:";<meghajtó>;<sáv>;<szektor> vagy
PRINT#hf, "BLOCK-ALLOCATE:"+<sztring>

Az egyes paraméterek jelentése megegyezik a B-R illetve a B-W utasításban leírtakkal. (A <csatorna> megadására értelemszerűen nincs szükség.) A "B-A:" rövidítés használata megengedett.

Példánk egy szubrutint mutat be, amelyik a következő üres (nem foglalt) blokk sáv- és szektorszámával tér vissza. A 65-ös hiba figyelésén kívül még arról is gondoskodni kell, hogy a blokk - néhány kivételtől eltekintve - nem lehet a katalógus része. A program az S, SZ értékpárban adja vissza a következő szabad blokk paramétereit. E értéke a hibakódot tartalmazza. Ez 0, ha sikerült szabad blokkot találni, különben a felmerült DOS-hiba kódját tartalmazza:

```
1000 PRINT#15,"B-A:";0;S;SZ
1010 INPUT#15,E,E$,ES,EZ
1020 IF E=0 THEN RETURN
1030 IF E<>65 THEN RETURN
1040 S=ES:SZ=EZ:IF S=18 THEN S=19
1050 GOTO 1000
```

B-F

BLOCK-FREE utasítás

A DOS-parancs a B-A parancs ellentettje. A B-F parancsban specifikált blokknak a BAM-ban megfelelő bitje magas lesz, jelezve, hogy a továbbiakban a DOS azt más célokra felhasználhatja.

Szintaxis: PRINT#hf, "BLOCK-FREE:";<meghajtó>;<sáv>;<szektor>

Az egyes paraméterek jelentése megegyezik a B-R illetve a B-W utasításban leírtakkal. (A <csatorna> megadására értelemszerűen nincs szükség.) A "B-F:" rövidítés megengedett. Például a

```
PRINT#15,"B-F:";0;1;SZ
```

lemez 1. sávjának SZ-ik szektorát a DOS rendelkezésére bocsátja. (Csak akkor próbáljuk ki, ha tudjuk, hogy nem sértünk meg egyetlen programot sem!)

B-P

BUFFER-POINTER utasítás

A # pszeudo-file-hoz egy mutató is tartozik, amelyik a lehetséges (1-255-ig számozott) byte valamelyikére mutat. Az író/olvasó utasítások végrehajtása ettől a byte-tól kezdődik. A mutatót a B-P utasítás segítségével tetszőleges helyre pozícionálhatjuk. A mutató utolsó értéke, mint a blokk 0. byte-ja a lemezre íródik.

Szintaxis: PRINT#hf, "BUFFER-POINTER:";<csatorna>;<mutató>

A hf és a <csatorna> jelentése ugyanaz, mint a B-R utasításban; <mutató> a # pszeudo-file pufferének pointerét állítja át. A "B-P:" rövidítés itt is megengedett. A B-P parancsot elsősorban a B-W és B-R utasításokkal együtt használjuk. (B-W példájában már szerepelt.)

M-R

MEMORY-READ utasítás

Az utasítás lehetővé teszi a lemezegység memóriájának byte-onkénti olvasását. (Akár a RAM-ból, akár a ROM-ból olvashatunk.) Ez lehetővé teszi a DOS működésének tanulmányozását, a 0. lap megismerését stb. Minden egyes byte-ra egy új utasítást kell kiadni.

Szintaxis: PRINT#hf, "M-R:";CHR$(L);CHR$(H)

Az utasítás fenti alakjában hf a lemezegység 15. csatornájához tartozó logikai file szám; L és H a memória címének alsó illetve felső byte-ja. Az utasítást követően a szóban forgó memória címen levő byte a GET#hf, A$ utasítás segítségével olvasható, ahol hf a hiba-csatornának megfelelő logikai file szám.

M-E

MEMORY-EXECUTE utasítás

Az utasítás lehetővé teszi a lemezegység tetszőleges címén kezdődő (gépi kódú) program végrehajtását. Ez lehet egy ROM alprogram, de lehet a RAM-ban az M-W parancs segítségével megírt program is. A szintaxis megegyezik az M-R utasítás szintaxisával.

Szintaxis: PRINT#hf, "M-E:";CHR$(L);CHR$(H)

ahol L és H a kezdőcím alsó illetve felső byte-ja.

M-W

MEMORY-WRITE utasítás

A parancs lehetővé teszi - egyidőben 34 byte - beírását a DOS memóriájába. Ezeket azután például az M-E parancs használhatja.

Szintaxis:
PRINT#hf, "M-W:";CHR$(L);CHR$(H);<karakterek száma>;<byte-ok>

hf, L és H jelentése ugyanaz, mint az M-R utasításban. Ezt követi a <karakterek száma> paraméter, amelyik definiálja, hány byte-ból áll a DOS memóriájába írandó sorozat. Ezt követik a sorozat byte-jai CHR$(B) alakban. Például

```
PRINT#15, "M-W:";CHR$(0);CHR$(5);1;CHR$(96)
```

a $0500 címre egyetlen byte-ot (96) helyez el. Ez egy RTS utasításnak felel meg.

USER

Ez a DOS-parancs lehetővé teszi a DOS memóriájában tárolt bizonyos címekre való ugrást. Ezek általában további JMP utasításokat használnak, amelyek lehetővé teszik egy tetszőleges DOS rutin elérését.

Szintaxis: PRINT#hf, "<USER utasítás>"

hf a 15. csatorna megnyitásában használt logikai file szám (általában maga is 15). Az utasításhoz néha további byte-okat is el kell küldeni. A <USER utasítás> lehetséges értékeit és azok jelentését a következő táblázat foglalja össze:

USER utasítás	Jelentés
U1 vagy UA	B-R a puffer-pointer felhasználása nélkül
U2 vagy UB	B-W a puffer-pointer felhasználásával
U3 vagy UC	JMP $0500
U4 vagy UD	JMP $0503
U5 vagy UE	JMP $0506
U6 vagy UF	JMP $0509
U7 vagy UG	JMP $050C
U8 vagy UH	JMP $050F
U9 vagy UI	JMP $FFFA (=NMI vektor)
U; vagy UJ	RESET vektor
UI+	C-64 sebességének kiválasztása
UI-	VC20 sebességének kiválasztása
U0>Mm	Mód: m=0 1541, m=1 1570/71
U0>Hs	Lemezoldal: csak 1541 módban és 1571-re s=0: alsó, s=1: felső
U0>v	Egységszám: v=CHR$(4),...,CHR$(30)

Az **U0-lal kezdődő** felhasználói parancsok csak a 1570/71-es egységen hajthatók végre!

Külön szólunk az U1 és U2 hatásáról. A B-R utasítás az adott blokkot csak a blokk első byte-jaként tárolt puffer-mutató értékéig olvassa be. Ha az utasítás U1 alakját használjuk, akkor mind a 256 byte a pufferbe kerül. A B-W utasítás a puffer-pointer értékét és mind a 255 adat-byte-ot az adott blokkba kiírja. Az utasítás U2 alakja a puffert csak a lemezen levő mutató értékéig másolja vissza.

Példák:

```
  (i)     PRINT#15,"U0>M0"
          PRINT#15,"U0>H0"
 (ii)     PRINT#15,"U0>"+CHR$(9)
```

Az (i) példában a 1571-es lemezegységet egy Commodore 64 számítógéphez kötöttük, s egy olyan programot futtatunk rajta, amelyik direkt elérési módot használ a lemez kezelésére. Ebben az esetben nem használhatjuk ki a 1571-es kétoldalas írási képességét, hiszen a program ezt nem 'tudja'. Ezért az első paranccsal kiválasztjuk a 1541-es módot, majd utasítjuk, hogy a lemez alsó oldalát használja. A felső oldalt egyéb feladatokra (persze ugyancsak 1541-es módban) használhatjuk.

A (ii)-ben a lemezegységet 9-es hardver számúra módosítjuk. Ha még egy lemezegységünk van, azt erre az időre ki kell kapcsolni. A parancs végrehajtás után visszakapcsolható.

Példa: # file használata

```
1000 rem"**********************
1010 rem" # file használata   *
1020 rem" rekordok tárolására *
1030 rem"**********************
1040 :
1050 print"<CLR><LE><LE> tomeges/egyedi proba(t/e)?"
1060 get a$: if a$="" then goto 1060
1070 if a$="t" then goto 1390
1080 if a$="e" then goto 1130
1090 goto 1060
1100 :
1110 rem egyedi proba
1120 rem ############
1130 print"<CLR><LE><LE>  egyedi proba"
1140 input "<LE>  rekordhossz";ho
1150 if (ho<1) or (ho>256) then print"<FEL<FEL>";: goto 1140
1160 open 15,8,15: open 5,8,5,"#"
1170 if ds<>0 then printds$: close15 : close 5:end
1180 :
1190 print"<CLR>iras/olvas/vege"
1200 get a$: if a$="" then goto 1200
1210 if a$="o" then goto 1320
1220 if a$="v" then close 5: close 15: end
1230 if a$<>"i" then goto 1200
1240 :
1250 rem rekord bekerese es kiirasa
1260 input "x$=";x$: x$=left$(x$+"                              ",14)
1270 input "ix=";ix
1280 gosub 1930
1290 goto 1190
1300 :
1310 rem rekord beolvasasa
1320 input "ix=";ix: gosub1550
1330 printx$
1340 getkey a$
1350 goto 1190
```

```
1360 :
1370 rem tomeges proba
1380 rem ##############
1390 open 15,8,15: open 5,8,5,"#"
1400 x$="0123456789abcd":ho=14
1410 print "<CLR><LE><LE>  tomeges proba":print
                                          :print"  rekordhossz=14"
1420 :
1430 print "<LE><LE>    rekordok iras":print
1440 for ix=540 to 500 step -1
1450 : print " ";ix;"= ";: gosub 1930: printx$;s(0);sz(0);
                                              mp(0);"<BALRA>    "
1455 : print"<LE>";
1460 next ix
1470 :
1480 print "<LE><LE>  rekordok olvasasa":print
1490 for ix=500 to 540
1500 : x$="": gosub 1550
1510 : print " ";ix;"= ";x$;s(0);sz(0);mp(0);"<BALRA>  "
1515 : print"<LE>";
1520 next ix
1530 close 15: close 5: end
1540 :
1550 rem rekord beolvasasa
1560 rem ******************
1570 :
1580 mu=(ix-1)*ho :rem elso byte
1590 mr(0)=int(mu/256)
1600 mp(0)=mu and 255
1610 mr(1)=mr(0): if mp(0)+ho>256 then mr(1)=mr(0)+1: goto 1620
1620 s(0)=int(mr(0)/17)+1: sz(0)=mr(0)-s(0)*17+17
1630 if s(0)>=18 then s(0)=s(0)+1
1640 s(1)=int(mr(1)/17)+1: sz(1)=mr(1)-s(1)*17+17
1650 if s(1)>=18 then s(1)=s(1)+1
1660 :
1670 print#15,"u1";5;0;s(0);sz(0)
1680 print#15,"b-p:";5;mp(0): x$=""
1690 if mr(0)<>mr(1) then goto 1780
1700 :
1710 rem egy blokkba elfert
1720 for i=mp(0) to mp(0)+ho-1
1730 : get#5,a$: if a$="" then a$=chr$(0)
1740 : x$=x$+a$
1750 next i: return
1760 :
1770 rem ket blokkba fert bele
1780 for i=mp(0) to 255
1790 : get#5,a$: if a$="" then a$=chr$(0)
1800 : x$=x$+a$
1810 next i
1820 :
1830 print#15,"u1";5;0;s(1);sz(1)
1840 print#15,"b-p:";5;0
1850 for i=0 to mp(0)+ho-256-1
1860 : get#5,a$: if a$="" then a$=chr$(0)
1870 : x$=x$+a$
1880 next i: return
1890 :
1900 rem rekord kiirasa
```

```
1910 rem **************
1920 :
1930 mu=(ix-1)*ho :rem elso byte
1940 mr(0)=int(mu/256)
1950 mp(0)=mu and 255
1960 mr(1)=mr(0): if mp(0)+ho>256 then mr(1)=mr(0)+1
1970 s(0)=int(mr(0)/17)+1: sz(0)=mr(0)-s(0)*17+17
1980 if s(0)>=18 then s(0)=s(0)+1
1990 s(1)=int(mr(1)/17)+1: sz(1)=mr(1)-s(1)*17+17
2000 if s(1)>=18 then s(1)=s(1)+1
2010 if mr(1)<>mr(0) then goto 2110
2020 :fp0
2030 rem egy blokkba elfer
2040 print#15,"u1";5;0;s(0);sz(0)
2050 print#15,"b-p:";5;mp(0)
2060 print#5,x$;
2070 print#15,"u2";5;0;s(0);sz(0)
2080 return
2090 :
2100 rem ket blokkba fer
2110 print#15,"u1";5;0;s(0);sz(0)
2120 print#15,"b-p:";5;mp(0)
2130 print#5,left$(x$,256-mp(0));
2140 print#15,"u2";5;0;s(0);sz(0)
2150 :
2160 print#15,"u1";5;0;s(1);sz(1)
2170 print#15,"b-p:";5;0
2180 print#5,right$(x$,mp(0)+ho-256);
2190 print#15,"u2";5;0;s(1);sz(1)
2200 return
```

A relatív file-ok használata rendkívül kényelmessé tenné egy sor feladat programozását. Sajnos a 1541-es lemezegység szoftverje hibás. Ha a relatív file valamelyik rekordja átlépi a blokkhatárt, akkor a rekord módosításánál karakterek veszhetnek el. Ha tehát komoly adatfeldolgozó programot akarunk írni, akkor a rekordok elhelyezését magunknak kell megoldanunk.A fenti program a blokkok közvetlen írását/olvasását használja erre a célra. Az adatokat az 1-17, illetve 19-35 sávok első 17 szektorában helyezhetjük el. (Csak azért, hogy ne kelljen figyelembe venni, hogy a széső sávokban több szektor van. Miután közvetlenül írunk a blokkokba, összesen 256 adatot helyezhetünk el, nem kell az első kettőt a következő blokk mutatójaként használni.A fenti programban a HO változó tartalmazza a rekord hosszát, ami legfeljebb 256 lehet.Attól függően, hogy hányadik rekordról van szó, a rekord vagy egy,vagy maximum két blokkba kerül. Ennek megfelelően mind az írás, mind az olvasás esetén két alapeset lehetséges.

5.7 Automatikus indítású lemezek

A Commodore 128 számítógépet úgy konstruálták meg, hogy lehetőség van bizonyos programok automatikus elindítására. A reset gomb megnyomása, bekapcsolás vagy paraméter nélküli BOOT parancs kiadása után a C-128 a 8-as lemezegység 1. sávjának 0. szektorát a kazetta pufferba olvassa, és ellenőrzi, hogy a blokk első három byte-ja a 'CBM' jelsorozat-e? (Hexadecimálisan: $43, $42, $4D.) Ha nem, akkor semmi sem történik. Ha igen, akkor a blokk további részének megfelelő információ szerint a lemezről további programrész töltődik a memóriába, s meghatározott gépi kódú programra kerül a vezérlés. A blokk a további információkat kell, hogy tartalmazza:

1.-3. byte CBM azonosító kód.
4. byte a következő blokk kezdőcímének alsó byte-ja
5. byte a következő blokk kezdőcímének felső byte-ja
6. byte a betöltendő blokkok szelete
7. byte a betöltendő blokkok száma

8. byte-tól az első $00 byte-ig a BOOTING után kiírandó szöveg. Ha rögtön a 8. byte $00, akkor '...' jelenik meg.

az első $00-tól a következő $00 byte-ig a betöltendő BASIC program neve.
a második $00 byte-tól a blokk végéig gépi kódú program, amit az interpreter a megadott számú blokk és a BASIC program betöltése után elindít. Ha ennek első byte-ja 0, akkor a processzor egy BRK gépi kódú utasítást hajt végre, s a BOOT végrehajtása befejeződik.

Automatikus indítású a CP/M rendszerlemez, a DOS SHELL utility lemez és a JANE programlemeze is.

A DOS SHELL lemezen található egy "AUTOBOOT MAKER" nevű BASIC program, amelyik lehetővé teszi automatius indítású lemezek készítését.

A program a következőképpen működik. Az 1040-es sorban megnyitjuk a 15-ös csatornát és inicializáljuk a lemezegységet. Ezt követően beolvassuk a ba$-ba az 1. sáv 0. szektorának foglaltságát jelző byte-ot, a bl$-ba pedig az 1090-1100 programrészben a blokk első 24 byte-ját. Ennek ismeretében ellenőrizni tudjuk, hogy nem egy automatikus indítású lemezről van szó. Ha igen az 1110-1230 sorokban kiírja ezt a tényt, s hogy milyen szöveget talált a blokkban. Ha mégis úgy döntünk, hogy folytatjuk, akkor az eredeti programot már nem tudjuk automatikusan elindítani. Ezután a program megkérdezi, hogy mi a programunk neve, s hogy BASIC vagy gépi kódú. Az automatikus indítást lehetővé tevő információt az 1630 soroktól kezdődően írjuk ki a blokkba. A további blokkok betöltését szabályozó részbe csupa 0-t írunk, majd kiírjuk a betöltendő file nevét. Ennek hatására megjelenik a BOOTING szöveg után a file neve (f$). Nem töltünk be semmilyen BASIC programot. A gépi kódú program végrehajtja a BOOT "<PROGNEV vagy a RUN "<PROGNEV BASIC parancsot, s ezzel visszatér a BASIC interpreterbe.

```
1000 print"<CLR> *** auto boot creator ***<LE>"
1010 rem
1020 rem   by  fred bowen & terry ryan
1030 rem
1040 open15,8,15,"ij": nu$=chr$(0)
1050 :  if ds then print"*** disk error: ";ds$: e=-1: goto1720
1060 open8,8,8,"#"
1070 print#15,"u1:8 0 18 0":print#15,"b-p";8;5
1080 :  get#8,ba$
1090 print#15,"u1:8 0 1 0"
1100 :  fori=0to24: get#8,a$: bl$=bl$+chr$(asc(a$)): next
1110 if (asc(ba$)and1)=0 then begin
1120 :  if left$(bl$,3)="cbm" then begin
1130 :     print"*** already a boot disk: ";
1140 :     i=8
1150 :     do
1160 :        a$=mid$(bl$,i,1)
1170 :        printa$;
1180 :        i=i+1
```

```
1190 :         loop until a$=nu$
1200 :     print
1210 :     bend: else begin
1220 :      print"*** boot sector used by another program ***"
1230 :     bend
1240 : input"<LE><RVSON> continue (y/n)
                                    <RVSOFF>   n<BALRA><BALRA><BALRA>";a$
1250 : if a$<>"y" then e=-1: goto1720
1260 : ba = -1
1270 bend
1280 print"<LE><RVSON> enter your program's name "
1290 print"<RVSON> (up to 16 characters)      "
1300 input"<LE>  startup<9-szer BALRA>";f$
1310 :
1320 if len(f$)>16 then begin
1330 :  print"q     that's too big!";
1340 :  sleep2
1350 :  goto1300
1360 :  bend
1370 if f$="" then end
1380 :
1390 print"<LE><RVSON> enter your program's type "
1400 print"<RVSON> ('basic' or 'binary')      "
1410 input"<LE>  basic<7-szer BALRA>";t$
1420 :
1430 if t$<>"basic" and t$ <> "binary" then begin
1440 :  print"<LE>     enter 'basic' or 'binary'";
1450 :  sleep2
1460 :  print"<FEL><FEL>"chr$(27)"'<FEL>"
1470 :  goto1410
1480 :  bend
1490 if t$="basic" then rm$="run" : else rm$="boot"
1500 :
1510 print"<LE><RVSON> remove the  write protect "
1520 print"<RVSON> tab from the disk and put "
1530 print"<RVSON> your disk into drive 0 of "
1540 print"<RVSON> unit 8.  be very careful! "
1550 print"<RVSON> this program will destroy "
1560 print"<RVSON> data in track 1 sector 0. "
1570 print"<RVSON> press  return to continue "
1580 getkeya$: if asc(a$)<>13 then e=-1: goto1720
1590 :
1600 aa= len(f$)+dec("b00")+15
1610 al= aa and 255:   ah= aa / 256
1620 :
1630 print#15,"b-p 8 0"
1640 print#8, "cbm";nu$;nu$;nu$;nu$;f$;nu$;nu$;
1650 print#8, chr$(162);chr$(al);chr$(160);chr$(ah);
1660 print#8, chr$(76);chr$(dec("a5"));chr$(dec("af"));
1670 print#8, rm$;chr$(34);f$;nu$
1680 print#15, "u2:8 0 1 0"
1690 if ba=0 then print#15, "b-a 0 1 0"
1700 a$=ds$
1710 :
1720 close8: close15
1730 if e then print"<LE>*** aborted ***": end
1740 print "<LE>*** done ***"
1750 end
```

6. fejezet

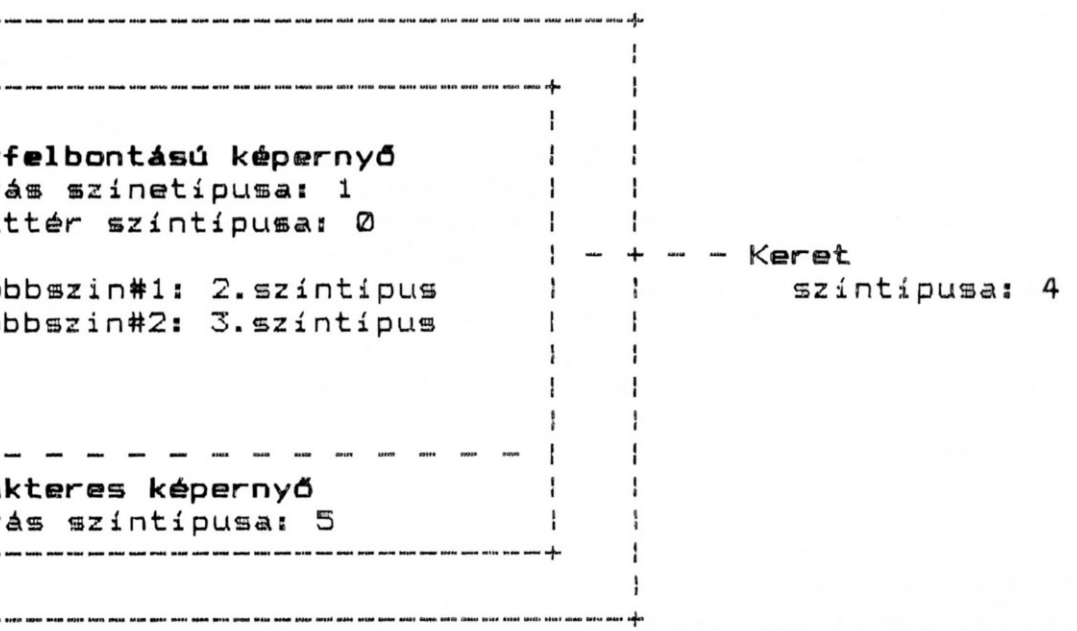
Programozói fogások

6.1 A 40 oszlopos képernyő használata

A C-128-as mikroszámítógép két - logikáját tekintve - eltérő video chipet tartalmaz. Az egyik egy 40, a másik egy 80 oszlopos képernyő generálására alkalmas. Mindkettőhöz más és más csatlakazó tartozik, így egyszerre két képernyőt is kezelhet a rendszer.

