DR. ÚRY LÁSZLÓ

COMMODORE 64
COMMODORE 128/64 ÜZEMMÓD
BASIC FELHASZNÁLÓI KÉZIKÖNYV

I. KÖTET

LSI ALKALMAZÁSTECHNIKAI
TANÁCSADÓ SZOLGÁLAT

dr. Úry László

COMMODORE 64

COMMODORE 64C
COMMODORE 128 / C-64-es üzemmód

BASIC felhasználói kézikönyv

Bővített, átdolgozott kiadás

LSI Alkalmazástechnikai Tanácsadó Szolgálat

Budapest, 1987

Programozói segédlet

Lektorálta: Donát János
Vadnai Szabolcs

ISBN I 963592 668 X
II 963592 696 0
ISBN össz.: 963592 697 9

COMMODORE 64

COMMODORE 64C
COMMODORE 128 / C-64-es üzemmód

I. kötet

****

COMMODORE 64C

A Commodore cég 1986-ban a Commodore 64 személyi számítógép - technikailag továbbfejlesztett - változatát dobta piacra, s ezzel együtt megszüntette a Commodore 64 gyártását. Az új gép teljes egészében kompatibilis a régivel, a benne szereplő chipek azonban új technológiával készültek. Ezzel együtt a 1541-es lemezegység helyett megjelent a 1541C lemezegység.

A könyvünkben mondottak változtatás nélkül érvényesek a C-64C és a 1541C berendezésekre is. Tartalmazza az ezek használatához szükséges *valamennyi* információt.

A GEOS operációs rendszer használata az ***1001/2 játék Commodore 64/128*** című kiadványban található meg.

****

****

COMMODORE 128 / C-64-es üzemmód

Az eredetileg Commodore 64-re készült kézikönyv jelenlegi kiadása az összes olyan ismeretet magában foglalja, amelyik a Commodore 128-as számítógép C-64-es üzemmódjának a használatához szükséges. Egytelen kivétel: a számítógép üzembeállítását a Commodore 128 Basic felhasználói kézikönyvben leírtak szerint kell elvégezni.

A 1541-es lemezegységről mondottak teljes egészében használhatók a 1571/70-es lemezegységek esetén is.

A Commodore 128 nagy mértékben könnyíti a C-64-es üzemmód használatát. Lehetőség van pl. gépi kódú programok C-128-as üzemmódban való megírására és tesztelésére. Az így elkészített program azután C-64-es üzemmódban is használható.

A Commodore 128-as gép C-64-es üzemmódja néhány esetben eltér (általában többet tud) a Commodore 64-től. Ezeket a ***Commodore 128 BASIC és felhasználói kézikönyv*** című kiadvány I. kötetének a 8. függelékében ismertettük.

****

Előszó a bővített, átdolgozott kiadáshoz

Mi tagadás, szeretem a mikroszámítógépeket. Sokfajta számítástechnikai eszköz használatát tanítottam, kezdve az abakusztól, a legkülönfélébb mikro-, és minigépeken át egészen az V. generációs gépek logikai alapjaiig. Oktatási segédletek is szép számmal terhelik a lelkiismeretemet, köztük olyan kalkulátoré is, aminek ma már a nevére sem emlékezik senki. Amikor az LSI ATSZ-től megkaptam a megbizást egy Commodore kézikönyv megirására, magam sem gondoltam, hogy a hazánkban hamarosan legelterjedtebb mikroszámitógépről fogok irni.

A könyv megjelenése óta kettős életet élek. Mert igaz ugyan, hogy szeretem a Commodore 64 háztartási személyi számítógépet; de arra a nem is ritkán feltett kérdésre, hogy professzionális személyi számítógép-e a C-64, mindig határozott nemmel felelek. A Commodore 64 semmiképpen sem alkalmas - mondjuk - egy atomerőmű vezérlésére (a villanyvonatomnak azonban tökéletesen megfelel...)