A 40 oszlopos képernyőt kezelő VIC IIa chip egyik változata már a Commodore-64-ben is megtalálható. (Az eltérésekre nézve lásd az F.8 függeléket!) A BASIC V2.0 azonban a grafikus lehetőségek egy részét nem támogatja. Ennek megfelelően ezek csak a POKE/PEEK utasításpár segítségével érhetők el. Természetesen ugyanez igaz a Commodore 128 C-128-s módjaban is. A BASIC V7.0 azonban - a bővített háttérszínű üzemmód kivételével - valamennyi lehetőség elérésére speciális grafikus utasításokat tartalmaz!

A grafikus üzemmód kiválasztása
A 40 oszlopos képernyő struktúráját az alábbi ábra mutatja:



A bittérképes, (grafikus vagy nagyfelbontású) üzemmódban a képernyő 320x200 raszter pontját külön-külön kapcsolhatjuk ki és be. Ilyenkor a pontok színezése korlátozott. A képernyő egy karakterhelyén (azaz egy 8x8-as darabján) a pontok kétféle színűek. Ezek közül az egyik szín valamennyi karakterhelyre a háttérszínnel azonos, míg a másik szint a színmemóriában levő kód határozza meg. Nagyfelbontású üzemmódba a GRAPHIC 1 parancs kiadásával térhetünk át.

A többszínű üzemmódban a felbontás a felére (160*200) csökken, de a használható színek száma 4-re nő. Ezek közül egy minden karakterhelyre a háttérszín, míg a másik három tetszőleges karakterhelyre önállóan

megadható, s a 16 szín bármelyike lehet. A GRAPHIC 3, illetve GRAPHIC 4 BASIC utasítások nagyfelbontású, illetve vágott többszínű képernyőt adnak. A megszokott karakteres kijelzésre a GRAPHIC 0 paranccsal térhetünk vissza.

A képernyők közti váltás nem törli a képernyőket. Ehhez a GRAPHIC második paraméterének 1-et kell megadni. Bármelyik képernyő törölhető az SCNCLR utasítással. Ennek egyetlen paramétere van: a törlendő képernyő száma.

Lehetőség van a grafikus üzemmód és a karakteres képernyő egyidejű megjelenítésére. Ezt a megszakító rendszer végzi, ezért a két képernyő találkozásánál időnként egy kicsit villódzik a képernyő.

A grafikus üzemmód kiválasztására a GRAPHIC BASIC utasítás szolgál. Vágott képernyő esetén lehetőség van a képernyő alján levő karaktersorok számának megadására is.

Az RGR függvény a kiválasztott grafikus kijelzési mód kódjával tér vissza.

A képernyő szineinek megválasztása

A BASIC V7.0 hét színtípussal képes dolgozni. Ezekre a 0-6 számokkal kell hivatkozni. (Ezek közül a 6-os színtípus csak a 80 oszlopos képernyőn használható.) Ezen túlmenően a karakteres képernyőn a betűk színét a színbillentyűk segítségével is meg lehet adni. Valamennyi színtípus színe a lehetséges 16 féle szín bármelyike lehet. A konkrét színeket a COLOR utasításban az 1-16 kódokkal lehet megadni. Ezeket a rendszer azonban 0-15-nek tárolja. Az 1 pl. a fekete kódja (lásd az F.6 függeléket!).

Ha a háttér színét feketére akarjuk beállítni, akkor a COLOR 0,1 parancsot kell kiadnunk.

A színtípusok aktuális értékét az RCLR függvény adja vissza.

A színtípusokat a rajzoló utasításokban is használhatjuk. Pl. a következő program rajzol egy szakaszt, majd törli azt:

```
10 GRAPHIC 1,1
20 DRAW 1,0,0 TO 100,100: SLEEP 1
30 DRAW 0,100,100 TO 0,0
```

A törlést azzal értük el, hogy a háttér színével rajzoltunk.

Az egyes színtípusok jelentése a következő:

színtípus	**jelentés**
0	háttér(papír) színe
1	grafikus írás (tinta) színe
2	több szín#1
3	több szín#2
4	keret színe
5	szöveg írás (tinta)színe

A kurzorok

Mind a karakteres, mind a nagyfelbontású képernyőhöz tartozik egy kurzor. a kurzor a karakteres képernyőn könnyen felismerhető, hiszen a helyén egy téglalap villog. A nagyfelbontású képernyőn egy ún. grafikus kurzor-t használhatunk, amely jelentése hasonló a 'normális' kurzorhoz: azt a helyet mutatja, ahol a következő rajzoló utasítás kezdődik. A grafikus kurzort a **LOCATE** utasítás segítségével tetszőlegesen beállíthatjuk. A grafikus kurzor pillanatnyi helyzetét az RDOT függvény adja meg. A nagyfelbontású képernyőt az első GRAPHIC 1, GRAPHIC 3, GRAPHIC 4 utasítás hozza létre. Addig a rajzoló utasítások nem használhatók.

Rajzolás

A BASIC V7.0 egy sor utasítást tartalmaz, amely segítségével a nagyfelbontású képernyőn rajzolni lehet. Ezek a rajzolás módját és a rajzolandó alakzatokat is megadják. A **SCALE** segítségével a képernyő logikai méretét adhatjuk meg. Fizikailag a nagyfelbontású képernyő 320 x 200 pontból áll. A SCALE segítségével a képernyőt maximum 1024 x 1024 pontból állónak tekinthetjük. A rajzoló utsítások a szükséges átalakításokat automatikusan elvégzik, s ennek megfelelően állítják be a képernyő pontjait. Például a SCALE 1,500,300 hatására a rendszer a nagyfelbontású képernyőt 500 x 300 -asnak tekinti.

A rajzoló utasítások szélessége normál körülmények közt 1. Ez azt jelenti, hogy ha pl. egy vízszintes szakaszt rajzolunk, akkor egyetlen vízszintes rasztersor fizikai pontjai kapcsolódnak be. Lehetőség van a rajzolás szélességét 2-re növelni. Ekkor két egymás melletti rasztersor pontjait kapcsolja be a rendszer. Ez a ferde szakaszokat és a köröket is összefüggővé teszi. A vonalvastagság a WIDTH i utasítással változtatható. i értéke 1 vagy 2 lehet.

Az egyes rajzoló utasítások a következők:

utasítás	**jelentés**
BOX	téglalap rajzolása
CHAR	karakterek rajzolása
CIRCLE	ellipszis, kör, szabályos sokszög
DRAW	pontok és szakaszok rjzolása
PAINT	képernyő egy részének festése

A nagyfelbontású képernyő tárolása

A nagyfelbontasú képernyő lemezen történő tárolására, illetve onnan történő visszatöltésére a BSAVE, illetve a BLOAD utasításokat használhatjuk. Ehhez természetesen tudni kell, hogy a képernyő tartalmát meghatározó memóriaészek hol helyezkednek el. A nagyfelbontású képernyő színmemóriája és maga a bittérkép is a 0. memóriaszeleten található. A színmemória a $1C00-$1FFF (7168-8191), a bittérkép a $2000-$3FFF (8192-16383) címeken található. Ennek megfelelően a nagyfelbontású képernyő tartalmát a

```
BSAVE "KEP", ON B0, P7168 TO  P16383
```

BASIC utasítás segítségével írathatjuk ki a "KEP"nevű lemezes file-ba.

Ugyanezt a file-t a BLOAD "KEP" ,ON B0 utasítás kiadásával tölthetjük vissza. Vigyázzunk, ezt megelőzően adjunk ki legalább egy GRAPHIC 1 utasítást, hogy az interpreter a 7168-16383 területet bittérkép számára lefoglalja.

Nagyfelbontású képernyő nyomtatása

A nagyfelbontású képernyő papírra való átmásolása egyáltalán nem egyszerű feladat. Először is grafikus nyomtatóra van szükség, s a kiíratás módja mindig az adott nyomtató tulajdonságaitól függ. Másodszor el kell dönteni, hogy a színstrukturát hogyan tükrözzük vissza a nyomtatáskor.

Az alábbi program a nagyfelbontású képernyőt egy MPS801-gyel funkcionálisan ekvivalens nyomtatóra írja ki. A színmemória tartalma érdektelen: a bittérkép bekapcsolt pontjainak egy-egy pont felel meg a papíron. A többszínű üzemmódban rajzolt képek tehát egyáltalán nem nyomtathatók így.

A program meglepően bonyolult. Ennek oka, hogy az MPS801-nek 7 tűje van, szemben a programozási szempontól ideális 8-cal. Ezért jelenik meg felállítva az ábra. (Az MPS802-nek 8 tűje van, de nincs grafikus üzemmódja...)

```
100 rem ****************************
105 rem * Nagyfelbontású képernyő *
110 rem *       kinyomtatása       *
115 rem ****************************
120 print"<clr>"
130 open 4,4: dim z%(199)
135 s=8192: bank 0: fast
140 for i=39 to 0 step -1:for j=0 to 24
145 for k=0 to 7
150 q=j*8+k
155 z%(q)=z%(q)+(peek(s+320*j+i*8+k))*2↑y
160 a$=a$+chr$((z%(q) and 127) + 128)
165 z%(q)=z%(q)/128
170 next k:next j
175 print#4,chr$(8);a$:a$=""
180 y=y+1:if y=7 then y=0:goto 195
185 print i: next i
190 r=1
195 for l=0 to 199
200 a$=a$+chr$((z%(l) and 127)+ 128)
205 z%(l)=z%(l)/128:next l
210 print#4,chr$(a);a$:a$=""
215 if r=0 then 185
220 close 4: slow
225 end
```

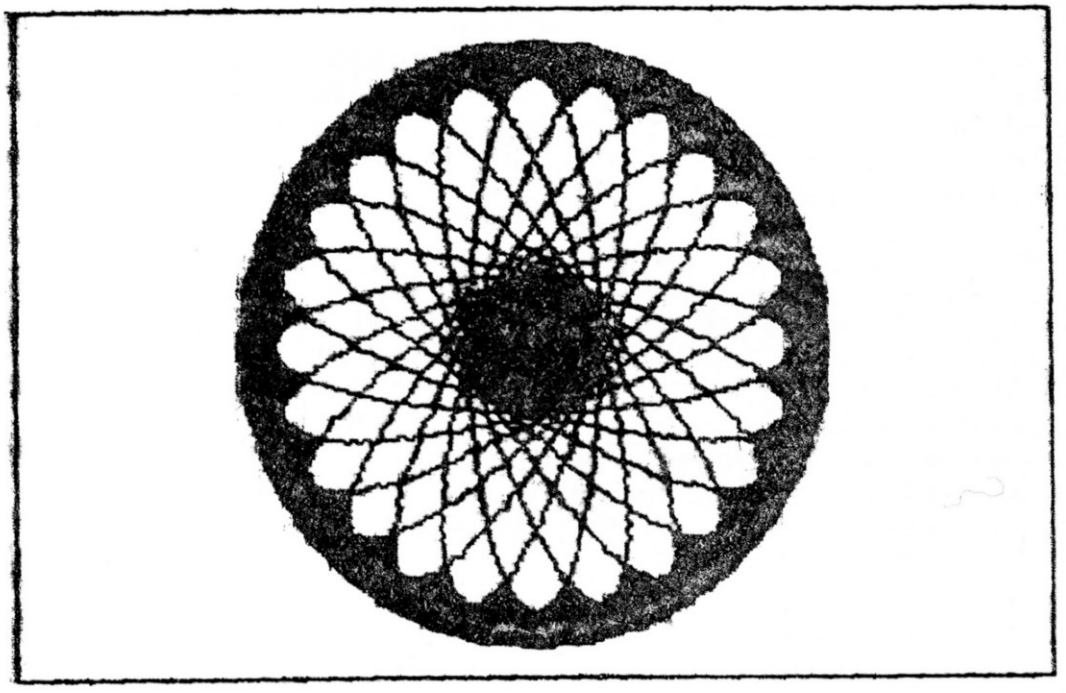
Példaprogramok

Virág

Az alábbi példa mutatja, hogy a legegyszerűbb utasítások felhasználásával is igen szép rajzok készíthetők. Példánk a CIRCLE utasítás azon tulajdonságán alapszik, hogy egyrészt lehetőség van ellipszis rajzolására, másrészt az ellipszis tengelyének hajlásszögét magunk adhatjuk meg.

```
10 graphic 1,1
20 for i=0 to 11
30 circle 1,159,100,20,80,,,15*i
```

```
40 next i
50 paint 1,159,100
60 circle 1,159,100,90,90
70 paint 1,70,100
80 getkey a$
90 graphic 0
```



Szinusz

Az alábbi példaprogram megfelelője szinte valamennyi számítógép gépkönyvében megtalálható. Feladata egyszerű, szinuszgörbét rajzol a képernyőre. Ezt az 50-80 ciklus végzi. A DRAW utasítás segítségével a szinuszgörbe utoljára megrajzolt pontját összekötjük a görbe következő pontjával. Ilyen módon a görbe a meredekebb szakaszain is folytonos lesz.

```
10 color 0,1
20 color 1,2
30 graphic 1,1
40 locate 0,100
50 for x=1 to 319
60 : y=int(100+99*sin(x/20))
70 : draw 1 to x,y
80 next x
90 :
100 sleep 5
110 graphic 0
```

Léggömb

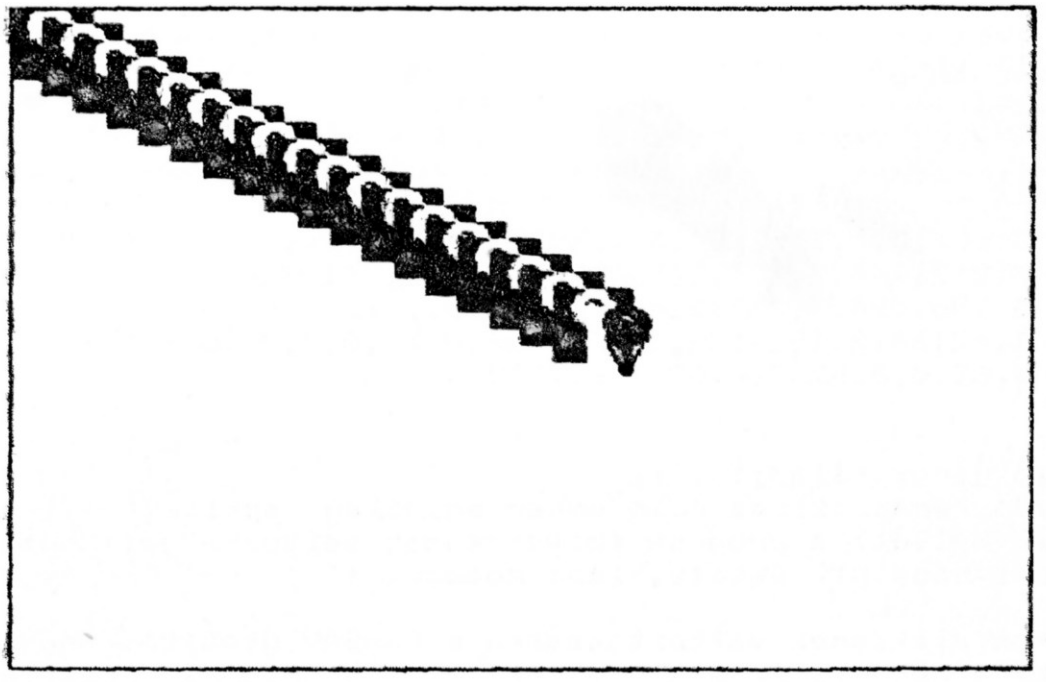
Az alábbi program az SSHAPE és GSHAPE használatát mutatja be. Az SSHAPE segítségével a képernyő egy téglalap alakú darabját egy sztring változóba tudunk átmenteni. Ez csak a bittérkép pontjaira vonatkozik, a színek nem tárolódnak. A téglalap nagyságának csak az szab határt, hogy egy sztring hossza legfeljebb 255 lehet.

A program egy léggömböt mozgat a képernyő átlójában. A léggömb adatait a 190-230 DATA sorokban tároltuk. A 10-es sorban beállítjuk a nagyfelbontású üzemmódot, majd beállítjuk a g változóval a bittérkép kezdőcímét.

A 30-90 ciklus a képernyő bal felső sarkába másolja a léggömböt. A 70 sor mutatja, hogy a DRAW utasítás nélkül mennyire bonyolult egy adott pont bekapcsolása! (Lásd C-64 II.kötet 24-225 oldalak!)

A 100-as sorban az x$ változóba mentjük a léggömb alakját. Ezután a léggömb mozgatására a 110-140 cilusban kerül sor. A 120, 130 utsításokban - az x értékétől függő helyre - visszamásoljuk az x$-ban levő 'képrészt'. A visszamásolás módja a GSAHPE utolsó paraméterétől függ. A 120-as sorban ez 1, ami inverz kiírásnak felel meg. Ennek hatására a léggömb egy csíkot húz maga után.

```
10 graphic 1,1
20 g=8192
30 for i=0 to 62
40 : read a
50 : s=int(i/3):o=i-3*s
60 : ss=int(s/8):so=s-8*ss
70 : ci=g+320*ss+so+8*o
80 : poke ci,a
90 next i
95 :
100 sshape x$,0,0,24,21
110 for x=10 to 320 step 10
120 : gshape x$,x-10,x/2-5,1
130 : gshape x$,x,x/2
140 next x
150 getkey a$: graphic0: end
155 :
190 data 0,127,0,1,255,192,3,255,224,3,227,224
200 data 7,217,240,7,223,240,7,217,240,3,231,224
210 data 3,255,224,3,255,224,2,255,160,1,127,64
220 data 1,62,64,0,156,128,0,156,128,0,73,0,0,73,0
230 data 0,62,0,0,62,0,0,62,0,0,28,0
```



Sprite-ok használata
A sprite-ok 21x24 pontból álló képelemek, amelyek a képernyőn bárhol megjeleníthetők - függetlenül a használt kijelzési módtól. A sprite-okat a VIC IIa chip kezeli, s egyszerre 8-at képes megjeleníteni. A chip működésének ez a része részletesen megtalálható a C-64 II.kötet 7.6 fejezetében (231-243 oldalak!). Itt csak arról lesz szó, hogy a BASIC V7.0 segítségével hogyan lehet a sprite-okat használni.