Ennek ellenére a C-64-et lehet profi módon használni. Ennek csak egyik eleme, hogy tisztában legyünk a gép adottságaival, és jó programokat tudjunk irni. Ennél talán lényegesebb az, hogy az adott feladat (számitástechnikai) megvalósitásához ki tudjuk választani az adekvát eszközrendszert. Ha ez C-64, akkor legyen C-64! Ma már - főleg a mikrogépek kategóriájában - a magas ár, a beszerzés nehézségei nem legyőzhetetlen akadályok. A kézikönyv második, átdolgozott kiadása a C-64 profi használatát igyekszik támogatni. A géppel csak most ismerkedők számára az LSI ATSZ gondozásában hamarosan egy külön tankönyv jelenik meg, amelyik a C-64 alapjaival ismerteti meg az Olvasót.

Ezzel összhangban kimaradtak az oktató (egyesek szerint kioktató) részek, igy például a teljes 10. fejezet. Ugyanakkor lényegesen bővült a lemezegységekről, illetve a nyomtatókról szóló rész.

Sokan javasolták, hogy a könyvet bővítsem ki további felhasználói programok leirásával. Ezt sem a könyv jellege, sem terjedelme nem teszi lehetővé. Itt elsősorban magáról a gépről van szó, nem pedig a rajta futtatható szoftverekről. Különben is, ezen programtermékek leirásai más helyeken már megjelentek.

Végül kedves kötelességem mindazoknak köszönetet mondani, akik ötleteikkel, javaslataikkal, s nem utolsó sorban munkájukkal ennek a kiadványnak a megjelenését elősegítették. Legnagyobb köszönet az átdolgozott kiadás lektorát, Donát Jánost illeti, aki alapos lektori munkájával, javaslataival sokat segitett a szöveg végleges megfogalmazásában. Köszönet illeti feleségemet, Úry Ágnest, aki nemcsak az első kiadás programjait ellenőrizte, hanem szó nélkül tűrte, hogy – a könyv írásának ürügyén – otthonunkból kisebb számítóközpontot alakítsak ki.

Köszönet illeti Bakonyi Matildot és Bálint Zsuzsát, akik az első kiadás kéziratát gépelték, s Szegedi Gyöngyit, aki a könyv szövegszerkesztővel történő rögzítését végezte. Hálás vagyok az LSI ATSZ valamennyi dolgozójának, különösen Sziklai Klárának, akik buzdításaikkal, s nem utolsó sorban a határidőkre való folytonos figyelmeztetésükkel nagy mértékben segítették a könyv megjelenését.

Végül speciális köszönet illeti az AS-119/83 leltári számú C-64 mikroszámitógépet, amelyik egy ilyen könyv megjelenésével járó minden megpróbáltatást elviselt.

Budapest, 1985. május 14-én.

dr. Úry László

(dr. Úry László)

Utószó a második és a harmadik változatlan utánnyomáshoz

A Commodore 64 BASIC Felhasználói Kézikönyv első kiadása néhány hét alatt elfogyott. Ez a siker - gondolom - nem elsősorban a könyv erényeinek, hanem a C-64 mikroszámítógép iránti érdeklődésnek köszönhető. Hazánkban jelenleg közel háromezer C-64 típusú gép található, és számuk - egyes szakértők becslése szerint - az év végére eléri a négyezret. Ilyen kereslet mellett az első kiadás ötezres példányszáma valóban nem túl sok.

Számomra úgy tűnik, hogy a C-64-hez nyújtott - hazai (!) - hardver és szoftver szolgáltatások színvonala egyedülállóan magas a többi mikroszámítógéphez képest. A szoftver és hardver fejlesztésében a legnagyobb rendszerházaktól az "egyszemélyes" GMK-ig szinte mindenki résztvesz.

A KSH Számítástechnika-alkalmazási Főosztálya, illetve az LSI Alkalmazástechnikai Tanácsadó Szolgálat által az 1984. évi tavaszi BNV-re megjelentetett mikroszámítógépes katalógusok együtt közel 200 szoftver, illetve hardver terméket kínáltak a C-64-re. (Az egyik az SKV, a másik az Akadémia Kiadó könyvesboltjában kapható.) A termékek skálája igen széles. A C-64 összekapcsolása ESzR, MSzR, IBM és HwB típusú gépekkel (általában az RS-232-es csatorna segítségével) megtörtént. A C-64-hez a legkülönfélébb terminál emulátorokat kínálják. A perifériális eszközök kínálata is bőséges a magyar ékezetes nyomtatótól a szabványos numerikus billentyűzetig. Az alapszoftver területén szövegszerkesztőket, fordítóprogramokat, file-kezelő rendszereket, egyéb programnyelveket (pl. FORTH) vásárolhatunk (borsos áron persze). A SZÁMALK OSAK - más néven ugyan, de - forgalmazza a könyvben is szereplő HELP+-t és magyarosított alapszavakkal a SIMONS' BASIC-et. Legbőségesebb a választék természetesen az alkalmazói szoftverek területén. Ezek színvonala igen eltérő, ezért megvásárlásuk előtt célszerű alaposan kipróbálni őket.

Végül az ut(ols)ó szó jogán néhány további kiegészítést teszünk, s felsoroljuk az első kiadásban talált értelemzavaró hibákat.

1. fejezet

Bevezetés

1.1 A kézikönyv feladata

A könyv, amit az Olvasó a kezében tart, kettős céllal készült. Alapvető feladata, hogy programozói kézikönyvként a C-64 számítógéppel végzett munkát segítse; a felmerülő problémákra, vagy legalább is a legtöbbjükre választ adjon. Másik feladata, hogy a C-64-gyel még csak most ismerkedők számára segítséget nyújtson a gép 'birtokba vételéhez'. A két cél nem könnyen egyeztethető össze. Egy mikrogép programozásának elsajátítása a ROM-ban tárolt program (jelen esetben a BASIC) használatától halad a gépi kódú (vagy ami majdnem ugyanaz: az assembly) programozás felé. A gyakorlott programozó programjai általában tartalmaznak gépi kódú betéteket. A géppel még csak ismerkedők viszont először a gép alapjaival szeretnének tisztába jönni.

A könyv a C-64 programozásához szükséges **valamennyi ismeretet** tartalmazza. Nem tankönyv azonban, még akkor sem, ha sok és főleg egyszerű BASIC, illetve gépi kódú programrészt iktattunk a szövegbe. Ezek részletes áttanulmányozása, kipróbálása sokat segít az utasítások jobb megértésében. A C-64-gyel még csak most ismerkedőket segíti a könyvhöz külön megvásárolható **oktató lemez**, ami számos mintapéldát tartalmaz.

Végül - a fent említett két célon kívül - volt egy harmadik, nehezen kifejezhető szempontunk is a könyv megírásakor. Ez a szempont szorosan kapcsolódik a BASIC programozási nyelv problémáihoz. A BASIC - túl azon, hogy igen hamar megtanulható - nem a világ legtökéletesebb programozási nyelve. Alapvető hibája, hogy nem strukturált, s ezért - a priori - nem támogatja a strukturált programozást. Másik probléma a BASIC mikrogépeken való megvalósításával kapcsolatos, amelyek csak interpretert tartalmaznak, s ez meglehetősen lassítja a programok futását. Tetézi mindezt az, hogy a C-64 BASIC a gép egy sor lehetőségét közvetlenül nem támogatja.

Az első probléma módszertani jellegénél fogva nem tartozik szorosan a témánkhoz. A második problémával kapcsolatban az az álláspontunk, hogy a BASIC programok gépi kódú rutinokkal jól használhatók bonyolult és összetett feladatok megoldására is.

Néhány szót a könyvben használt jelölésekről. A Commodore irodalom a hexadecimális (16 alapú) számokat a szám elején levő $ jellel vezeti be. A $1C például decimálisan 28.