A sprite-ok használatához a következőket kell elvégezni:

a/ a sprite alakjának megadása ;
b/ a sprite alakjának változtatása;
c/ többszínű üzemmód esetleges megadása;
d/ a sprite egyéb paramétereinek megadása;
e/ a sprite kezdő helyzetének beállítasa;
f/ a sprite bekapcsolása;
g/ a fenti értékek folyamatos változtatása, a megszakítások kezelése.
(Lásd még C-64 II.kötet 237-238 oldalak)

Sprite alakjának megadása
A sprite alakjának definiálására külön **sprite-szerkesztőt** tartalmaz a rendszer. A szerkesztőt a SPRDEF parancs kiadásával lehet aktivizálni. Mind normál, mind többszínű sprite-ok szerkeszthetők. A sprite-ok alakját az interpreter a 0.szelet $0E00-$0FFF címein tárolja. Egy definíció 64 byte-ból áll (a 64. byte-ot a rendszer nem használja). Ennek ismeretében közvetlenül a memóriába írva is megadhatjuk a sprite-ok alakját.

Ha a sprite-ok alakját tárolni akarjuk, akkor azt a BSAVE "SPRITE", ON B0, P3584 TO P4095 BASIC utasítással tehetjük meg. Az elmentett definíciókat a BLOAD "SPRITE", ON B0 utasítassal tölthetjük vissza.

Az alábbi programrész közvetlenül a memóriába írással adja meg egy sprrite alakját:

```
140 for i=0 to 62
150 : read a
160 : poke dec("0e00")+i,a
170 next i: return
180 :
190 data 0,127,0,1,255,192,3,255,224,3,227,224
200 data 7,217,240,7,223,240,7,217,240,3,231,224
210 data 3,255,224,3,255,224,2,255,160,1,127,64
220 data 1,62,64,0,156,128,0,156,128,0,73,0,0,73,0
230 data 0,62,0,0,62,0,0,62,0,0,28,0
```

Sprite alakjának változtatása
A képernyőn keresztülmasírozó ember egyetlen sprite, aminek alakját (és persze helyét) a program folyamatosan változtatja. Ennek hatására úgy látjuk, hogy pl. a keze, lába mozog.

A sprite-ok alakjának változtatására a SPRSAV utasítást használhatjuk. Ennek hatása hasonló az SSAHPE, illetve a GSHAPE utasításokhoz: a sprite alakját leíró byte-okat egy sztringben tárolja. Az alábbi programrész pl. az első sprite alakjat adja a 2. és 3. sprite-nak is:

```
100 SPRSAV 1,UL$
110 SPRSAV UL$,2
120 SPRSAV UL$,3
```

Többszínű üzemmód

Lehetőség van a sprite-okat többszínű üzemmódban megjeleníteni. Ebben az esetben a 01 és 11 bitkombinációk a sprite többszín regiszterek színével jelennek meg. Ezeket a SPRCLR utasításban lehet megadni. Az RSPCOLOR függvény a sprite többszínt adja vissza.

Minden egyes sprite-ra külön megadhatjuk, hogy normál, vagy többszínű üzemmódban jelenik-e meg.

Sprite-ok paramétereinek megadása

A SPRITE utasítás segítségével a sprite-ok összes paraméterét beállíthatjuk. Ezek a paraméterek sorban a következők:

= a sprite bekapcsolása
= a sprite színe
= a sprite-háttér prioritás megadása
= X irányú nagyítás bekapcsolása
= Y irányú nagyítás bekapcsolása
= többszínű üzemmód bekapcsolása

Ezek a paraméterek minden sprite-ra külön beállíthatók. A RSPRITE függvény ezeket a paramétereket adja vissza.

Sprite bekapcsolása

Amíg a sprite-ot nem aktivizáltuk, azaz nem kapcsoltuk be, addig nem látszik és semmilyen hatása sincs, pl. nem ütközik. Ha a sprite-ot bekapcsoljuk, akkor a látható részbe eső, nem takart darabjai megjelennek, s ütközik is.

Sprite helyzetének beállítasa és mozgatása

A MOVSPR parancs sprite-ok elhelyezésére es mozgatására szolgál. Az alábbi programrész három - azonos alakú - sprite-ot jelenít meg. A 10-es sorban - az előbb már bemutatott programrészek segítségével beállítjuk az 1-3. sprite-ok alakját, majd megadjuk a paramétereit és be is kapcsoljuk mind a hármat (20-40 sorok). Végül a 60-80 sorokban a képernyő közepére helyezzük őket. A program futtatása után a sprite-ok a képernyőn maradnak. Listázzuk ki a programot! Látni fogjuk, hogy a két szélső sprite a szöveg felett, míg a harmadik alatta látszik.

```
10 gosub 140: gosub 100
20 sprite 1,1,2,1,1
30 sprite 2,1,4
40 sprite 3,1,2,,1,1,1
50 :
60 movspr 1,100,100
70 movspr 2,60,60
80 movspr 3,140,140
90 end
100 sprsav 1,ul$
110 sprsav ul$,2
120 sprsav ul$,3: return
130 :
```

```
140 for i=0 to 62
150 : read a
160 : poke dec("0e00")+i,a
170 next i: return
180 :
190 data 0,127,0,1,255,192,3,255,224,3,227,224
200 data 7,217,240,7,223,240,7,217,240,3,231,224
210 data 3,255,224,3,255,224,2,255,160,1,127,64
220 data 1,62,64,0,156,128,0,156,128,0,73,0,0,73,0
230 data 0,62,0,0,62,0,0,62,0,0,28,0
```

A sprite-ok elhelyezésére egy speciális koordinátarendszer szolgál, az ún. **sprite tér** (lásd C-64 II.kötet 235. oldal). Ennek nagysága 512x256, s ebben helyezkedik el a látható képernyő 320x200-as darabja. A sprite-tér feladata, hogy a sprite-ok folyamatosan rágörgethetők legyenek a képernyőre. A sprite térben a látható rész egy ablakot alkot, ami a (24,50) koordinátájú ponttal kezdődik, s a (343,249) pontig tart.

Vigyázat, a sprite-ok a nem látható részben is ütközhetnek egymással!

Megszakítások kezelése

A VIC IIa chip automatikusan jelzi, hogy két sprite, vagy egy sprite a háttérrel ütközött, azaz legalább egy, nem áttetsző pontjuk fedésbe került. Lehetőség van ennek figyelésére, illetve ütközés esetén feltételes vezérlésátadásra.

Előbbire a BUMP függvényt használhatjuk, ami az ütközési regiszterek tartalmával tér vissza.

Az utóbbihoz a COLLISION <típus> [,<sorszám>] utasítást használhatjuk. A <típus> értéke 1, ha a sprite-sprite, 2 ha a sprite-háttér ütközéseket akarjuk megfigyelni. Az utasítás kiadását követően ha a megfelelő típusú ütközés bekövetkezik, akkor a vezérlés a <sorszám> sorra kerül. Innen a RETURN utasítással térhetünk vissza. Ha nem adjuk meg a sorszámot, akkor a megfelelő ütközések figyelése elmarad. Az RSPPOS függvény megadja a sprite-ok X,Y koordinátáját és a sebességüket is!

Végül egy - a C-64-es könyvben is szereplő játékprogram C-128-as változatát adjuk meg. Az égen átúszó léggömböt kell lelőni. A rakéta a <szóköz> lenyomásával indul el. (Időnként a rendszer teljesen lemerevedik, miért?!)

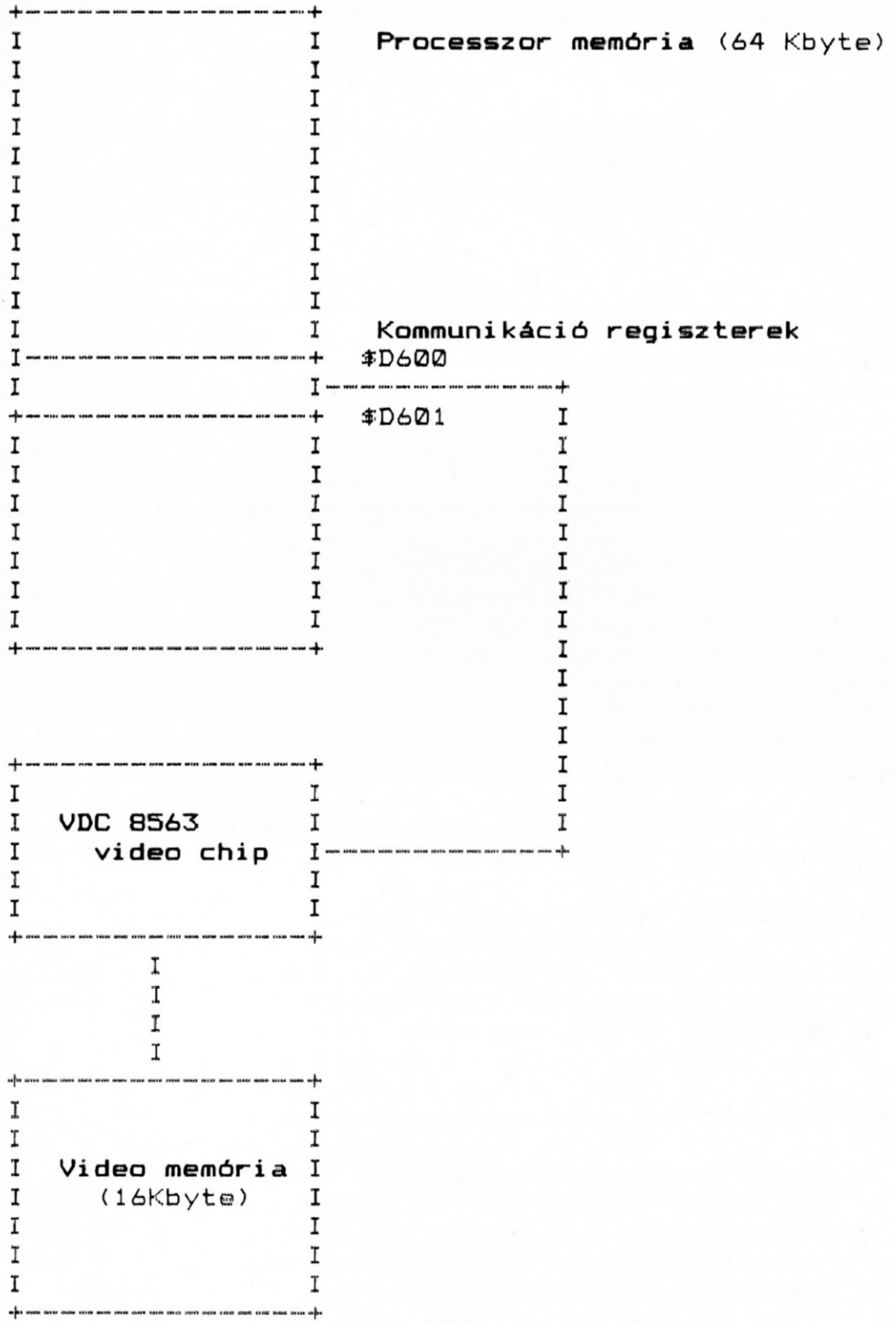
```
10 rem ***********************************
20 rem *                                 *
30 rem *           vadaszat              *
40 rem *                                 *
50 rem *                                 *
60 rem * az egen atuszo leggombot kell *
70 rem * a space billentyuvel leloni.  *
80 rem *                                 *
90 rem ***********************************
100 print"<CLR>";
110 rem sprite alakjanak betoltese
120 for n=0 to 62: read q:poke 3584+n,q:next
130 for n=0 to 62: read q:poke 3648+n,q:next
140 rem hatter es keret szine
150 color 0,1:color 4,1
160 :
170 rem idomeres
180 tj=ti: print"<CLR>";: j=0
190 :
200 do while ti-tj<=3600
210 : x=bump(1): k=0
220 : sprite 2,0,8,0,0,0,0
230 : sprite 1,0,4,0,0,0,0
240 : movspr 2,rnd(1)*130+195,229
250 : movspr 1,0,rnd(0)*100+45
260 : sprite 1,1: sprite 2,1
270 : do: get a$: loop until a$=""
280 :
290 : movspr 1,90#5
300 : do
310 : : if a$<>"" then goto 380
320 : : get a$: if a$<>"" then begin
330 :
340 : : : rem a raketa inditasa
350 : : : movspr 2,0#10
360 : : bend
370 :
380 rem utkozes ellenorese
390 : : if bump(1)=0 then goto 440
400 : : k=1-k
410 : : sprite 1,,,,k,k
420 : : sprite 2,,,,k,k
430 : : j=j+1:print"S  ";j
440 : : if rsppos(1,0)>380 then movspr 1,0#0
450 : : if (rsppos(2,1)<20) or (rsppos(2,1)>225) then movspr 2,0#0
460 :loop until (rsppos(1,0)>380) and ((rsppos(2,1)<20)
or rsppos(2,1)>225)
470 : movspr 1,0#0
480 : movspr 2,0#0
490 loop
500 :
510 rem *************************
520 rem * a vegeredmeny kiirasa *
530 rem *************************
540 print"<CLR><LE>      talalatok szama=";j
550 input"<LE><LE>   akarsz meg egyet jatszani
                                    i<JOBBRA><JOBBRA><JOBBRA>";a$
560 if a$="i" then goto 180
570 print"<CLR>":sprite 1,0:sprite 2,0
```

```
580 end
590 :
600 rem ********************
610 rem * a leggomb adatai *
620 rem ********************
630 data 0,127,0,1,255,192,3,255,224,3,227,224
640 data 7,217,240,7,223,240,7,217,240,3,231,224
650 data 3,255,224,3,255,224,2,255,160,1,127,64
660 data 1,62,64,0,156,128,0,156,128,0,73,0,0,73,0
670 rem ********************
680 rem * a raketa adatai *
690 rem ********************
700 data 0,62,0,0,62,0,0,62,0,0,28,0
710 data 0,0,0,0,0,0,0,0,0,0,0,0
720 data 0,0,0,0,0,0,0,0,0,0,0,0
730 data 0,0,0,0,0,0,0,0,0,0,0,0
740 data 0,24,0,0,24,0,0,24,0,0,24,0
750 data 0,24,0,0,24,0,0,60,0,0,60,0
760 data 0,126,0
```

6.2 80 oszlopos video chip

A 80 oszlopos képernyő kezelését a VDC 8563 jelű videochip biztosítja. Ehhez egy speciális RGBI monitorra is szükség van. A monitor kimenetének a specifikációját a F.10 függelékben adjuk meg. A VDC regisztereinek részletes ismertetése a C-128 könyv II. kötetében, illetve összefoglalóan a Commodore információs kártyában található meg.

A VDC chip használata lényegesen eltér a többi periféria vezérlő chiptől. Azok regiszterei és kép megjelenítéséhez szükséges memóriarészek a processzor memóriaterületén helyezkedtek el. A VDC chip-nek saját (maximum 64Kbyte-os) videó memóriája, önálló regiszterei vannak. A processzor operatív memóriájában csak a kommunikációs regiszterek helyezkednek el:


Processzor memória (64 Kbyte)
Kommunikáció regiszterek
$D600
$D601
VDC 8563
video chip
Video memória
(16Kbyte)

A VDC chipnek két kommunikációs regisztere és 37 önálló regisztere van. A chip video memóriájanak címbusza mindössze 8 bites, ezért külön adja ki a sor és külön az oszlop címét. Végeredményben azonban egy 64 Kbyte-os video memóriát használhat a chip. A Commodore 128-ban csak 16 Kbyte-nyi video memóriát építettek be. A chip használatához ez bőven elegendő.

A kommunikációs regiszterek

A kommunikációs regiszterek az I/O területen a $D600 és a $D601 címeken helyezkednek el. Ha használni akarjuk őket, akkor a memóriaszeletet ennek megfelelően kell a BANK utasításban megválasztani.

cím	leírás	bitek
$D600	regiszterszám írás	--- --- R5 R4 R3 R2 R1 R0
	olvasás	status LP VBLANK
$D601	adat írás-olvasás	D7 D6 D5 D4 D3 D2 D1 D0

A $D600 regiszterbe történő írással kiválaszthatjuk, hogy a VDC chip melyik regiszterébe akarunk írni, vagy onnan olvasni. A regiszter olvasásával tájékozódhatunk arról, hogy a VDC chip végrehajtotta-e már az utolsó utasítást. Erre a **status** bit szolgál. Ha a regiszterbe való írás után ezt kiolvassuk és még nem alacsony (=0), akkor a VDC még nem kész az adat olvasására-írására. Ha már alacsony, akkor a $D601-ben a megfelelő regiszter tartalma van, s ha ekkor írunk bele, akkor annak értékét a VDC átmásoja a megfelelő regiszterbe!

A következő gépi kódú rutin az X indexregiszternek megfelelő regiszterbe írja az akkumulátor (A) értékét:

```
      STX $D600  ; a regiszter értékének beírása
VARJ  BIT $D600  ; a Z bit értéke a STATUS bit lesz
      BMI VARJ   ; ha magas, akkor várakozás
      STA $D601  ; az új regisztertartalom beírása
      RTS        ; visszatérés az alprogramból
```

A regiszter tartalmát hasonló rutinnal lehet kiolvasni:

```
      STX $D600  ; a regiszter értékének beírása
VARJ  BIT $D600  ; a Z bit értéke a STATUS bit lesz
      BMI VARJ   ; ha magas, akkor várakozás
      LDA $D601  ; a regisztertartalom kiolvasása
      RTS        ; visszatérés az alprogramból
```

Az LP bit a fényceruza pozitív átmenetét mutatja, míg a VBLANK bit azt, hogy a videochip RGBI kimenete éppen passzív. Ez utóbbit bizonyos szinkronizációkhoz lehet használni.

A regiszterek funkcióiról részletesen a II. kötetben szólunk.

A video memória írása/olvasása

A video memóriába közvetlenül sem írni, sem onnan olvasni nem lehet.

Ehhez a VDC chip megfelelő regisztereit kell beállítani. Ezek a következők:

Regiszter	Tartalom
18 ($12)	memóriacím felső byte
19 ($13)	memóriacím alsó byte
31 ($1F)	memória tartalma

A 18-19 regiszterek tartalma a video memória egy címét tartalmazza. Ha a 31-ik regisztert kiolvassuk, a videomemória adott címét kapjuk, ha oda írunk, a regiszter tartalma automatikusan átmásolódik a video memóriába.

A következő gépi kódú programrészt a SYS VBE,KO,HI,LO BASIC utasítással kell meghívni. Ennek hatására a KO értéke (0-255) a video memória LO+256*HI címére másolódik:

```
VBE   PHA          ; A verembe mentése
      TXA          ; X átírása A-ba, majd
      PHA          ;  a verembe mentése
      LDX #$13     ; alsó byte regisztere
      TAY          ;  tartalma A-ban
      JSR RBE      ;  beírás a regiszterbe
      DEX          ; X-ben a felső byte regisztere
      PLA          ;  visszatöltése a veremből
      JSR RBE      ;  beírás a regiszterbe
      LDX #$1F     ; az adat regisztere
      PLA          ;  adatbyte visszatöltése
      JMP RBE      ;  beírása
RBE   STX $D600    ; a regiszter értékének beírása
VARJ  BIT $D600    ; a Z bit értéke a STATUS bit lesz
      BMI VARJ     ; ha magas, akkor várakozás
      STA $D601    ; az új regisztertartalom beírása
      RTS          ; visszatérés az alprogramból
```

A video memória kiolvasása hasonlóan történik. A különbség a rutin hívásában van: SYS RKI,HI,LO: RREG KO. Ennek hatására a videomemória LO+256*HI címének tartalma a KO változóba kerül.

```
VKI   PHA          ; felső byte verembe töltése
      TXA          ; X átírása A-ba, majd
      PHA          ;  a verembe mentése
      LDX #$13     ; alsó byte regisztere
      PLA          ;  tartalma A-ban
      JSR RBE      ;  beírás a regiszterbe
      DEX          ; X-ben a felső byte regisztere
      PLA          ;  visszatöltése a veremből
      JSR RBE      ;  beírás a regiszterbe
      LDX #$1F     ; az adat regisztere
                   ;  adatbyte visszatöltése
      JMP RKI      ;  az adat kiolvasása
RBE   STX $D600    ; a regiszter értékének beírása
VARJ1 BIT $D600    ; a Z bit értéke a STATUS bit lesz
      BMI VARJ1    ; ha magas, akkor várakozás
      STA $D601    ; az új regisztertartalom beírása
      RTS          ; visszatérés az alprogramból
```

```
RKI   STX $D600  ; a regiszter értékének beírása
VARJ2 BIT $D600  ; a Z bit értéke a STATUS bit lesz
      BMI VARJ2  ; ha magas, akkor várakozás
      LDA $D601  ; a regisztertartalom kiolvasása
      RTS        ; visszatérés az alprogramból
```

A video memória használata

Az interpreter a következőképpen használja a video memóriát:

```
$0000-$07CF   képernyő memória
$0800-$13CF   attributum memória
$2000-$3FFF   karaktergenerátor
```

Az attributum memória egyes bitjei a következőt jelentik:

Bit	Jelentés
7	alternativ karakterkészlet jelző
6	inverz jelző
5	aláhúzás jelző
4	villogás jelző
3	red (piros)
2	green (zöld)
1	blue (kék)
0	intenzitás

A 3.-0. bitek mint színvezérlő jelek közvetlenül megjelennek az RGBI monitoron, míg a 7.-4. bitek a kijelzés módját befolyásolják.