Abban az esetben, ha egy számot kerek zárójelek közé rakunk, azzal azt akarjuk kifejezni, hogy a számnak megfelelő memória egy két byte-os mutató alsó byte-ját tartalmazza. Például a (*7A) mutató az a két byte-os egész szám, melynek alsó byte-ja a *7A, felső byte-ja pedig a $7B címen található. (Decimális 122, 123.) A mutató értéke tehát:

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

Gyakran van szükségünk az egyes parancs- vagy programsorok pontos beírásának az ismeretére. Ilyen esetekben a vezérlő billentyűk nevét **csúcsos zárójelek közé** írjuk. Például a PRINT "<CLR>HELLO" parancsban a <CLR> **egyetlen** billentyű lenyomását jelenti. A képernyőn ennek hatására az idézőjelben egy inverz grafikus jel jelenik meg.

A kurzort mozgató billentyűk esetében **a nyilak helyett** a mozgás irányát jelöljük: <CRSR FEL>, <CRSR LE>, <CRSR JOBBRA> és <CRSR BALRA>.

Egyes esetekben két billentyűt kell lenyomni egyszerre. Ilyenkor a csúcsos zárójelek közt mind a két billentyű elnevezése szerepel kötőjellel összekötve. Ilyenkor az elsőnek feltüntetett billentyű **lenyomva tartása közben** kell a második billentyűt lenyomni. Például <CTRL-RVS ON>.

Vannak olyan esetek, amikor a grafikus jelek lényegesek, ilyenkor magát a programlistát közöljük.

1.2 Hogyan használjuk a könyvet?

A könyv fejezetei többé-kevésbé függetlenek egymástól. Kezdőknek először az első négy fejezet olvasását ajánljuk, ezek azok, amelyek elsősorban a BASIC-kel foglalkoznak. További fejezetek és paragrafusok vannak, amelyek különösebb előismeretek nélkül is használhatók, ilyenek elsősorban a perifériás egységekkel foglalkozó részek. A C-64 további lehetőségeinek kihasználásához a gépi kódú programozás ismerete szükséges. A 9. fejezet áttanulmányozása nagymértékben megkönnyíti a 4.-8. fejezetek gépi kódú utalásainak a megértését.

A könyv olvasásához a programozásban való különösebb jártasság nem szükséges. Az alapvető fogalmakat a megfelelő fejezetekben összefoglaljuk. Ezeket az ismereteket egy kezdő már néhány hét alatt minden nehézség nélkül elsajátíthatja. Az I/O-val kapcsolatos részek mélyebb ismereteket igényelnek. A gépi kódú programozásról szóló fejezet lehetővé teszi, hogy ezt a témával most először találkozók is megérthessék.

1.3 A C-64 mikroszámítógép üzembeállítása

A C-64 mikroszámítógép hátoldalán, oldalán csatlakozókat találunk. (Lásd a 13. oldalt!). A helyes működés alapfeltétele a kiegészítő berendezések megfelelő csatlakoztatása. A csatlakozók úgy vannak ugyan kialakítva, hogy mindegyik csak saját ellendarabjával csatlakoztatható, de azért röviden ismertetjük a C-64 üzembeállításának legfontosabb lépéseit.

1. A C-64 központi egysége a billentyűzettel egybeépült. A C-64 **transzformátora** azonban külön egység, ennek egyik csatlakozója a hálózati villásdugó. A másikat nyomjuk be a C-64 jobb oldalán található hálózati csatlakozóba. A transzformátort magát úgy helyezzük el, hogy mind az - esetleg használt - lemezegység(ek)-től, mind a televíziótól a lehető legmesszebb legyen.