A 6.-4. bitek jelentése világos. Kiemeljjük 6. bitet, a VDC chip z inverz alakú kijelzést hardver úton valósítja meg. A VIC IIa chip esetén az inverz karakterek alakját külön kellett tárolni. A 7. bitről is külön szólunk. A C-128 képernyő szerkesztőjénél említettük, hogy a 80 oszlopos képernyő szerkesztő mind a két karakterkészletet képes megjeleníteni. Ezt a 7. bit segítségével teszi meg.

Karakteres kijelzés

A VDC chip esetén megadható, hogy egyetlen kaarakterhely hány rasztersorból épül fel, s hogy két karaktersor közt hány raszterpont a távolság. Ilyen módon pl. 8x12 raszterpontból álló karaktereket használhatunk.

Egy karakter maximális mérete 8x16 raszterpont lehet. Ennek megfelelően a karaktergenerátor a memóriában mindig ekkora helyet foglal. Ha pl. 8x8-as karaktereket használunk, akkor a karaktergeneráláskor a felesleges 8 byte nem használódik semmire. Az alábbi program 8x16-os karakterek használatát mutatja be. A karaktereket függőleges irányban megnyújtjuk, s minden byte-ot megdupláztunk.

```
100 rem 16*8-as matrix
110 i=0: restore 380
120 read a: if a=-1 then sys dec("b00"): goto 150
130 poke dec("b00")+i,a: i=i+1: goto 120
140 :
150 a=dec("d600"): b=a+1: restore 290
160 poke 228,16
170 :
180 read x
190 : if x=-1 then begin
200 : : print"<CLR>";
210 : : window 0,1,79,15,1
220 : : end
230 : bend
240 : read y
250 : poke a,x: poke b,y
260 goto 180
270 :
280 rem vdc chip regiszter/adat
290 data 9,15
300 data 6,17
310 data 23,15
320 data 4,19
330 data 7,19
340 data 11,15
350 data -1
360 :
370 rem gepi kodu rutin
380 data 162,3,189,65,11,149,250,202,16,248,162,1,142,0,255,160,0
390 data 177,250,72,162,0,142,0,255,166,252,164,253,32,70,11,230,252
400 data 208,2,230,253,104,166,252,164,253,32,70,11,230,252,208,2,230
410 data 253,230,250,208,2,230,251,164,251,192,224,144,202,96,
        0,208,0,32
420 data  0,72,138,72,152,72,169,2,141,40,10,162,18,104,32,97,11,232
430 data 104,32,97,11,162,31,104,76,97,11,142,0,214,44,0,214,16,
        251,141,1
440 data 214,96,-1
```

6.3 Hanggenerálás

A Commodore 128 számítógép ugyanazt a hanggenerátor chipet tartalmazza, mind a Commodore 64, így zenei lehetőségei, hanghatásai teljes mértékben megegyeznek vele. A különbség - hasonlóan a grafikához - a lényegesen könnyebb programozhatóság. A SID működésének leírását megtalálhatjuk a C-64 II.kötet 8.fejezetében, ami az elméleti alapokat ismerteti. Részletesebben lásd. Csikós Zsolt : Commodore 64 Zenekedvelőknek c. könyvében. (LSI ATSz.1987)
Röviden megismételve az ott elmondottakat: a SID-nek három független oszcillátora van, mindegyik egy ADSR típusú burkológörbe generátorral szólaltatható meg. Hangonként négy hullámforma közül választhatunk. Lehetőség van az oszcillátorok kimenő jelének szűrésére. Mindezeket kényelmesen használhatjuk a megfelelő BASIC parancsok használatával.

A hangerő beállítasa
Ahhoz, hogy a SID által generált hangokat egyáltalán halljuk, a hangerőt 0-nál nagyobb értékre kell állítani. Ez a VOL H parancs kiadásával történhet. (max. 15-ig)

A ADSR paraméterek beállítása
Az ADSR paraméterek azt állítják be, hogy hogyan szólal meg egy hang, majd hogyn hal el. Ennek megadására az ENVELOPE utasítást használhatjuk. Ugyancsak ebben az utasításban adhatjuk meg, hogy milyen hullámformát használunk. Ez a megszólaló hang felharmónikusait adja meg.

Előzetesen tíz hangot (burkológörbét és ADSR szintetizátort) definiálhatunk az ENVELOPE segítségével. Ezeknek a hangoknak kezdeti értéke is van, s ezeket rögtön használhatjuk is.

Szűrés
A SID háromfajta szűrőt tartalmaz. Ezek paramétereit a FILTER utasítással kell beállítani. Erre a szűrők még nem kapcsolódnak be.

A hanggenerátor megszólaltatása
A kiválasztott paramétereknek megfelelő hang(ok) megszólaltatására a BASIC V7.0 kétféle lehetőséget kínál. Az első esetben a SOUND segítségével egyetlen hang szólal meg. A második esetben a PLAY utasítással a paramétereként megadott sztring szerint egy egész dallamot játszik az interpreter. A dallam lejátszásáig a program nem fut tovább. A dallam lejátszásának idejét a TEMPO utasítással lehet beállítani. Ez nem befolyásolja a megszólaló hangokat, csupán azt, hogy az egyes hangok megszólalása közt mennyi idő telik el. TEMPO a dallam lejátszásának ütemét adja meg.

6.4 Teknősbéka grafika

A teknősbéka grafika lényege, hogy nem abszolút, hanem relativ utasításokkal dolgozik. Ezt a technikát legkövetkezetessebben a **LOGO** programozási nyelv valósította meg. Erről a C-128 II. kötetében szólunk részletesen.

A teknősbéka grafika alapötlete a következő. Ha a BASIC V7.0 utasításaival közvetlenül akarunk egy szakaszt megrrajzolni, akkor mind a **két végpontjának koordinátáit ismerni kell.** Ugyanezt a szakaszt úgy is megrajzolhatjuk, hogy egyik végpontjából adott irányú és hosszúságú szakaszt húzunk.

Egy kicsit részletesebben a következőről van szó. Képzeljünk el egy teknősbékát, amelyik a képernyőn mászik, a szájában ceruzát és radirt tartva. Ha a fejét felemelve tartja mászás közben, akkor a képernyőn semmilyen váaltozás nincs, csak a teknősbéka helyzete változik. Ha azonban lehajtja a fejét és a szájában ceruz van, akkor mászás közben egy vonalat húz a képernyőn. Ha a szájában a radir van, akkor útja során törli a képernyőt.

A teknősbéka grafika utasításai a teknősbékát irányító parancsok. Valamennyi utsítás relativ, tehát a teknősbéka utolsó helyéhez képest kell majd a mozgás iráanyát megadni.

A teknősbéka grafika utasításait BASIC alprogramokkal valósítjuk meg. Ehhez kell hozzáírni a teknősbékát mozgató parancsokat, amelyek egy sor szubrutinhívást tartalmaznak.

Először felsoroljuk a teknösbéka rutinokat:

Funkció	LOGO parancs	Belépési pont	Paraméter
képernyő törlése	CLEARSCREEN	1050	
ceruza letevése	PENDAWN	1140	
ceruza felemelése	PENUP	1220	
középre mozgás	HOME	1300	
jobbra fordulás	RIGHT	1390	SZOG
balra fordulás	LEFT	1470	SZOG
előre mozgás	FORWARD	1560	HOSSZ
hátra mozgás	BACKWARD	1650	HOSSZ
inicializálás	DRAW	1740	
ceruza szine	PENCOLOR	1840	PC

A programot magát 2000-től kezdődően lehet megírni. Az alábbi programot egyszer megírjuk, s akárhányszor teknősbéka grafikát akarjuk használni, betöltjük a memóriába.

A teknösbéka helyzetét az alábbi változók segítségével írjuk le:

ALFA a teknősbéka fejének a függlegessel bezárt szöge;
XT a teknősbéka X koordinátája;
YT a teknősbéka Y koordinátája;
PD a fej letevését jelzi
PD=1 lehajtott fej
PD=0 felemelt fej
PC a ceruza színe. PC=0: törlés
CT a használt szintípus. Ha PC igazi szín,
akkor CT=1, az írás színe,
különben CT=0, a háttér színe

A 10-es sor a főprogramra adja a vezérlést.

a/ A képernyő törlése csak a képernyőt törli, nem változtatja meg a teknösbéka helyzetét. (1050-1120)

b/ A ceruza letevését PD változó beállításával jelezzük. PD=1 lesz. (1140-1200)

c/ Hasonlóan jelezzük, hogy a ceruzát felemelte. (1220-1280)

d/ A képernyő közepére mozgás kettőt jelent: egyrészt a teknősbéka a képernyő közepére mászik, másrészről a feje alapállapotba kerül. (1300-1360)

e/ A jobbra fordulás az ALFA csökkentését jelenti az adott paraméterrel. (1390-1460)

f/ A balra fordulás az ALFA SZOG-gel történő megnövelését jelenti. (1470-1540)

g/ Az előre mozgáshoz először kiszámítjuk a teknős új koordinátáját 1610-1611 sorok. Ha a ceruza le van rakva, akkor a DRAW, ha nincs, akkor a LOCATE utasítás segítségével mozgatjuk a teknősbékát (s a grafikus kurzort). A rajzolás a CT által meghatározott színnel történik. Ez vagy az írás színe, vagy törlés esetén a háttér színe. (1615-1636)

h/ A hátramozgás az előremozgáshoz hasonlóan történik. (1650-1728)

i/ A rendszer inicializálása a képernyő törlését, a teknös középre mozgatását és a fejének lerakását jelenti. (1800-1820)

j/ A CT beállítasa a PC értékétől függ. Ha a PC az 1-16 tartományba esik, akkor az lesz az írás szine és azzal rajzol arendszer. Ha PC=0, akkor a háttér szinével rajzolunk, ami törlésnek felel meg. (1840-1950)

```
10 GOTO 1960
1000 REM **********************
1010 REM * TEKNOSBEKA GRAFIKA *
1020 REM **********************
1030 :
1040 :
1050 REM ##########################
1060 REM # CLEARSCREEN            #
1070 REM #                        #
1080 REM # A KEPERNYO TORLESE     #
1090 REM ##########################
1100 :
1110 GRAPHIC 1,1: COLOR 0,1: COLOR 4,1
1120 RETURN
1130 :
1140 REM ####################
1150 REM # PENDAWN          #
1160 REM #                  #
1170 REM # CERUZA LETEVESE  #
1180 REM ####################
1190 :
1200 PD=1: RETURN
1210 :
1220 REM ########################
1230 REM # PENUP                #
1240 REM #                      #
1250 REM # CERUZA FELEMELESE    #
1260 REM ########################
1270 :
1280 PD=0: RETURN
1290 :
1300 REM ##############################
1310 REM # HOME                       #
1320 REM #                            #
1330 REM # KEPERNYO KOZEPERE MOZGAS   #
1340 REM ##############################
1350 :
1360 DRAW 1 TO 159,100:XT=159:YT=100
1370 ALFA=-90:RETURN
1380 :
1390 REM ############
1400 REM # RIGHT    #
1410 REM #          #
1420 REM # JOBBRA   #
1430 REM ############
1440 :
1450 ALFA=ALFA-SZOG
1460 RETURN
1470 REM ##########
1480 REM # LEFT   #
1490 REM #        #
1500 REM # BALRA  #
1510 REM ##########
1520 :
1530 ALFA=ALFA+SZOG
1540 RETURN
1550 :
```

```
1560 REM ############
1570 REM # FORWARD #
1580 REM #         #
1590 REM # ELORE   #
1600 REM ############
1610 DEL=<pi>/180*ALFA
1611 XT=XT+HOSSZ*COS(DEL): YT=YT+HOSSZ*SIN(DEL)
1615 IF PD=0 THEN GOTO 1635
1620 DRAW CT TO XT,YT
1630 RETURN
1635 LOCATE XT,YT
1636 RETURN
1640 :
1646 :
1650 REM #############
1660 REM # BACKWARD #
1670 REM #          #
1680 REM # HATRA    #
1690 REM #############
1700 DEL= /180*ALFA
1705 XT=XT-HOSSZ*COS(DEL): YT=YT-HOSSZ*SIN(DEL)
1708 IF PD=0 THEN GOTO 1725
1710 DRAW CT TO XT,YT
1720 RETURN
1725 LOCATE XT,YT
1728 RETURN
1730 :
1740 REM #############
1750 REM # DRAW      #
1760 REM #           #
1770 REM # RAJZOLAS  #
1780 REM #############
1790 :
1800 GOSUB 1110: COLOR 1,2: LOCATE 159,100
1810 ALFA=-90: PD=1: XT=159: YT=100: PC=2: CT=1
1820 RETURN
1830 :
1840 REM ##################
1850 REM # PENCOLOR       #
1860 REM #                #
1870 REM # CERUZA SZINE   #
1880 REM ##################
1890 :
1900 IF PC<1 THEN CT=0: ELSE: CT=1: COLOR 1,PC
1910 RETURN
1920 :
1960 REM ****************
1970 REM *  FOPROGRAM   *
1980 REM ****************
1990 :
```

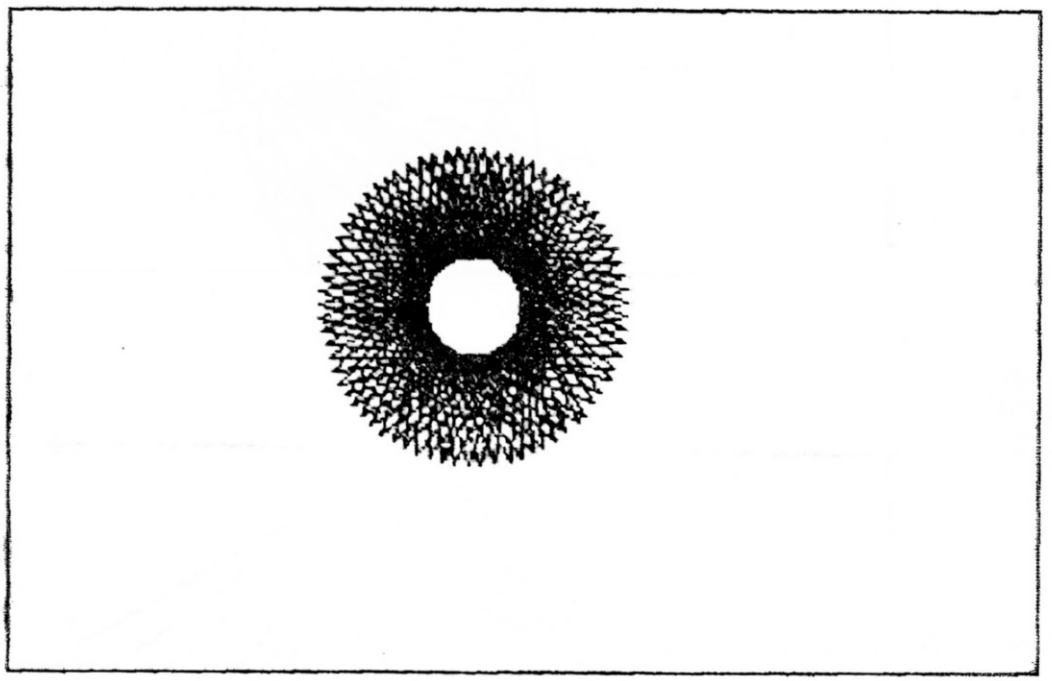
CSILLAG

Első alkalmazásként egy szabályos sokszöget rajzoló teknösbéka programot írunk. Ennek föprogram része az alábbi:

```
1960 REM ************************
1970 REM *  FOPROGRAM  CSILLAG *
1980 REM ************************
1990 :
```

```
2000 INPUT "<CLR>n,hossz,szog=";N,HOSSZ,SZOG
2010 GOSUB 1740     :REM Inicializálás
2020 FOR I=1 TO N
2030 : GOSUB 1560   :REM FORWARD (előre) HOSSZ
2040 : GOSUB 1390   :REM RIGHT (jobbra) SZOG
2050 NEXT I
2060 GETKEY A$
2070 GRAPHIC 0
2080 END
```

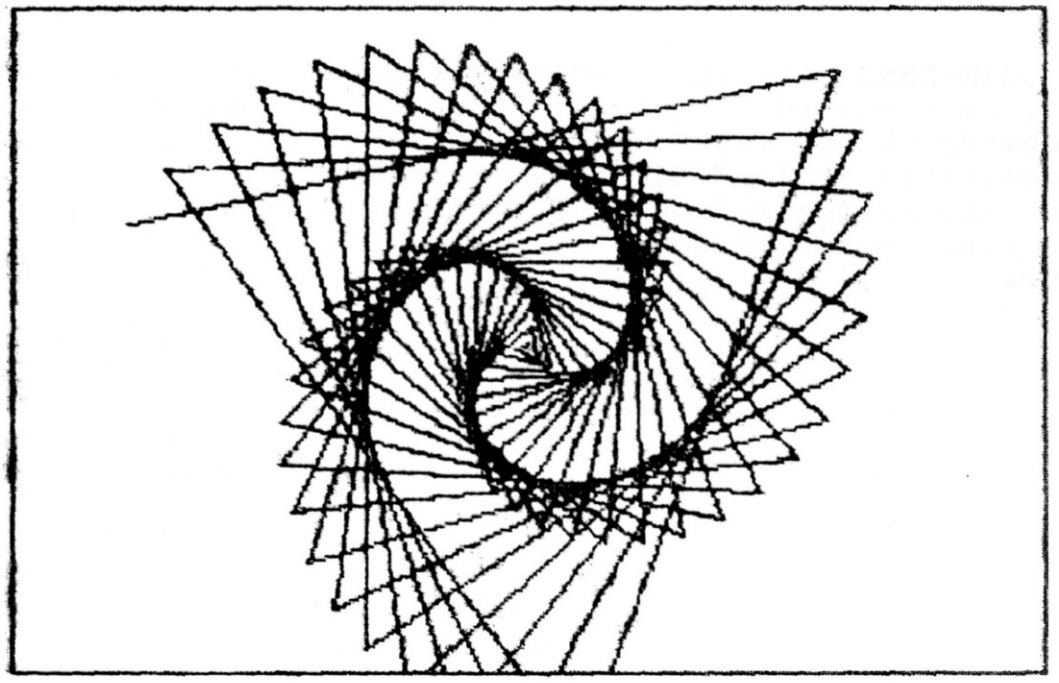
A teknős a 2010-2050 ciklusban N-szer ismétli azt, hogy előremegy valamennyit, majd egy adott szöggel elfordul. Emlékezni kell arra, hogy egy sokszög külső szögeinek összege mindig 360 fok. Egy szabályos ötszög rajzolásához tehát 360/5 = 72 fokot kell fordulni. Ha a 2000. sorban levő inputra tehát 5,80,72 választ adjuk, akkor a program egy szabályo 80 oldalhosszú ötszöget rajzol. A 73,143,90 válasz futásának az eredménye:

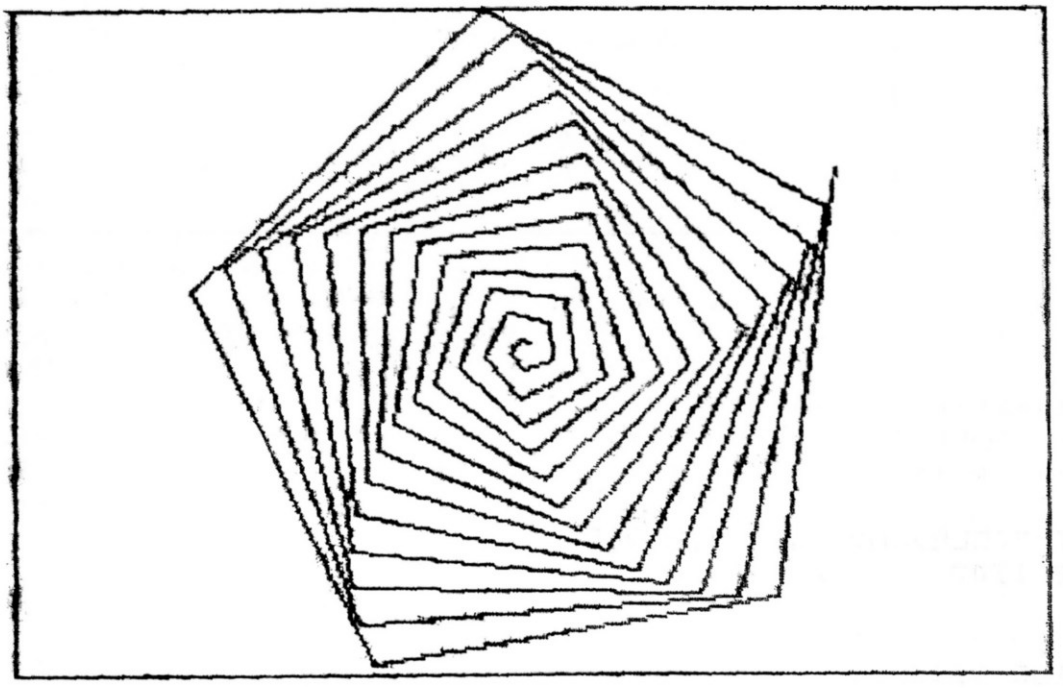


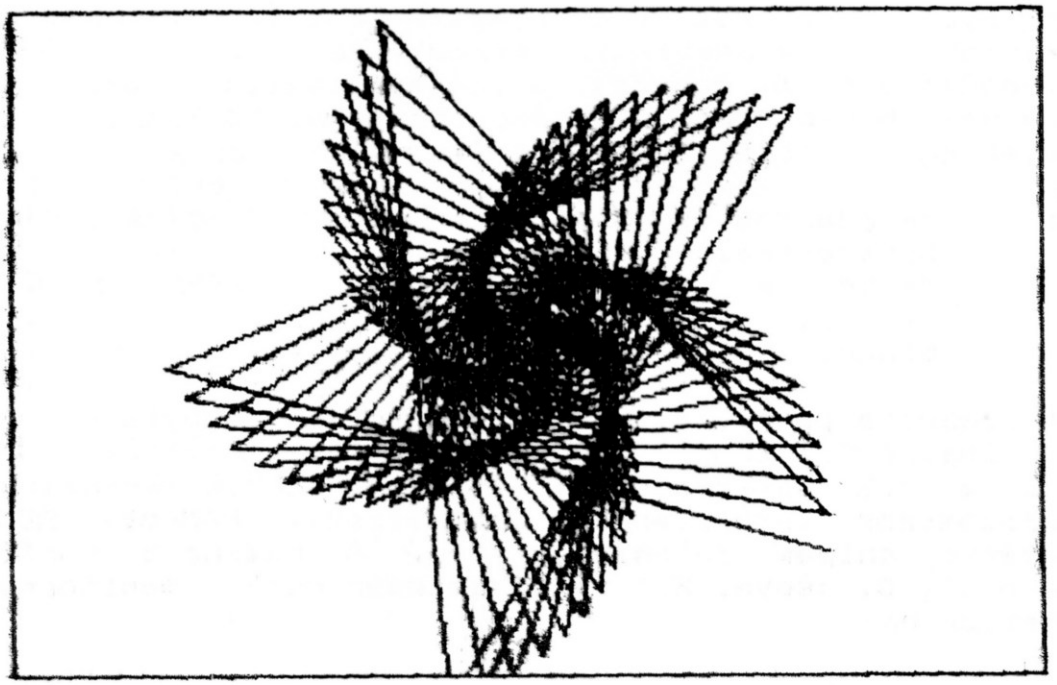
SPIRÁL

```
1960 REM ************************
1970 REM *  FOPROGRAM  SPIRAL *
1980 REM ************************
1990 :
2000 INPUT "<CLR>nov,szog=";NOV,SZOG
2010 GOSUB 1740
2020 HO=1
2030 DO
2040 : GOSUB 1390
2050 : HO=HO+NOV: GOSUB 1560
2060 LOOP UNTIL HO>220
2070 GETKEY A$
2080 GRAPHIC 0
2090 END
```

A spirálrajzoló progam a CSILLAG módosításán alapszik. A csillag program abban különbözik a csillagtól, hogy minden fordulás után a teknős lépéshosszát egy állandoval megnöveljük. A ciklus (2030-2060 sorok) addig fut, míg a HO>220 feltétel először igaz nem lesz. Igen szép ábrákat kaphatunk pl. a 3,123 a 2,73 és a 2,143 válaszokkal!







7. fejezet

Gépi kódú programozás

7.1 A gépi kódú monitor használata

A C-128 számítógép egy TEDMON nevű monitorral rendelkezik. A BASIC-ből a monitort a

MONITOR

parancs kiadásával érhetjük el. Ezt követően - egészen a monitorból való kilépésig - nem használhatjuk a BASIC utasításokat.

A gépi kódú monitor a processzor regisztereinek, illetve a memóriacímek közvetlen írását, olvasását, a memória adott részének elmentését, visszatöltését; továbbá egyszerű assembly nyelvű programozást tesz lehetővé. Ebben a fejezetben a gépi kódú monitor használatát ismertetjük. Nem célunk a processzor utasításkészletének ismertetése, hiszen azt már mások sok helyen megtették. A 8510-es processzor szoftver kompatibilis a 6510-es processzorral. A részletek iránt érdeklődők ezért a szükséges ismereteket megszerezhetik:
dr. Úry László: Commodore 64 BASIC felhasználói kézikönyv,
vagy Erdős Iván: Commodore 64 assembly programozás című, az LSI ATSZ gondozásában megjelent könyveiből.

A TEDMON parancsai egyetlen karakterből állnak, amit - szóközzel elválasztva - követnek a paraméterek. A parancsok kiadására a C-128 teljes képernyős szerkesztője használható. A paraméterek sztringkonstansok vagy hexadecimális számok. Ez utóbbiak a paraméter jellegétől függően vagy 2, 4 vagy 5 hexadecimális számból állnak. Lehetőség van nem hexadecimális számok használatára is. Ezeket a következő jelek egyikével kell bevezetni:

(nincs jel)	hexadecimális (alapértelmezés)
$	hexadecimális
+	decimális
&	oktális
%	bináris

A processzor címbusza 16 bites, ennek megfelelően 64Kbyte memóriát tud egyszerre kezelni. A Commodore 128 memória kezelését az MMU chip végzi. (Lásd a 7.2 paragrafust!) Ez gondoskodik arról, hogy a 64Kbyte-os processzor területen eltérő fizikai RAM-ok, ROM-ok és periféria vezérlő chipek jelenjenek meg. A használt konfiguráció indexe a cím első, 5. jegye. Ezt természetesen csak a monitornak szóló címekhez írhatjuk be!

A

Helyben fordítás. Az 'A' a '.'-tal helyettesíthető

Szintaxis: **A** <cím> <mnemonik> {<operandus>}

A parancs kiadása után a monitor a parancsban szereplő assembly nyelvű utasítást lefordítja, s a neki megfelelő gépi kódot a <cím>-től kezdődően elhelyezi. A következő sorban megjelenik egy

A <újcím>

alakú üzenet, ahol <újcím> a fordítás és az elhelyezés utáni első szabad memória címe. Az assembly utasítások így folyamatosan írhatók.

A <mnemonik> az M6510-es processzor szokásos mnemonikjainak bármelyike lehet. Az <operandus> azonban nem tartalmazhat semmilyen kifejezést vagy címkét. Egyetlen könnyebbség, hogy relatív ugrások esetén is az **abszolút címet kell beírni.** A monitor azután kiszámítja a különbséget.

Példák: A 1400 LDX #$03
A 1402

C

Memóriaterületek összehasonlítása.

Szintaxis: **C** <cím1> <cím2> <cím3>

A parancs a <cím1>-től <cím2>-ig tartó memóriarész tartalmát hasonlítja össze a <cím3>-tól kezdődő hasonló hosszú memóriarésszel. A <cím2> memóriarész tartalma **már** nem szerepel az összehasonlításban.

Példa: C 2000 2501 4000

D

A memória adott helyén levő gépi kódú utasítást mnemonik alakban állítja elő.

Szintaxis: **D** {<cím1>} {<cím2>}
A parancs a megadott memóriacímek közti tartományban levő gépi kódú programot fordítja vissza mnemonikos alakúvá. Ha értelmezhetetlen kódot talál, akkor ??? választ ad:

Példák: D 3000 3004
```
. 3000  A9 00    LDA #$00
. 3002  FF       ???
. 3003  D0 2B    BNE $3030
```

A visszafordított sorok előtti . lehetővé teszi, hogy azonnal módosíthassuk és újrafordíthassuk az utasítást.

F

Karakterfeltöltés.

Szintaxis: **F** <cím1> <cím2> <érték>

A monitor a memória <cím1>-től <cím2>-ig terjedő részét az <érték> byte-tal tölti fel. A <cím2>-be magába már nem töltődik be ez az érték. Az érték két byte-os hexadecimális szám lehet csak.

Példák: F 0400 0518 EA

G

Elindít egy gépi kódú programot.

Szintaxis: **G** {<cím>}

A vezérlés a <cím>-nél kezdődő gépi kódú rutinra kerül. Egészen egy gépi kódú RTS vagy BRK végrehajtásáig a vezérlés kikerül a gépi kódú monitor felügyelete alól. Ha nem adunk címet akkor a programszámláló értékétől fog elindulni.

H

Adott karaktersorozat keresése.

Szintaxis: **H** <cím1> <cím2> <adatok>

A monitor a <cím1>-től a <cím2>-ig terjedő memóriarészben keres az <adatok>-nak megfelelő byte sorozatot. Ha ilyent talál, a sorozat első elemének címét kiírja a képernyőre, majd folytatja a keresést. Az adatok vagy egymástól szóközzel elválasztott hexadecimális számok, vagy sztringkonstansok lehetnek.

Példák: H 8000 9000 "KUKAC"
```
H 2000 2400 EE EE EE FA
```

L

PRG típusú file betöltése.

Szintaxis: L <filenév> <egységszám>

A monitor a file első két byte-ja által meghatározott címtől kezdve betölti a programot.

Példák: L "ALAKZAT" 08 (töltés lemezről)
L "ADATOK" 01 (töltés kazettáról)

M

Kiírja a memória tartalmát.

Szintaxis: M {<cím1>} {<cím2>}

A parancs a <cím1>-től kezdődően kiírja a memória tartalmát mind hexadecimális, mind karakteres alakban. Ha a második paraméter elmarad, akkor a következő 96 memóriarekesz tartalmát írja ki. A kiírás formátuma megegyezik a > parancs formájával, így lehetőség nyílik a memória tartalmának azonnali módosítására.

Példák: M A048 A048
>A048 41 E7 00 AA AA 00 98 56 :A!.**..V

R

Kiírja a regiszterek tartalmát.

Szintaxis: R

A parancs kiírja az utasításszámláló, az állapotregiszter, az A, X, Y regiszterek, illetve a veremmutató értéket (ebben a sorrendben). A kiírás formátuma megfelel a ; utasítás formátumának, s ezért a regiszterek tartalma azonnal módosítható.

Példák: R
PC SR AC XR YR SP
; FB000 00 00 00 00 F8

S

A memória adott részének kiírása a háttértárakra.

Szintaxis: S <filenév> <egységszám> <cím1> <cím2>

A parancs a C-128 memóriájának <cím1>-től <cím2>-ig terjedő részét menti el az <egységszám>-nak megfelelő eszközre <filenév> filenév alatt. A <cím2> cím tartalma **már** nem mentődik el.

Példák: S "RAJZ" 08 2000 4000
S "KEPERNYO" 01 0400 0BFF

T

Memóriarész átmásolása.

Szintaxis: T <cím1> <cím2> <cím3>

A parancs a memória <cím1>-től <cím2>-ig terjedő részét másolja át a <cím3>-tól kezdődően. A <cím2> tartalma **már** nem kerül átmásolásra.

Példák: T 2000 2FFF 3000
T 1C00 1DFF 1E00

V

A memória tartalmának összehasonlítása egy file tartalmával.

Szintaxis: V <filenév> <egységszám> <cím>

A parancs összehasonlítja a memória tartalmát - a <cím>-től kezdődően - a file tartalmával. Az összehasonlítás az első eltérésnél abbamarad. Ebben az esetben ?VERIFY ERROR hibaüzenetet kapunk.

Példák: V "PROGRAM",08

X

Visszatérés a BASIC-be.

Szintaxis: X

A parancs kiadása után megint használhatjuk a BASIC programszerkesztőjét. A gépi kódú monitor parancsai pedig csak egy újabb MONITOR parancs kiadása után használhatók. A TEDMON önmaga nem módosítja a BASIC munkaterületet, ezért a program és a változók sértetlenül megmaradnak. Valamely TEDMON paranccsal természetesen tönkretehetjük a BASIC programot, illetve a változókat.

>

A memória tartalmának módosítása.

Szintaxis: > <cím> <adatok>

A <cím>-től kezdődően a memória tartalmát az <adatok>-ra cseréli át. Az <adatok> maximum 8, egymástól vesszővel elválasztott hexadecimális számot tartalmazhat.

Példák: > 0800 64 64 64 64
> 0400 03 03 03 03

;

A processzor regisztereinek tartalmát módosítja.

Szintaxis: ; <cím> <státusz> <A> <X> <Y> <veremmut.>

A parancs hatása módosítja a regiszterek értékét. Pontosabban arról van szó, hogy a G TEDMON parancs kiadása után a vezérlés a ; parancsban megadott <cím>-től, s az ott megadott regiszter tartalommal kezd el futni. A G paranccsal a <cím> még módosítható, de a regiszterek tartalma már nem.

7.2 KERNAL rutinok

Hasonlóan a Commodore 64-hez, a ROM utolsó lapját a rendszer egy ugrótáblának használja. Ezeket a rutinokat KERNAL rutinoknak hívjuk. Ha valaki ezeket használja gépi kódú programjának megírásakor, akkor az elég könnyen átvihető egyik Commodore típusú gépről a másikra.

A 15. szeleten a $FF81-től kezdődő belépési pontok használata megegyezik a Commodre 64-esével. Ezekről részletes információ található a **Commodore 64 II. kötet 317.-329., a Commodore 64 file-kezelés és input-output 76.-87., valamint a Commodore 64 assembly 139.-146. oldalain.** Itt az 'új' KERNAL rutinokról szólunk csak.

C64MODE

Belépési pont: $FF4D (65357)
Használt regiszterek: -
Előkészítő rutinok: -

Áttér C-64-s üzemmódba.

DMA-CALL

Belépesi pont: $FF50 (65360)
Használt regiszterek: X
Előkészítő rutinok: -

A bővítőbe helyezett memóriabővítőt inicializálja. X regiszter az MMU által használandó konfiguráció indexét tartalmazza.

BOOTCALL

Belépési pont: $FF53 (65363)
Használt regiszterek: -
Előkészítő rutinok: -

A gépi kódú rutin hívása egy paraméter nélküli BOOT BASIC parancs kiadásával ekvivalens.

PHOENIX

Belépési pont: $FF56 (65366)
Használt regiszterek: -
Előkészítő rutinok: -

A C-128 hideg indítása.

LKUPLA

Belépési pont: $FF59
Használt regiszterek: A,X,Y
Előkészítő rutin: OPEN

A rutin meghívásakor az akkumulátor egy logikai file számot tartalmaz. Ha visszatéréskor a C bit magas, akkor a szóbanforgó logikai file-t még nem nyitottuk meg. Ha alacsony, akkor igen, és X-ben az egységszám, Y-ban a csatornaszám található.

LKUPSA

Belépési pont: $FF5C
Használt regiszterek: A,X,Y
Előkészítő rutin: OPEN

A rutin meghívásakor az Y regiszter egy csatorna számot tartalmaz. Ha visszatéréskor a C bit magas, akkor a szóbanforgó file-t még nem nyitottuk meg. Ha alacsony, akkor igen, és X-ben az egységszám, Az A regiszterben pedig a logikai file-szám található.

SWAPPER

Belépési pont: $FF5F (65375)
Használt regiszterek: -
Előkészítő rutinok: -

Csere a 40 és 80 oszlopos képernyő közt. A rendszer újra írja a $E0-$FA és $0A40-$0A5A közti regisztereket. Hatása ekvivalens az <ESC-X> billentyűzéssel.

DLCHR

Belépési pont: $FF62 (65378)
Használt regiszterek: -
Előkészítő rutinok: -

A CHARROM-ot átmásolja a 80 oszlopos VDC chip megfelelő helyére.

PFKEY

Belépési pont: $FF65 (65381)
Használt regiszterek: A,X,Y
Előkészítő rutinok: -

Segítségével újradefiniálhatjuk az <F1>-<F10> funkciós billentyűkhöz, valamint a <RUN> és a <HELP> billentyűkhöz rendelt szövegeket. A rutin meghívásakor A a 0.lapon levő mutató címét tartalmazza. Ez mutat sztring első karakterére. Az X regiszter a billentyű sorszámát (1-12) tartalmazza, míg Y a sztring hosszát.

SETBNK

Belépési pont: $FF68 (65384)
Használt regiszterek: A,Y
Előkészítő rutinok: -

Valamennyi LOAD, SAVE, VERIFY és OPEN rutin előtt meg kell hívni ezt a rutint, ami megmondja, hogy milyen MMU konfigurációt használjanak a fenti rutinok. Y regiszterbe kell tölteni a filenevet, A-ba pedig a használandó memória szeletnek az indexét.

GETCONF

Belépési pont: $FF6B (65387)
Használt regiszterek: A,X
Előkészítő rutinok: -

Hivásкor az X regiszterbe kell tölteni egy 1-16 közötti szelet számot. A rutin azzal az értékkel tér vissza az A-ban, amit az MMU chip $FF00 regiszterébe kell tölteni, hogy a X-nek megfelelő szeletet kapjuk.

JSRFAR

Belépési pont: $FF6E (65390)
Használt regiszterek: $02-$09
Előkészítő rutinok: -

A $02-$09 regiszterek által megadott szelet megadott címére történik ugrás egy JSR-nal. Visszatérés után a rutin visszaállítja az eredeti konfigurációt. Lásd részletesen a 7.3 paragrafust!

JMPFAR

Belépési pont: $FF71 (65393)
Használt regiszterek: $02-$09

Előkészítő rutinok: -

A $02-$09 regiszterek által megadott szelet megadott címére történik ugrás egy JMP-pal. Lásd részletesen a 7.3 paragrafust!

INDFET

Belépési pont: $FF74 (65396)
Használt regiszterek: A,X,Y
Előkészítő rutinok: -

A 0.lapon levő mutató által mutatott, az X regiszternek megfelelő szeletről hoz egy byte-ot az A regiszterbe. A mutató értékét az Y regiszterrel módosítja. Lásd részletesen a 7.3 paragrafust!

INDSTA

Belépési pont: $FF77 (65399)
Használt regiszterek: $02-$09
Előkészítő rutinok: -

A 0.lapon levő mutató által mutatott, az X regiszternek megfelelő szeletre írja ki az A regiszter tartalmát. A mutató értékét az Y regiszterrel módosítja. Lásd részletesen a 7.3 paragrafust!

INDCMP

Belépési pont: $FF7A (65402)
Használt regiszterek: A,X,Y
Előkészítő rutinok: -

A 0.lapon levő mutató által mutatott, az X regiszternek megfelelő szeleten levő byte-ot hasonlítja össze az A regiszterrel. Az összehasonlítás eredményéül kapott status byte-ot a $02C8-ba helyezi. A mutató értékét az Y regiszterrel módosítja. Lásd részletesen a 7.3 paragrafust!

PRIMM

Belépési pont: $FF7D (65402)
Használt regiszterek: -
Előkészítő rutinok: -

Adott szöveg kiírása. A szöveget a szubrutinhívást követően kell elhelyezni. A szöveg végét egy $00 byte jelzi.

7.3 A C-128 memória szervezése

7.3.1 Az MMU chip működése

A Commodore 128 - a Commodore 64-hez képest - két új perifériakezelő chipet tartalmaz. Az egyik a 80 oszlopos képernyő kezelését oldja meg, ezt a 6.2 paragrafusban ismertetjuk. A másik chip a memória kezelését végzi, neve MMU: **m**emory **m**anagement **u**nit.

Mint ismeretes a C-128 mindkét processzora 8-bites. Ennek megfelelően 16 bites címbuszuk és 8 bites adatbuszuk van. A processzor tehát egyidejűleg 64Kbyte 8 bites adat kezelésére képes. A C-128 ezen úgy segít, hogy az MMU a processzor területére más- és más memóriarészeket lapoz be. Így pl. a $4000-es cím lehet a ROM-ban, lehet az 1. RAM-ban, illetve a 2. RAM-ban is. Az I/O területen levő címek még több részhez tartozhatnak.

Az MMU regiszterei két különböző részén találhatók a memóriának, s ezek rendeltetése eltérő:

MMU regiszterek

$D50B	VR	verzió regiszter
$D50A	P1H	verem mutató: felső fél byte
$D509	P1L	verem mutató: alsó byte
$D508	P0H	0. lap mutató: felső fél byte
$D507	P0L	0. lap mutató: alsó byte
$D506	RCR	RAM konfigurációs regiszter
$D505	MCR	mód konfigurációs regiszter
$D504	PCRD	előkonfigurációs regiszter D
$D503	PCRC	előkonfigurációs regiszter C
$D502	PCRB	előkonfigurációs regiszter B
$D501	PCRA	előkonfigurációs regiszter A
$D500	CR	konfigurációs regiszter (másolata $FF00-ban)

MMU 'felső' regiszterek

```
        +---------+
$FF04   |  LCRD   |  töltő regiszter D
        +---------+
$FF03   |  LCRC   |  töltő regiszter C
        +---------+
$FF02   |  LCRB   |  töltő regiszter B
        +---------+
$FF01   |  LCRA   |  töltő regiszter A
        +---------+
$FF00   |   CR    |  Konfigurációs regiszter ($D501 másolata)
        +---------+
```

Konfigurációs regiszterek
A memória kiosztását, azt hogy mely címeken milyen fizikai memória részek jelenjenek meg, a konfigurációs regiszterek határozzák meg. A regiszter egyes bitjei az alábbiak szerint működnek:

7. és 6. bit A két bit együttesen megadja, hogy a processzor melyik 64Kbyte-os RAM-ot használja. A 00 bitkombináció a 0. RAM-nak felel meg, a 01 az 1. RAM-nak. Ha bővítést használunk, akkor további két 64Kbyte-os RAM memória címezhető meg. Ezeknek az 10, illetve 11 bitkombinációk felelnek meg. Ha nem használunk memória bővítést, akkor az 10 kombináció a 00-nak, az 11 pedig a 01-nek felel meg.

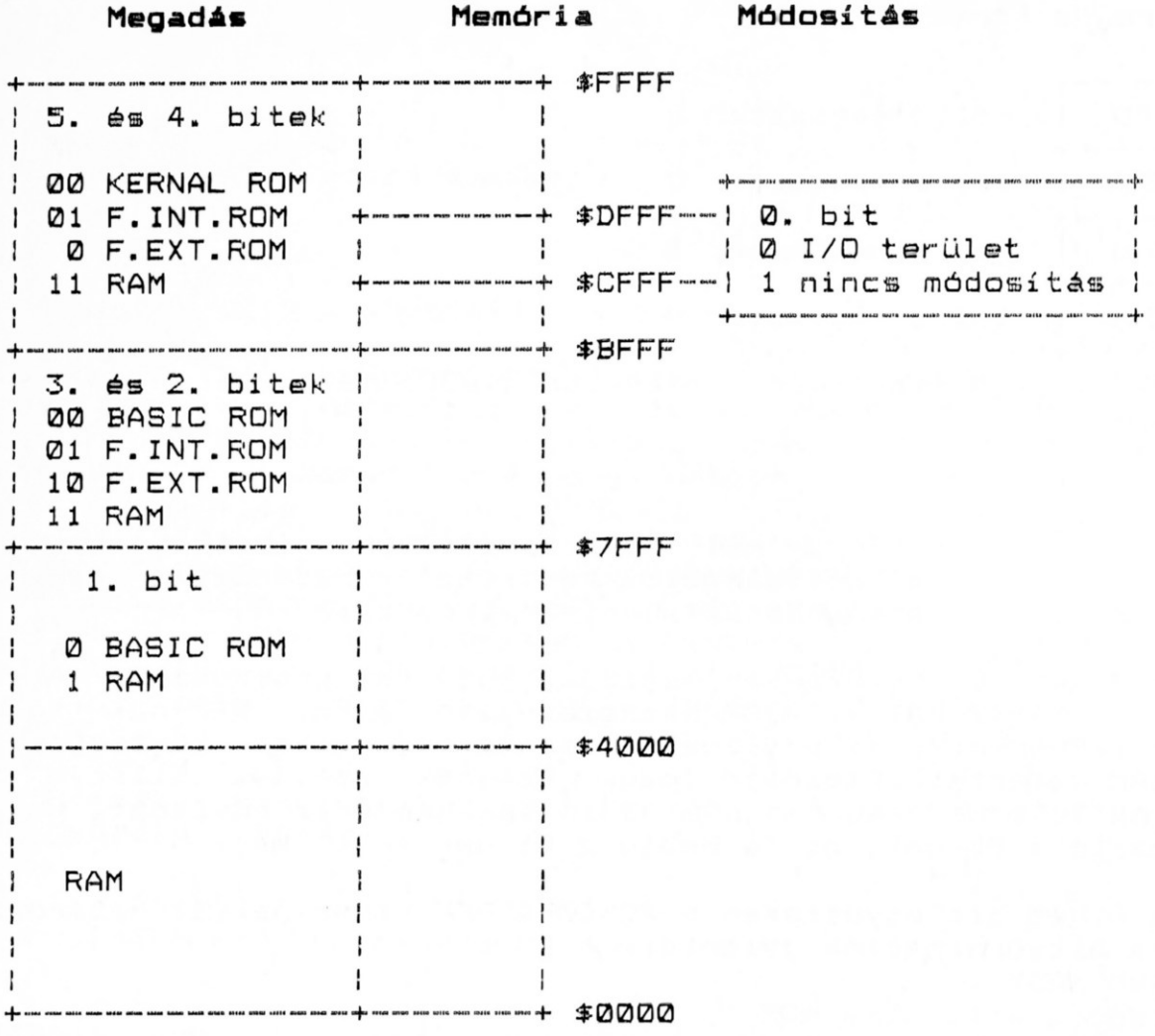
5. és 4. bit A két bit együttesen a $C000-$FFFF címek helyét határozza meg. Az egyes bitkombinációk jelentése a következő:
  00 = KERNAL ROM
  01 = Belső funkcionális ROM
  10 = Külső funkcionális ROM
  11 = RAM (amit a 6. és 7. bit definiál)

3. és 2. bit A két bit együttesen a $8000-$BFFF címek helyét határozza meg. Az egyes bitkombinációk jelentése a következő:
  00 = BASIC ROM
  01 = Belső funkcionális ROM
  10 = Külső funkcionális ROM
  11 = RAM (amit a 6. és 7. bit definiál)

1. bit A $4000-$7FFF címek helyét határozza meg:
  0 = BASIC ROM
  1 = RAM (amit a 6. és 7. bit definiál)

0. bit A $D000-$DFFF címek helyét módosítja:
  0 = Az I/O chipek
  1 = a 5-4. bitek által meghatározott memóriarész.

A konfigurációs regiszter működését a következő ábrán szemléltetjük:



A konfigurációs regiszter két példányban is létezik. Ennek oka, hogy ha a konfigurációs regiszter csak a $D500 címen létezne,akkor az I/O-t területet folyton bekapcsolva kellene tartanunk, különben az MMU használhatatlan lenne. Ezt elkerülendő a gépet úgy tervezték meg, hogy az utolsó lapon, azaz a $FF00-$FFFF címeken mindig ugyanaz látszik.Nevezetesen az MMU felső regiszterei,illetve a KERNAL ugrótáblája.Kivételt csak a C-64-es üzemmód képez,amikor ezeken a címeken a C-64 KERNAL ROM-ja van. Ezért a C-64 üzemmódban az MMU - sajnos - nem használható.

Másik szempont a konfiguráció váltás meggyorsítása volt. A ROM-ban levő BASIC például a RAM-ban levő programokat hajtja végre, s emiatt folyton lapoznia kell. Ennek meggyorsítására szolgálnak a töltő konfigurációs regiszterek.

Adott memória konfiguráció kiválasztására három lehetőségünk kínálkozik. Az első esetben a processzor területén található az MMU chip, azaz az I/O területet bekapcsoltuk. Ebben az esetben a kívánt konfigurációnak megfelelő byte-ot be kell írni a konfigurációs regiszterbe ($D500). Ez az érték azonnal megjelenik a felső konfigurációs regiszterben ($FF00) is.

A második esetben az I/O terület nincs bekapcsolva, s ilyen módon a konfigurációs regiszterbe nem tudunk írni.Ehelyett a $FF00-on levő konfigurációs regiszterbe kell írnunk,s a rendszer ezt azonnal átmásolja a $D500-ba is! Megfordítva, a $FF00 címet olvasva megtudhatjuk,hogy milyen aktuális memória konfigurációval dolgozik a gép.

Harmadik esetben a töltő konfigurációs regisztereket használjuk. Ezek az előkonfigurációs regiszterrel vannak kapcsolatban, majdnem úgy, mint a konfigurációs regiszter két példánya. Használatuk azonban eltér. Olvasáskor a töltő konfigurációs regiszterek az elő-konfigurációs regiszterek tartalmát adják vissza, még akkor is, ha az I/O terület nincs bekapcsolva. A töltő konfigurációs regiszterekbe való írásnak speciális jelentése van. **Hatására a megfelelő elő-konfigurációs regiszter tartalma átmásolódik a konfigurációs regiszterbe!** A konfigurációt tehát pusztán azáltal tudjuk módosítani, hogy a megfelelő töltő konfigurációs regiszterbe írunk. Eközben az I/O területnek ugyancsak nem kell bekapcsolva lennie. Természetesen az előkonfigurációs regiszterek tartalmát csak olyankor tudjuk beállítani, mikor az MMU chip a processzor területén van.

Pl. a STX $FF01 hatására a $D501-en levő, A előkonfigurációs regiszter tartalma a $D500 és $FF00 konfigurációs regiszterekbe íródik. Megjegyezzük, hogy $FF01 tartalma nem változik meg. Ezzel a lehetőséggel élve a memória átkonfigurálását - 4 előre megadott konfigurációra - 2 gépi ciklus alatt bármilyen memóriakonfiguráció mellett elvégezhetjük. Ehhez a proceszor regisztereinek tartalmát nem kell módosítani.

Mód konfigurációs regiszter
Ennek a regiszternek a tartalma a számítógép működésének további paramétereit adja meg. Az egyes bitek jelentése a következő:

7. bit Ez a bit érzékeli a <40/80 DISPLAY> billentyű állapotát.
0 = lenyomott (80 oszlopos)
1 = felengedett (40 oszlopos)

6. bit Ennek segítségével lehet a C-64 és C-128 üzemmód közt választani. Ha a C-64-es üzemmódot választjuk, akkor a C-64-es ROM-ok kapcsolódnak be, s az MMU chip automatikusan kikapcsolódik. Egy RESET vagy egy POWER UP a C-128-as üzemmódot kapcsolja be.
0 = C-128-as üzemmód
1 = C-64-es üzemmód

5. és 4. bit A bitek a bővítő egység GAME és EXROMIN vonalait érzékelik. Ha ezek egyike alacsony, akkor egy C-64-es cartridge-t helyeztünk a bővítő egységbe, s a rendszer RESET és POWER UP után C-64-es üzemmódba tér át (szoftver úton). A C-128-as üzemmód nem használja ezeket.

3. bit A gyors soros átvitel irányát vezérli. Ha a Commdore küld a perifériának, akkor a Commodore adja az adatbitek shifteléséhez szükséges órajelet.
0 = input (a C-128 kap adatokat)
1 = output (a C-128 küld adatokat)

2. és 1. bit nem használt.

0. bit a használt processzor típusát adja meg.
0 = Z80-A
1 = M8502

A Z80-B processzor bekapcsolásával egyidejűleg a processzor területére egy 4 Kbyte-os ún. BIOS ROM kapcsolódik be. Ez egyébként a $D000-$DFFF

címeken található, de a Z80-A címei a C-128-hoz képest el vannak tolva, s ezért ez a Z80-A 0000H-$0FFFH címein jelenik meg. Az M8502 processzor működése felfüggesztődik, a Z80-A pedig végrehajt egy RESET 0 gépi kódú utasítást.

RAM konfigurációs regiszter

A RAM konfigurációs regiszter legfontosabb feladata a **közös RAM részek** meghatározása. Ahhoz, hogy a különböző memória részeket együtt tudjuk használni, valamilyen módon kapcsolatot kell találnunk köztük. Erre a közös RAM részek kijelölése szolgál. A közös részekbe történő írás egyszerre több fizikai memóriába is megtörténik. Ennek köszönhetően a 64Kbyte-os memória szeletek közt információt lehet átvinni. A közös részen általában a legfontossabb rendszerváltozók és rutinok találhatók. A C-128 esetén a $0000-$03FFF-ig terjedő rész közös. Itt találhatók a rendszerváltozók és a lapozást segítő rutinok.

A közös rész nagyságát és elhelyezkedését, továbbá egyéb paramétereket is a RAM konfigurációs regiszterrel lehet beállítani.

7. és 6. bit A VIC chip által használt memóriaszelet kiválasztása. A VIC chip - szemben a VDC chippel - működése a processzor területből igényel írható-olvasható memóriát. Hogy ez a lehetséges négy 64Kbyte-os RAM szelet közül melyiken található, azt adja meg ez a két bit.

00 = RAM 0. szelet
01 = RAM 1. szelet
10 = RAM 2. szelet (bővítés esetén, különben 0. szelet)
11 = RAM 3. szelet (bővítés esetén, különben 1. szelet)

5. és 4. bit Az MMU segítségével közvetlenül 4 darab 64Kbyte-os RAM memóriaszelet érhető el. Ez a két vonal lehetőséget biztosít további 3*4 darab 64Kyte-os memória szelet illesztésére. Így végül az MMU chip 1Mbyte-os RAM-ot tud kezelni. A C-128-as gép - alapkiépítésben - nem használja ezt a két vonalat.

3. és 2. bit Megadja, hogy az 1-0. bitek által meghatározott nagyságú közös rész a memória elején, végén vagy mindkettőn megjelenjen:

00 = nincs közös rész (függetlenül az 10. bitektől)
01 = a memória eleje közös
10 = a memória vége közös
11 = a memória eleje is és a vége is közös

1. és 0. bit A közös rész nagyságát határozzak meg. Ez nem pontos kifejezés. Ha a 3. és 2. bitek 0-k, akkor nincs is közös rész. Ha pedig mindkettő 1, akkor a közös memória rész az itt jelzett kétszerese; a memória két vége közös:

00 = 1Kbyte
01 = 4Kbyte
10 = 8Kbyte
11 = 16Kbyte.

Verem és 0.lap mutató

Az MMU lehetőséget biztosít a verem és 0.lap áthelyezésére. Mint ismeretes, a 0.lapot ($0000-$00FF), illetve az 1.lapot, azaz a vermet ($0100-$01FF) a 8502-es processzor speciálisan használja. Elsősorban a 0.lap 'tele' van, az interpreter szinte valamennyi byte-ot elhasználja. Lehetőség van ezeket áthelyezni máshová. Ekkor a 0.lapos

és a vermet használó műveletek erre a memória részre fognak vonatkozni, s az eredeti 0.lap és verem érintetlenül megmarad.

Mindegyik helyét 12 bites mutatóval adhatjuk meg. A 0.lap mutatójának felső 4 bitjét a $D508, az alsó byte-ját a $D507 regiszter tartalmazza. Hasonlóan a verem (1.lap) mutatójának felső 4 bitjét a $D50A, alsó byte-ját a $D509 regiszter tartalmazza.

Verzió regiszter
A regiszter 7-4. bitjei a beépített összes RAM nagyságát adják meg. Ez az érték $2, utalva arra, hogy 2 darab 64Kbyte-os RAM van a rendszerben.

A 3-0. bitek az MMU chip verziószámát adják meg, ez 0. így ennek a regiszternek a tartalma $20.

7.3.2 A C-128 BASIC interpreter memória felhasználása

Ha a C-128-as módban dolgozunk, akkor a két RAM első 1Kbyte-os része közös, azaz a $0000-$03FFF címek. Az interpreter ezért ezeken a címeken helyezi el azokat a programrészeket, amelyek segítségével bármelyik memória részről tud az akkumulátorba (A regiszter) byte-ot olvasni, onnan a memóriába írni, vagy összehasonlítani a memória tartalmával. Ezeknek a meghívása előtt azonban be kell állítani néhány 0. lapon (közös terület!) levő mutatót.

Memória olvasása
A rutin meghívása előtt a $02AA címre egy 0.lapon levő mutatót kell tölteni. X regiszter a kért konfigurációt, Y pedig a 0-ás lapon levő mutató címét kell tatalmaznia. Ha például a mutató a $fe címen található, az Y értéke 0, s a RAM 1. szeletéből akarjuk a byte-ot olvasni, akkor a rutint a következőképpen kell meghívni:

```
LDA #$FE
STA $02AA
LDY #$00
LDX #$7F    ; ez felel meg a RAM 1 bekapcsolásának
JSR $02A2
```

Maga a rutin a következőképpen működik:

```
02a2:  ad 00 ff    lda $ff00    ; az aktuláis konfiguráció A-ban
02a5:  8e 00 ff    stx $ff00    ; az új konfiguráció beállítás
02a8:  aa          tax          ;  s tárolása X-ben
02a9:  b1 ff       lda ($ff),y  ; a byte beolvasása
02ab:  8e 00 ff    stx $ff00    ; a régi konfig. visszaállítása
02ae:  60          rts          ; visszatérés az alprogrmból
```

Memória írása
Hasonlóképpen írhatunk tetszőleges memóriaszeletre. Ehhez a $02B9 bytera egy 0. lapon levő mutatót kell beírni. X-ben az új konfigurációnak megfelelő byte-ot kell elhelyezni, Y-nal pedig megnövekszik a mutató értéke. A tárolandó byte értékét az A regiszterbe kell rakni. Ha az előbbi memóriarekeszbe most írni szeretnénk azt így tehetjük meg:

```
    LDA #$FE
    STA $02B9
    LDY #$00
    LDX #$7F    ; ez felel meg a RAM 1 bekapcsolásának
    JSR $02BF
```

Maga a programrész így működik:

```
02bf:  48          pha           ; a tárolandó byte verembe mentése
02b0:  ad 00 ff    lda $ff00     ; az aktuális konfiguráció A-ban
02b3:  8e 00 ff    stx $ff00     ; az új konfiguráció beállítása
02b6:  aa          tax           ; régi konfiguráció x-ben
02b7:  68          pla           ; kiírandó byte A-ban
02b8:  91 ff       sta ($ff),y   ; a byte kiírása
02ba:  8e 00 ff    stx $ff00     ; régi konfig. visszaállítása
02bd:  60          rts           ; visszatérés az alprogramból
```

Memória és az akkumulátor összehasonlítása

Hasonló elvekre épül az a rutin is, amelyik az akkumulátor tartalmát egy tetszőleges szeleten levő byte-tal hasonlítja össze. A 0.lapon levő mutató címét most a $02C8 címre kell beírni, Y-nal növeljük a mutatót, X tartalmazza a konfigurációt, az A regiszter az összehasonlítandó byte-ot:

```
02be:  48          pha           ; az adatbyte verembe mentése
02bf:  ad 00 ff    lda $ff00     ; konfiguráció az A-ban
02c2:  8e 00 ff    stx $ff00     ; új konfiguráció beállítása
02c5:  aa          tax           ; a régi konf. tárolása X-ben
02c6:  68          pla           ; adatbyte visszahozása
02c7:  d1 ff       cmp ($ff),y   ; összehasonlítása
02c9:  8e 00 ff    stx $ff00     ; a régi konfig. visszaállítása
02cc:  60          rts           ; visszatérés az alprogramból
```

Programvezérlés átadása tetszőleges szeletbe

A 2. lapon további két rutin található. Egyik egy ugrást, a másik egy alprogramhívást hajt végre egy tetszőleges szeletre. Ebben az esetben a szelet a BANK utasításban szereplő érték. Ez nem íródik közvetlenül a konfigurációs regiszterbe, hanem a $F7F0 táblázatnak megfelelő módon konvertálja a rendszer. Szemben tehát a fenti rutinokkal, nem X, hanem $F7F0,X értéke kerül a konfigurációs regiszterbe.