2. A C-64 számítógép központi egységéhez kapott video kábel segítségével kössük össze a C-64 TV kimenetét és a televízió antenna csatlakozóját. Régebbi televíziós készülékek esetén antenna adapterre is szükség lehet. Ezt külön kell beszereznünk. Akiknek a televíziójún monitor bemenet is van, azok ezt közvetlenül is összeköthetik a C-64 monitor csatlakozójával. Az ehhez szükséges kábelt is külön kell beszerezni.

3. A transzformátor villásdugóját csatlakoztassuk a 220 V, 50 Hz-es hálózatba.

4. Helyezzük áram alá a televíziós készüléket, majd a C-64 számítógépet is.

5. Ezután kerülhet sor a televíziós készülék hangolására. Célszerű valamelyik VHF csatornát egyszer s mindenkorra a C-64 frekvenciájára hangolni. Ha ezen a csatornán egy közeli TV adó nagy erővel ad, akkor a C-64 hátoldalán levő kis csavar segítségével módosíthatjuk a csatornát, amelyiken a C-64 'ad'. Erre általában nincs szükség. A televíziós készüléket úgy kell beállítani, hogy tisztán jelenjen meg az alábbi üzenet:

```
   **** COMMODORE 64 BASIC V2 ****

64K RAM SYSTEM  38911 BASIC BYTES FREE
```

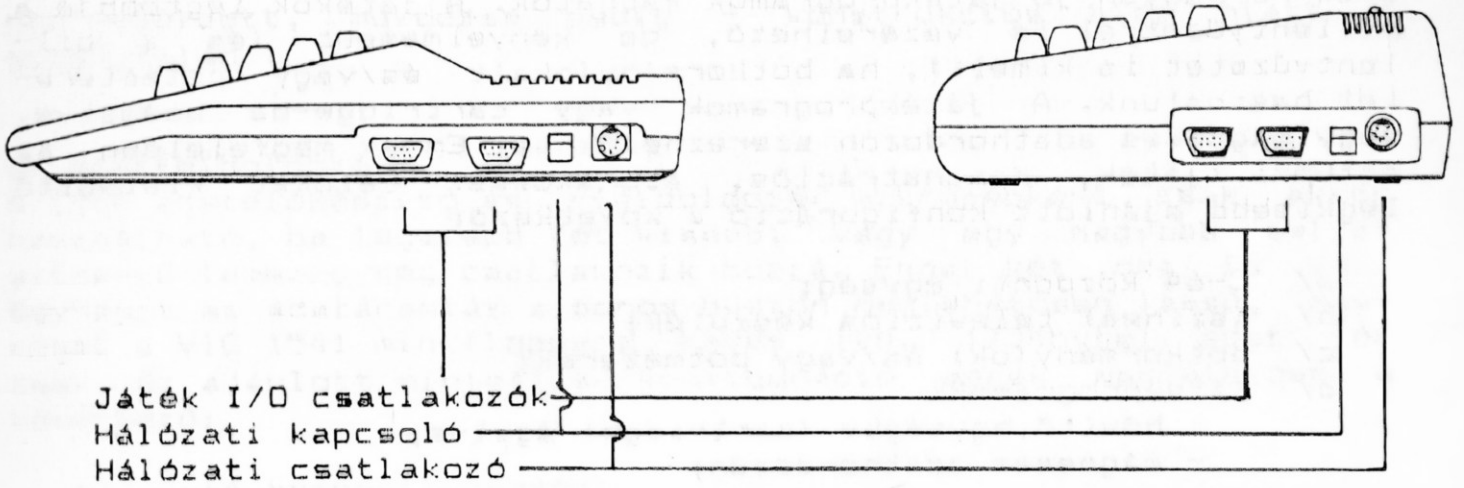
Színes készüléken a felirat világoskék keretben, sötétkék alapon, világoskék betükkel jelenik meg.