A paraméterek átadásának is más a módja. A processzor regiszterek tartalma érdektelen. A $02-$09 címek tartalma határozza meg a vezérlésátadást, s hogy mi lesz az A,X,Y,SP és PSW regiszterek tartalma:

- $02 = szelet száma
- $03 = programszámláló: alsó byte
- $04 = programszámláló: felső byte
- $05 = processzor státuszszó (PSW)
- $06 = A regiszter
- $07 = X regiszter
- $08 = Y regiszter
- $09 = veremmutató (SP regiszter)

Az ugrást végrehajtó rutin a $02E3, az alprogramhívó változat a $02CD címen kezdődik.

```
02cd:  20 e3 02   jsr $02e3   ; ugrás tetszőleges szeletre
02d0:  85 06      sta $06     ; A-regiszter elmentése
02d2:  86 07      stx $07     ; X-regiszter elmentése
02d4:  84 08      sty $08     ; Y-regiszter elmentése
02d6:  08         php         ; PSW a verembe
02d7:  68         pla         ;  kivesszük, így A-ban
02d8:  85 05      sta $05     ;  végül elmentjük
02da:  ba         tsx         ; SP értéke X-ben,
02db:  86 09      stx $09     ;  majd elmentjük
02dd:  a9 00      lda #$00    ; #$00-s konfig. kiválasztása
02df:  8d 00 ff   sta $ff00   ;  s bekapcsolása (ROM-ok)
02e2:  60         rts         ; visszatérés az alprogramból

                              ; ugrás tetszőleges szeletbe
02e3:  a2 00      ldx #$00    ; az utasításszámláló és a
02e5:  b5 03      lda $03,x   ;  PSW értékének
02e7:  48         pha         ;  a verembe írása
02e8:  e8         inx         ; következő index
02e9:  e0 03      cpx #$03    ; kell -e még írni
02eb:  90 f8      bcc $02e5   ; igen --> vissza
02ed:  a6 02      ldx $02     ; szelet szám X-ben
02ef:  20 6b ff   jsr $ff6b   ; a szeletnek megfelelő konfigu-
02f3:  8d 00 ff   sta $ff00   ; ráció beállítása
02f6:  a5 06      lda $06     ; A regiszter beállítása
02f8:  a6 07      ldx $07     ; X regiszter beállítása
02fa:  a4 08      ldy $08     ; Y regiszter beállítása
02fb:  40         rti         ; a verembe tárolt értékeknek
                              ;  megfelelő vezérlésátadás

                              ; a szeletnek megfelelő konfigu-
                              ;  rációs index előállítása
ff6b:  4c ec f7   jmp $f7ec   ; KERNAL belépési ponton át

f7ec:  bd f0 f7   lda $f7f0,x ; az érték betöltése A-ba
f7ef:  60         rts         ; visszatérés az alprogramból
                              ; a táblázat elemeinek jelentése
f7f0:  3f         .byte %00111111 ; RAM 0
f7f1:  7f         .byte %01111111 ; RAM 1
f7f2:  bf         .byte %10111111 ; RAM 2 (=RAM 0)
f7f3:  ff         .byte %11111111 ; RAM 3 (=RAM 1)
f7f4:  16         .byte %00010110 ; F.INT.ROM, RAM 0, I/O
f7f5:  56         .byte %01010110 ; F.INT.ROM, RAM 1, I/O
f7f6:  96         .byte %10010110 ; F.INT.ROM, RAM 2, I/O
f7f7:  d6         .byte %11010110 ; F.INT.ROM, RAM 3, I/O
f7f8:  2a         .byte %00101010 ; F.EXT.ROM, RAM 0, I/O
f7f9:  6a         .byte %01101010 ; F.EXT.ROM, RAM 1, I/O
f7fa:  aa         .byte %10101010 ; F.EXT.ROM, RAM 2, I/O
f7fb:  ea         .byte %11101010 ; F.EXT.ROM, RAM 3, I/O
f7fc:  06         .byte %00000110 ; KERNAL,F.INT.ROM(alsó),RAM 0,I/O
f7fd:  0a         .byte %00001010 ; KERNAL,F.EXT.ROM(alsó),RAM 0,I/O
f7fe:  01         .byte %00000001 ; KERNAL,BASIC,RAM 0, CHARROM
f7ff:  00         .byte %00000000 ; KERNAL,BASIC,RAM 0, I/O
```

FÜGGELÉK

F.1 BASIC alapszavak kulcsszó szerint rendezve

A 'képernyő' címszó alatt a rövidítés használatakor a képernyőn látható alakot adjuk meg; feltéve, hogy a 'kisbetűk/nagybetűk' karakterkészletet használjuk. A 'nagybetűk/grafikák' karakterkészlet használata esetén a siftelt (emelt) betűk helyén grafikus jelek láthatók.

Parancs	Rövidítés	Képernyő	Függvénytípus	Token(ek) C-64 mód	Token(ek) C-128 mód
ABS	A SHIFT B	aB	Numerikus	$B6(182)	$B6(182)
AND	A SHIFT N	aN		$AF(175)	$AF(175)
APPEND	A SHIFT P	aP			$FE $0E(254,14)
ASC	A SHIFT S	aS	Numerikus	$C6(198)	$C6(198)
ATN	A SHIFT T	aT	Numerikus	$C1(193)	$C1(193)
AUTO	A SHIFT U	aU			$DC(220)
BACKUP	BA SHIFT C	baC			$F6(246)
BANK	B SHIFT A	bA			$FE $02(254,2)
BEGIN	B SHIFT E	bE			$FE $18(254,24)
BEND	BE SHIFT N	beN			$FE $19(254,25)
BLOAD	B SHIFT L	bL			$FE $11(254,17)
BOOT	B SHIFT O	bO			$FE $1B(254,27)
BOX	nincs	box			$E1(225)
BSAVE	B SHIFT S	bS			$FE $10(254,16)
BUMP	B SHIFT U	bU	Numerikus		$CE $03(206,3)
CATALOG	C SHIFT A	cA			$FE $0C(254,12)
CHAR	CH SHIFT A	chA			$E0(224)
CHR$	C SHIFT H	cH	Sztring	$C7(199)	$C7(199)
CIRCLE	C SHIFT I	cI			$E2(226)
CLOSE	CL SHIFT O	clo		$A0(160)	$A0(160)
CLR	C SHIFT L	cL		$9C(156)	$9C(156)
CMD	C SHIFT M	cM		$9D(157)	$9D(157)
COLLECT	COL SHIFT L	colL			$F3(243)
COLLISION	CO SHIFT L	coL			$FE $17(254,23)
COLOR	COL SHIFT O	colL			$E7(231)
CONCAT	C SHIFT O	cO			$FE $13(254,19)
CONT	nincs	cont		$9A(154)	$9A(154)
COPY	CO SHIFT P	coP			$F4(244)
COS	nincs	cos	Numerikus	$BE(190)	$BE(190)
DATA	D SHIFT A	dA		$83(131)	$83(131)
DCLEAR	DCL SHIFT E	dclE			$FE $15(254,21)
DCLOSE	D SHIFT L	dL			$FE $0F(254,15)
DEC	nincs	dec	Numerikus		$D1(209)
DEF	D SHIFT E	dE		$96(150)	$96(150)
DELETE	DE SHIFT L	deL			$F7(247)
DIM	D SHIFT I	dI		$86(134)	$86(134)

Parancs	Rövidítés	Képernyő	Függvénytípus	Token(ek) C-64 mód	Token(ek) C-128 mód
DIRECTORY	DI SHIFT R	diR			$EE(238)
DLOAD	D SHIFT L	dL			$F0(240)
DO	nincs	do			$EB(235)
DOPEN	D SHIFT O	dO			$FE $0D(254,13)
DRAW	D SHIFT R	dR			$E5(229)
DS	nincs	ds	Numerikus		
DS$	nincs	ds$	Sztring		
DSAVE	D SHIFT S	dS			$EF(239)
DVERIFY	D SHIFT V	dV			$FE $14(254,20)
EL	nincs	el	Numerikus		
ELSE	E SHIFT L	eL			$D5(213)
END	nincs	end		$80(128)	$80(128)
ENVELOPE	E SHIFT N	eN			$FE $0A(254,10)
ER	nincs	er	Numerikus		
ERR$	nincs	err$	Sztring		
EXIT	E SHIFT X	eX			$ED(237)
EXP	E SHIFT X	eX	Numerikus	$BD(189)	$BD(189)
FAST	F SHIFT A	fA			$FE $25(254,37)
FETCH	F SHIFT E	fE			$FE $21(254,33)
FILTER	F SHIFT I	fI			$FE $03(254,3)
FN	nincs	fn		$A5(165)	$A5(165)
FOR	F SHIFT O	fO		$81(129)	$81(129)
FRE	F SHIFT R	fR	Numerikus	$B8(184)	$B8(184)
GET	G SHIFT E	gE		$A1(161)	$A1(161)
GETKEY	GETK SHIFT E	getkE			
GET#	nincs				
GO	nincs			$CB(203)	$CB(203)
GO64	nincs				
GOSUB	GO SHIFT S	goS		$8D(141)	$8D(141)
GOTO	G SHIFT O	gO		$89(137)	$89(137)
GRAPHIC	G SHIFT R	gR			$DE(222)
GSHAPE	G SHIFT S	gS			$E3(227)
HEADER	HE SHIFT A	heA			$F1(241)
HELP	HE SHIFT L	heL			$EA(234)
HEX$	H SHIFT E	hE	Sztring		$D2(210)
IF	nincs	if		$8B(139)	$8B(139)
INPUT	nincs	input		$85(133)	$85(133)
INPUT#	I SHIFT N	iN		$84(132)	$84(132)
INSTR	IN SHIFT S	inS	Numerikus		$D4(212)
INT	nincs	int	Numerikus	$B5(181)	$B5(181)
JOY	J SHIFT O	jO	Numerikus		$CF(207)
KEY	K SHIFT E	kE			$F9(249)
LEFT$	LE SHIFT F	leF	Sztring	$C8(200)	$C8(200)
LEN	nincs	len	Numerikus	$C3(195)	$C3(195)
LET	L SHIFT E	lE		$88(136)	$88(136)
LIST	L SHIFT I	lI		$9B(155)	$9B(155)
LOAD	L SHIFT O	lO		$93(147)	$93(147)
LOCATE	LO SHIFT C	loC			$E6(230)
LOG	nincs	log	Numerikus	$BC(188)	$BC(188)
LOOP	LO SHIFT O	loO			$EC(236)
MID$	M SHIFT I	mI	Sztring	$CA(202)	$CA(202)
MONITOR	MO SHIFT N	moN			$FA(250)
MOVSPR	M SHIFT O	mO			$FE $06(254,6)
NEW	nincs	new		$A2(162)	$A2(162)
NEXT	N SHIFT E	nE		$82(130)	$82(130)
NOT	N SHIFT O	nO		$A8(168)	$A8(168)

Parancs	Rövidítés	Képernyő	Függvénytípus	Token(ek) C-64 mód	Token(ek) C-128 mód
ON	nincs	on		$91(145)	$91(145)
OPEN	O SHIFT P	oP		$9F(159)	$9F(159)
OR	nincs	or		$B0(176)	$B0(176)
PAINT	P SHIFT A	pA			$DF(223)
PEEK	PE SHIFT E	peE	Numerikus	$C2(194)	$C2(194)
PEN	P SHIFT E	pE	Numerikus		$CE $04(206,4)
PLAY	P SHIFT L	pL			$FE $04(254,4)
POINTER	PO SHIFT I	poI	Numerikus		$CE $0A(206,10)
POKE	PO SHIFT K	poK		$97(151)	$97(151)
POS	nincs	pos	Numerikus	$B9(185)	$B9(185)
POT	P SHIFT O	pO	Numerikus		$CE $02(206,2)
PRINT	?	?		$99(153)	$99(153)
PRINT#	P SHIFT R	pR		$98(152)	$98(152)
PRINTUSING	?US SHIFT I	?usI			$99 $FB
PUDEF	P SHIFT U	pU			$DD(221)
RCLR	R SHIFT C	rC	Numerikus		$CD(205)
RDOT	R SHIFT D	rD	Numerikus		$D0(208)
READ	RE SHIFT A	reA		$87(135)	$87(135)
RECORD#	R SHIFT E	rE			$FE $12(254,18)
REM	nincs	rem		$8F(143)	$8F(143)
RENAME	RE SHIFT N	reN			$F5(245)
RENUMBER	REN SHIFT U	renU			$F8(248)
RESTORE	RE SHIFT S	reS		$8C(140)	$8C(140)
RESUME	RES SHIFT U	resU			$D6(214)
RETURN	RE SHIFT T	reT		$8E(142)	$8E(142)
RGR	R SHIFT G	rG	Numerikus		$CC(204)
RIGHT$	R SHIFT I	rI	Sztring	$C9(201)	$C9(201)
RND	R SHIFT N	rN	Numerikus	$BB(187)	$BB(187)
RREG	R SHIFT R	rR			$FE $09(254,9)
RSPCOLOR	RSP SHIFT C	rspC	Numerikus		$CE $07(206,7)
RSPPOS	R SHIFT S	rS	Numerikus		$CE $05(206,5)
RSPRITE	RSP SHIFT R	rspR	Numerikus		$CE $06(206,6)
RUN	R SHIFT U	rU		$8A(138)	$8A(138)
RWINDOW	R SHIFT W	rW	Numerikus		$CE $09(206,9)
SAVE	S SHIFT A	sA		$94(148)	$94(148)
SCALE	SC SHIFT A	scA			$E9(233)
SCNCLR	S SHIFT C	sC			$E8(232)
SCRATCH	SC SHIFT R	scR			$F2(242)
SGN	S SHIFT G	sG	Numerikus	$B4(180)	$B4(180)
SIN	S SHIFT I	sI	Numerikus	$BF(191)	$BF(191)
SLEEP	S SHIFT L	sL			$FE $0B(254,11)
SLOW	SL SHIFT O	slO			$FE $26(254,38)
SOUND	S SHIFT O	sO			$DA(218)
SPC(	SP SHIFT C	spC	Speciális	$A6(166)	$A6(166)
SPRCOLOR	SPR SHIFT C	sprC			$FE $08(254,8)
SPRDEF	SPR SHIFT D	sprD			$FE $1D(254,29)
SPRITE	S SHIFT P	sP			$FE $07(254,7)
SPRSAV	SPR SHIFT S	sprS			$FE $16(254,22)
SQR	S SHIFT Q	sQ	Numerikus	$BA(186)	$BA(186)
SSHAPE	S SHIFT S	sS			$E4(228)
STASH	S SHIFT T	sT			$FE $1F(254,31)
STATUS	ST	st	Numerikus		
STEP	ST SHIFT E	stE		$A9(169)	$A9(169)
STOP	ST SHIFT O	stO		$90(144)	$90(144)
STR$	ST SHIFT R	stR	Sztring	$C4(196)	$C4(196)
SWAP	S SHIFT W	sW			$FE $23(254,35)

Parancs	Rövidítés	Képernyő	Függvénytípus	Token(ek) C-64 mód	Token(ek) C-128 mód
SYS	S SHIFT Y	sY		$9E(158)	$9E(158)
TAB(	T SHIFT A	tA	Speciális	$A3(163)	$A3(163)
TAN	nincs	tan	Numerikus	$C0(192)	$C0(192)
TEMPO	T SHIFT E	tE			$FE $05(254,5)
THEN	T SHIFT H	tH		$A7(167)	$A7(167)
TIME	TI	ti	Numerikus		
TIME$	TI$	ti$	Sztring		
TO	nincs	to		$A4(164)	$A4(164)
TRAP	T SHIFT R	tR			$D7(215)
TROFF	TRO SHIFT F	troF			$D9(217)
TRON	TR SHIFT O	trO			$D8(216)
UNTIL	U SHIFT N	uN			$FC(252)
USR	U SHIFT S	uS	Numerikus	$B7(183)	$B7(183)
VAL	V SHIFT A	vA	Numerikus	$C5(197)	$C5(197)
VERIFY	V SHIFT E	vE		$95(149)	$95(149)
VOL	V SHIFT O	vO			$DB(219)
WAIT	W SHIFT A	wA		$92(146)	$92(146)
WHILE	W SHIFT H	wH			$FD(253)
WIDTH	WI SHIFT D	wiD			$FE $1C(254,28)
WINDOW	W SHIFT I	wI			$FE $1A(254,26)
XOR	X SHIFT R	xR	Numerikus		$CE $08(206,8)

A C-64 üzemmód esetén az alábbi rövidítéseket is használhatjuk:

Parancs	Rövidítés	Képernyő
CONT	C SHIFT O	cO
END	E SHIFT N	eN
PEEK	P SHIFT E	pE
POKE	P SHIFT O	pO
READ	R SHIFT E	rE
SPC(	S SHIFT P	sP
STOP	S SHIFT T	sT

F.2 BASIC hibaüzenetek

A hibaüzenet mindig egy ? kiírásával kezdődik és az ERROR (hiba) szóval végződik. Ha 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 DISK (36)
Nem megfelelően megformázott lemezt használtunk.

BAD SUBSCRIPT (18)
A program egy tömb olyan elemére hivatkozott, amely túlmutat az egyáltalán lehetséges, vagy a DIM utasításban megadott határon.

BEND NOT FOUND (37)
Az interpreter egy már megnyitott BEGIN utasítászárójelhez nem találja a BEND-et. Valószínűleg hiányzik, estleg rosszul struktúráltuk a programot.

BREAK (30)
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 (26)
CONT paranccsal nem tudjuk folytatni a programot,mert 1/ a programot nem indítottuk el a RUN paranccsal; 2/ hiba után akarjuk tovább-indítani a programot; 3/ időközben átszerkesztettük a programot.

CAN'T RESUME (31)
A RESUME utasítást a rendszer nem tudja végrehajtani, mert nem történt hiba, és nem lépett be a hibarutinba.

DEVICE NOT PRESENT (5)
A megcímzett I/O eszköz nincs a rendszerben, vagy nem üzemképes (pl. nincs bekapcsolva).

DIRECT MODE ONLY (34)
Csak parancs módban használható utasítást kíséreltünk meg programban használni.

DIVISION BY ZERO (20)
Nullával osztottunk.

EXTRA IGNORED (figyelmeztetés)
Túl sok adatot gépeltünk be az INPUT utasítás után, s csak az első néhány került elfogadásra.

FILE DATA (24)
Egy megnyitott file-ból sztring típusú adat érkezett, a program viszont számadatot várt.

FILE NOT FOUND (4)
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 (3)
A CLOSE, CMD, PRINT#, INPUT# vagy GET# által használni kívánt file-t még nem nyitottuk meg.

FILE OPEN (2)
Egy olyan file megnyitására történt kísérlet, amelyet már egyszer megnyitottunk, de még nem zártunk le.

FILE READ (41)
A file-ba való íráskor vagy olvasáskor (fizikai) hiba történt. Feltehetően az adathordozó hibája.

FORMULA TOO COMPLEX (25)
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 (21)
Az utasítás parancs (direkt) módban nem használható.

ILLEGAL DEVICE NUMBER (9)
Valamelyik használni kívánt perifériának a hardver száma nem esik az 1-255 intervallumba.

ILLEGAL QUANTITY (14)
A szám, amelyet egy függvény vagy utasítás argumentumaként használtunk, kívül esik a megengedett értékhatárokon.

LINE NUMBER TOO LARGE (38)
A RENUNBER parancs végrehajtása során úgy kellene átszámozni a programot, hogy valamelyik sor száma nagyobb lenne, mint 64000.

LOAD (29)
A szalagról vagy lemezről való töltés az adathordozó hibája miatt megszakadt. Elsősorban szalagról való töltés esetén fordulhat elő.

LOOP NOT FOUND (32)
Az interpreter nem találja az éppen végrehajtás alatt álló DO utasításhoz tartozó LOOP utasítást. Az utasítás hiányzik a programból.

LOOP WITHOUT DO (33)
Az interpreter nem találja az éppen végrehajtás alatt álló LOOP utasításhoz tartozó DO utasítást. A program nem megfelelően strukturált, vagy nem is hajtottuk végre a DO utasítást, vagy már törlődött a rá vonatkozó hivatkozás a veremből.

MISSING FILE NAME (8)
Egy LOAD, OPEN, SAVE utasításban nem, vagy nem jól adtuk meg a file nevét.

NEXT WITHOUT FOR (10)
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).

NO GRAPHICS AREA (35)
A nagyfelbontású képernyőt kíséreljük meg használni, de még nem adtuk ki a megfelelő GRAPHIC utasítást.

NOT INPUT FILE (6)
Egy outputként kijelölt file-t INPUT# vagy GET# utasításban akartunk használni.

NOT OUTPUT FILE (7)
Egy inputként kijelölt file-ba akartunk PRINT# utasítással írni.

OUT OF DATA (13)
A READ utasítás nem tudott a DATA-sorokból adatot olvasni, mert már az összeset felhasználtuk.

OUT OF MEMORY (16)
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 (15)
A számítási műveletek eredménye nagyobb, mint a megengedett legnagyobb szám, amely 1.70141883E+38.

REDIM'D ARRAY (19)
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 (figyelmeztetés)
Az INPUT utasítás végrehajtása során számjegy helyett karakter adatot írtunk be. Újra írjuk be az adatokat, most már helyesen, és a program futása folytatódik.

RETURN WITHOUT GOSUB (12)
RETURN utasítást hajtottunk végre azt megelőző GOSUB hívás nélkül.

STRING TOO LONG (23)
Az eredményül kapott sztring több, mint 255 karaktert tartalmaz.

SYNTAX (11)
Az utasítás nem felismerhető a C-128 BASIC V2.0 vagy V7.0 számára: hiányzó zárójel, hibás kulcsszó stb.

TOO MANY FILES (1)
Egyszerre maximum 10 nyitott file lehet. Ennél többet kíséreltünk meg megnyitni.

TYPE MISMATCH (22)
Szám helyett sztringet, (vagy fordítva) használtunk.

UNDEF'D FUNCTION (27)
Felhasználó által definiált függvényre hivatkoztunk, de azt még nem definiáltuk a DEF FN utasítással.

UNDEF'D STATEMENT (17)
Nem létező sorra történt hivatkozás GOTO, GOSUB, RUN, IF, RESUME, COLLISION, TRAP, illetve ON utasításokban.

UNIMPLEMENTED COMMAND ERROR (40)
A QUIT és az OFF parancsok valamelyikét adtuk ki. Ezeket az utasításokat csak megfelelő bővítmény használata esetén lehet használni, s ez nincs a rendszerben.

UNRESOLVED REFERENCE (39)
A RENUMBER végrehajtása közben a rendszer érzékelte, hogy nem létező sorra történt hivatkozás GOTO, GOSUB, RUN, IF, RESUME, COLLISION, TRAP, illetve ON utasításokban.

VERIFY (28)
A szalagra vagy lemezre másolt program nem egyezik meg a memóriában levővel.

F.3 DOS hibaüzenetek

A 20-nál kisebb hibakódú üzeneteket figyelmen kívül lehet hagyni. Ezek közül csak a 0 és 1 kódszámnak van közvetlen jelentése.

0 OK
Minden rendben.

1 FILES SCRATCHED
Az 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 kísérelték 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 közben olyan nagy rekordszámot használtunk, amelyik a lemez betelését eredményezné.

60 WRITE FILE OPEN
Egy írásra megnyitott file-t, annak lezárása nélkül, olvasásra kíséreltük 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 Ø, akkor ilyen szabad blokk már nincs.

66 **ILLEGAL TRACK AND SECTOR**
A DOS végrehajtása közben egy nem létező sáv-, 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.

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 nem megfelelő verziójú DOS-szal formázott lemezre akarunk írni. Bekapcsolás után ugyanezt az üzenetet kapjuk.

74 **DRIVE NOT READY**
Nincs egyetlen lemez sem a lemezmeghajtóban.

F.4 Szabvány BASIC programok átalakítása

Ha nem C-128 BASIC-ben írt BASIC programokkal rendelkezünk, és ezeket C-128-on akarjuk futtatni, akkor előbb néhány apróbb változtatást kell elvégeznünk. Néhány tanácsot adunk, hogy ezt az átalakítást megkönnyítsük.

Sztring típusú tömbök

A C-128 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 egydimenziós (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-128 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-128-on így valósitjuk 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 és C értéke is 0 lesz. A C-128-on 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-128-on természetesen kettősponttal kell elválasztani.

MAT függvények

Elsősorban minigépeken futó BASIC interpreterek lehetővé teszik mátrix műveletek használatát. Ezeket C-128-on külön alprogramok megírásával lehet csak előállítani.

F.5 Képernyő kódok

Az alábbi táblázatban megtalálhatjuk a C-128 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 egyidejűleg 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 DIN karakterkészlet használata esetén nem csak a karakterek alakja, hanem a kijelezhető karakterek készlete is megváltozik. Külön táblázatban tüntetjük fel az eltéréseket.

A következő intervallumba eső kódszámú karakterek a két karakterkészletben azonosak:

27-64,91-93,96-104,106-121,123-127.

Képernyő kódok:

Kisbetűk/nagybetűk karakterkészlet

	0	1	2	3	4	5	6	7
0	@	P		0	─	P		┌
1	a	q	!	1	A	Q	▌	┴
2	b	r	"	2	B	R	▬	┬
3	c	s	#	3	C	S	▔	┤
4	d	t	$	4	D	T	_	├
5	e	u	%	5	E	U	▏	▌
6	f	v	&	6	F	V	▒	▐
7	g	w	'	7	G	W	▕	▔
8	h	x	(	8	H	X	▔	▀
9	i	y	)	9	I	Y	▒	▄
A	j	z	*	:	J	Z	▕	▗
B	k	[	+	;	K	+	▐	▪
C	l	£	,	<	L	▒	▪	▪
D	m	]	-	=	M	▕	▖	▖
E	n	↑	.	>	N	▚	▝	▪
F	o	←	/	?	O	▒	▂	▘

Nagybetűk/grafikus jelek karakterkészlet

	0	1	2	3	4	5	6	7
0	@	P		0	─	┐		┌
1	A	Q	!	1	♠	●	▌	┴
2	B	R	"	2	│	─	▬	┬
3	C	S	#	3	─	♥	▔	┤
4	D	T	$	4	┌	│	_	├
5	E	U	%	5	┐	╱	▕	▌
6	F	V	&	6	└	╳	▒	▐
7	G	W	'	7	│	○	▕	▔
8	H	X	(	8	┘	♣	▔	▀
9	I	Y	)	9	╮	│	◤	▄
A	J	Z	*	:	╰	◆	▕	▗
B	K	[	+	;	╯	┼	▐	▪
C	L	£	,	<	└	▒	▪	▪
D	M	]	-	=	╱	▕	▖	▖
E	N	↑	.	>	╱	π	▝	▪
F	O	←	/	?	┌	◥	▂	▘

DIN karakterkészlet eltérések

	0	1	2	3	4	5	6	7
0	§							@
1							¦	µ
2							└	à
3							┴	ù
4							┘	â
5							├	ê
6							─	î
7							┤	ô
8							┌	û
9							┬	Ö
A							┐	Σ
B						Ä	┼	ä
C			\			Ö	é	ö
D						Ü	£	ü
E						π	è	ß
F			←			─	´	^

F.6 ASCII vezérlő karakterek és szín kódok

Ez a függelék a PRINT CHR$(X) utasítással végrehajtható vezérlő karaktereket tartalmazza táblázatos formában, X lehetséges értékeire. Az alfanumerikus jelek ASCII kódjai a **Commodore információs kártya** 59-60. oldalain találhatók meg.

ASCII kódok (vezérlő karakterek):

ASCII kód	C-64 mód	C-128 mód
0 ($00)	-	-
1 ($01)	-	-
2 ($02)	-	aláhúzás bekapcsolása
3 ($03)	<STOP>	<STOP>
4 ($04)	-	-
5 ($05)	<Wht>	<Wht>
6 ($06)	-	-
7 ($07)	-	hangjelzés adása
8 ($08)	<C=-SHIFT> tiltása	-
9 ($09)	<C=-SHIFT> engedélyezése	tabulálás
10 ($0A)	-	soremelés
11 ($0B)	-	<C=-SHIFT> tiltása
12 ($0C)	-	<C=-SHIFT> engedélyezése
13 ($0D)	<RETURN>	<RETURN>
14 ($0E)	kisbetűk/grafikák	kisbetűk/grafikák
15 ($0F)	-	villogás bekapcsolása
16 ($10)	-	-
17 ($11)	<CRSR LE>	<CRSR LE>
18 ($12)	<RVS ON>	<RVS ON>
19 ($13)	<HOME>	<HOME>
20 ($14)	<DEL>	<DEL>
21 ($15)	-	-
22 ($16)	-	-
23 ($17)	-	-
24 ($18)	-	tabulátor pont ki/be
25 ($19)	-	-
26 ($1A)	-	-
27 ($1B)	-	<ESC>
28 ($1C)	<Red>	<Red>
29 ($1D)	<CRSR JOBBRA>	<CRSR JOBBRA>
30 ($1E)	<Grn>	<Grn>
31 ($1F)	<Blu>	<Blu>

ASCII kódok (folytatás: vezérlő karakterek):

ASCII kód	C-64 mód	C-128 mód
128 ($80)	–	–
129 ($81)	–	<Orng>
130 ($82)	–	aláhúzás kikapcsolása
131 ($83)	–	–
132 ($84)	–	<HELP>
133 ($85)	<F1>	<F1>
134 ($86)	<F3>	<F3>
135 ($87)	<F5>	<F5>
136 ($88)	<F7>	<F7>
137 ($89)	<F2>	<F2>
138 ($8A)	<F4>	<F4>
139 ($8B)	<F6>	<F6>
140 ($8C)	<F8>	<F8>
141 ($8D)	<SHIFT-RETURN>	<SHIFT-RETURN>
142 ($8E)	nagy betűk/grafikák	nagy betűk/grafikák
143 ($8F)	–	villogás kikapcsolása
144 ($90)	<Blk>	<Blk>
145 ($91)	<CRSR FEL>	<CRSR FEL>
146 ($92)	<RVS OFF>	<RVS OFF>
147 ($93)	<CLR>	<CLR>
148 ($94)	<INST>	<INST>
149 ($95)	<Brn>	<Brn>
150 ($96)	<LRed>	<LRed>
151 ($97)	<LGry>	<LGrey>
152 ($98)	<MGry>	<MGrey>
153 ($9A)	<LGrn>	<LGrn>
154 ($9B)	<LBlu>	<LBlu>
155 ($9C)	<DGry>	<DGry>
156 ($9D)	<Pur>	<Pur>
157 ($9E)	<CRSR BALRA>	<CURSOR BALRA>
158 ($9E)	<Yel>	<Yel>
159 ($9F)	<Cyn>	<Cyn>

Szín kódok

Kód	Szín	Billentyűzés		COLOR kód
0	fekete	<CTRL 1>	<Blk>	1
1	fehér	<CTRL 2>	<Wht>	2
2	piros	<CTRL 3>	<Red>	3
3	türkisz	<CTRL 4>	<Cyn>	4
4	bíbor	<CTRL 5>	<Pur>	5
5	zöld	<CTRL 6>	<Grn>	6
6	kék	<CTRL 7>	<Blu>	7
7	sárga	<CTRL 8>	<Yel>	8
8	narancs	<C= 1>	<Orng>	9
9	barna	<C= 2>	<Brn>	10
10	sárgászöld	<C= 3>	<L Red>	11
11	rózsaszín	<C= 4	<D Gry>	12
12	kékeszöld	<C= 5>	<M Gry>	13
13	világoskék	<C= 6>	<L Grn>	14
14	sötétkék	<C= 7>	<L Blu>	15
15	világoszöld	<C= 8>	<L Gry>	16

F.7 A billentyűzet újradefiniálása

A Commodore 128 C-128-as üzemmódjában a billentyűk ASCII kódjait öt különböző táblázat tartalmazza. Az öt táblázat közül az első a váltók használata nélküli megnyomásnak, a többi rendre a <SHIFT>, <C=>, <CTRL>, illetve <ALT> váltókkal egyidejű megnyomásnak felel meg. A táblázatok megadására a 3. lapon elhelyezett mutatók szolgálnak. Normál körülmények között ezek a ROM-ban levő táblázatokra mutatnak. Az <ASCII/DIN> megnyomása nem csak a karakter generátor ROM-ot cseréli, de az interpreter átírja ezeket a mutatókat. Ennek hatására pl. helyet cserél az Y és a Z! Az egyes mutatók helye és alapértéke a következő:

Váltó	Mutató		Alapérték	
	HEX	DEC	ASCII	DIN
nincs	33E	830	FA80	FD29
SHIFT	340	832	FAD9	FD82
C=	342	834	FB32	FDDB
CTRL	344	836	FB8B	FB8B
ALT	346	838	FA80	FD29

Egy-egy táblázat 89 bejegyzésből áll, s a bejegyzések sorban a következő billentyűknek felelnek meg:

Sorszám	Billentyű	Sorszám	Billentyű
1	DEL	46	:
2	RETURN	47	@
3	CRSR JOBBRA	48	,
4	F7	49	FONT
5	F1	50	*
6	F3	51	;
7	F5	52	HOME
8	CRSR LE	53	SHIFT jobb
9	3	54	=
10	w	55	↑
11	a	56	/
12	4	57	1
13	z	58	-
14	s	59	CTRL
15	e	60	2
16	SHIFT bal	61	SZÓKÖZ
17	5	62	C=
18	r	63	q
19	d	64	STOP
20	6	65	HELP
21	c	66	8 (num)
22	f	67	5 (num)
23	t	68	TAB
24	x	69	2 (num)
25	7	70	4 (num)
26	y	71	7 (num)
27	g	72	1 (num)
28	8	73	ESC
29	b	74	+ (num)
30	h	75	- (num)
31	u	76	LINEFEED
32	v	77	ENTER
33	g	78	6 (num)
34	i	79	9 (num)
35	j	80	3 (num)
36	0	81	ALT
37	m	82	0 (num)
38	k	83	. (num)
39	o	84	FEL
40	n	85	LE
41	+	86	BALRA
42	p	87	JOBBRA
43	l	88	NO SCR
44	-	89	fenntartott
45	.		

A SHIFT BAL, SHIFT JOBB, CTRL, C=, ALT, NO SCR és a 'fenntartott'-tal jelzett hét értéknek valamennyi táblázatban rendre az alábbiaknak kell lennie: 1,1,4,2,8,255, illetve 255. Ez biztosítja a váltók korrekt használatát. A táblázat többi értékét magunk is megváltoztathatjuk. Ehhez persze a táblázat kezdetére mutató értékeket úgy kell beállítani, hogy a RAM-ba mutasson, s úgy kell megadni saját táblázatunkat. Megjegyezzük, hogy ezeket a kódtáblázatokat mindig a 15-ös szeleten várja a rendszer!

F.8 A C-64-gyel való kompatibilitás

Természetes kérdés: vajon milyen mértékben 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 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 épitenek.

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 ($FFFC) utasítás kerül végrehajtásra. Amikor a C-128-as gépen áttérünk a C-64-es üzemmódba, a processzor már végrehajtott egy ilyen ciklust, s az üzemmód váltás szoftver úton történik meg. Ezt kihasználva módosítani lehet a C-64-es üzemmód reset ciklusát, úgy, hogy az ne egy JMP ($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-tól megfelelő 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 utasitások 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 poke 0,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 kapjunk. **(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   ; 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         ; alprogram vége
0358  a2 03     ldx #$03    ; három billentyűzetsor tesztelése
035a  8a        txa         ;  melyikben van lenyomott billentyű
035b  48        pha         ;  az érték veremben tárolása
035c  bd 9c 03  lda $039c,x ; az x.-ik 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 le-
036c  f0 0b    beq $0379 ; nyomás, 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 kishiftellése és
0373  90 10    bcc $0385 ;  ellenőrzése, hogy 0-e
0375  e8       inx       ; nem: bitszám növelése és a
0376  88       dey       ;  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 $035a ; áttéres a következő oszlopra
037e  a9 ff    lda #$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       ; megszakítások engedélyezése
039c  60       rts       ; visszatérés az 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 printko$(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 6.2 paragrafusban adott POKE és PEEK rutinok segítségével 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 operativ tár és a videomemória közti adatcserét végzi. Ehhez adunk egy BASIC töltő progrmot is, amelyik egyben rögtön be is mutatja ennek használatát.

Az alábbi gépi kódú programrész három rutint tartalmaz, amelyek a $C000, $C003, illetve $C006 belépési pontokon keresztül érhetők el. Meghívásuk előtt három mutatót kell beállítani. Az első a ($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 proceszor 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 $c02d ; vdcmem <-> mem
c009  00 00     .byte 0,0 ; mutató az utolsó másolandó byt-ra.
                          ; a processzor memóriájának átmásolása
                          ;  a VDC memóriájába
                          ;  ($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 #$00  ; az akkumulátorba az elmentett byte
c042  68        pla       ;  majd beírása az
c043  91 fc     sta ($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
```

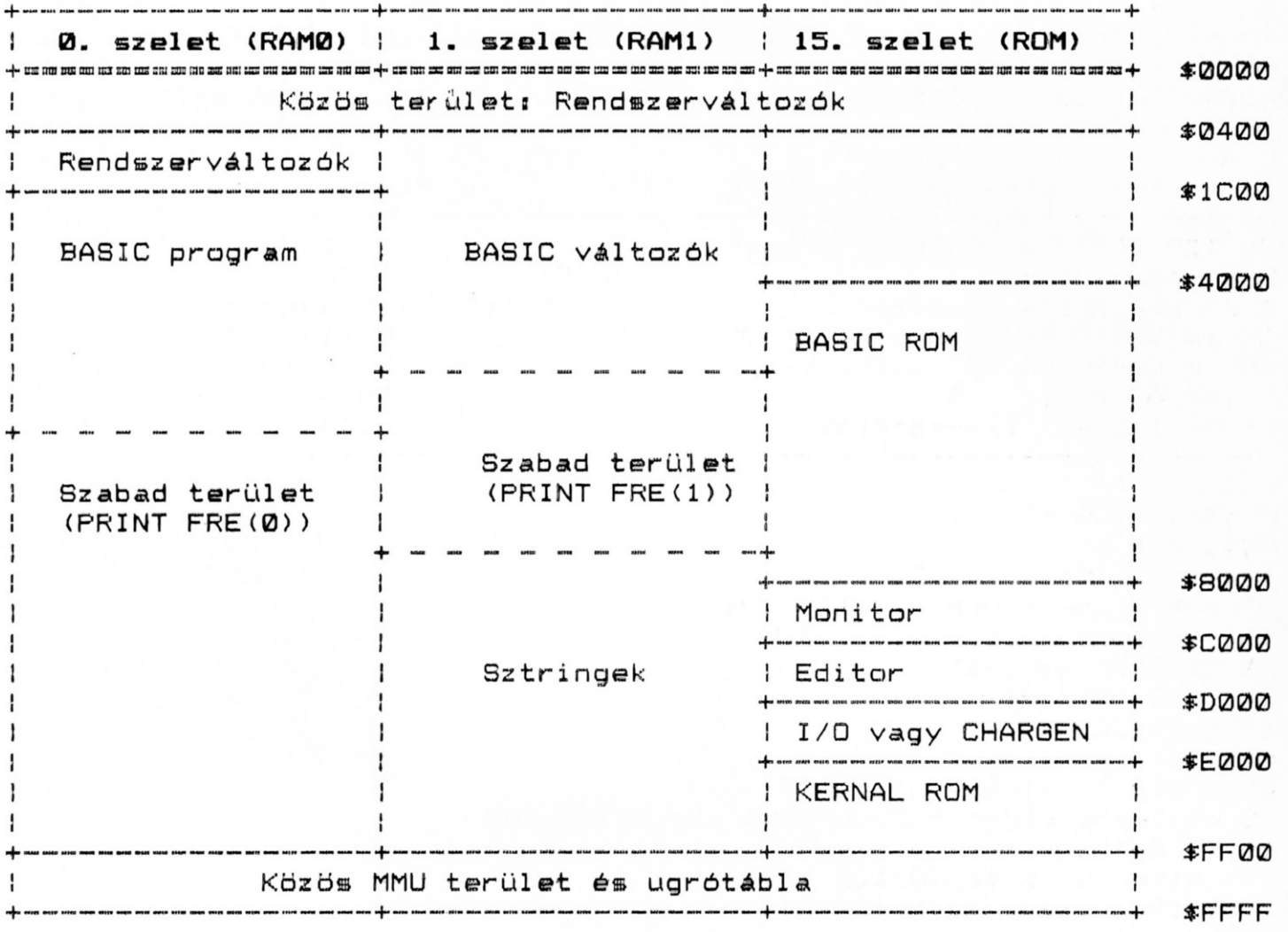
```
c060  cd 09 c0 cmp $c009 ;  is össze kell hasonlítani
c063  60       rts       ; visszatérés az alprogramból
                           ; PEEK rutin
                         ; a VDC memóriájának olvasása,
                         ; az eredmény az A regiszterben.
                         ; X a cím 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, a felső az Y-ban.
c07f  48       pha       ; A byte elmentése a verembe.
c080  8a       txa       ; X átírása A-ba, majd
c081  48       pha       ;  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-kban 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 360. sorban kezdődő alprogram a mutatók értékeit állítja be. 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 vizuálisan tudjuk ellenőrizni is! 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 110
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"
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 geoi 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.9 Memória térkép



F.10 Az RGBI monitor csatlakozója

A C-128 csatlakozói - két kivételével - megegyeznek a Commodore 64 számítógép csatlakozóival. Az egyik kivétel a transzformátor csatlakozója, ami a C-128 esetén négyszögletes.

Másik kivétel az 80 oszlopos monitor csatlakozója, ami a C-64-en egyáltalán nincs. Alább közöljük a csatlakozó lábkiosztását. Megjegyezzük, hogy a mért jelszintek az itt közölt gyári értéknél általában magasabbak!

Láb	Jelölés	Leírás
1	GND	Föld
2	GND	Föld
3	Red	TTL jel, 0-5V, az RGBI piros része (R)
4	Green	TTL jel, 0-5V, az RGBI zöld része (G)
5	Blue	TTL jel, 0-5V, az RGBI kék része (B)
6	Int	TTL jel, 0-5V, az RGBI intenzitása (T)
7	Monochrome	analóg jel, 1V, 15625 Hz
8	Hsync	TTL jel, 0-5V, vizszintes szinkron
9	Vsync	TTL jel, 0-5V, függőleges szinkron

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 találhatunk bővebb információt.

a könyv címe	jelölése:
C= 128 BASIC és felhaszn. kézikönyv	C-128
Z-80 Assembler	Z-80
CP/M operációs rendszer	CP/M
C= 64 BASIC és felhaszn. kézikönyv I-II.	C-64
C= 16 BASIC és felhaszn. kézikönyv	C-16
C= 64 Assembly	CASM
C= 64 I/O és file-kezelés	CI/O

Kiadó: LSI ATSZ
Felelős kiadó: Dr. Kovács Magda
Témafelelős: Székely László
ISBN: 963 592 593 X
Engedélyszám: 51258
Készült: OMIKK házinyomdájában
1011 Budapest, Gyorskocsi u. 5–7.
Felelős vezető: Tóth Károly

Ára: 279,– Ft