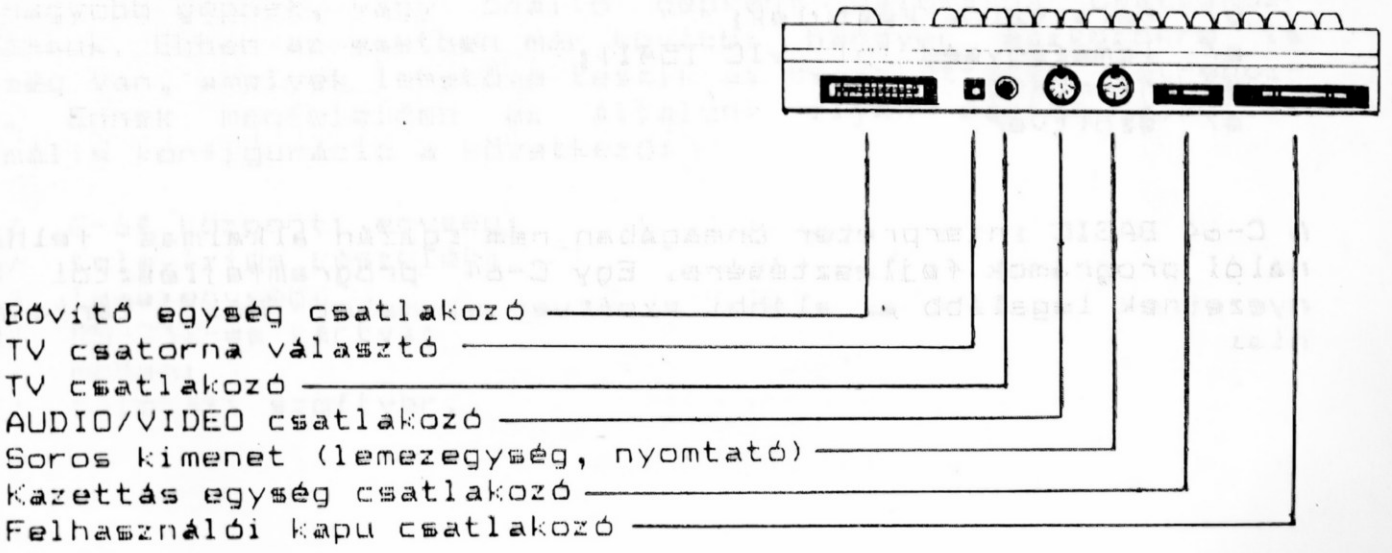
6. Amennyiben további kiegészítő berendezéseket is csatlakoztatunk a C-64-hez, azt a C-64 bekapcsolása előtt kell elvégezni. Ebben az esetben **először** a kiegészítő berendezéseket helyezzük áram alá, s csak azt követően a C-64-et.

7. Ha a C-64 bővítőjének vagy felhasználói kapujának csatlakozójába kártyát illesztünk, azt a használt berendezések áram alá helyezése előtt kell megtennünk. Vigyázzunk, hogy a kártyát megfelelő oldalával felfelé dugjuk be a gépbe!

8. Egyes esetekben - például ha C-64-et telefon vonalon keresztül más számítógéphez kötünk - célszerű szakember segítségét is igénybe venni.

A Commodore 64 számítógép csatlakozói





1.4 A kiegészítő berendezések kiválasztása

A C-64 számítógépet a tervezői több-célú, univerzális mikroszámítógépnek szánták. Ennek megfelelően - kiépítettségtől függően - más és más feladatok megoldására alkalmas

1/ **Otthoni felhasználás** (home computer)
Elsősorban kiváló tulajdonságú video chipjének köszönhetően a C-64-re nagyon jó játékprogramok kaphatók. A játékok legtöbbje a billentyűzetről is vezérelhető, de kényelmesebb (és a billentyűzetet is kíméli), ha botkormány(oka)t és/vagy potmétereket használunk. A játékprogramok vagy cartridge-ba beégetve, vagy mágneses adathordozón szerezhetők be. Ennek megfelelően az otthoni (játék, demonstrációs, szórakozás) célokat kielégítő legkisebb ajánlott konfiguráció a következő:

a/ C-64 központi egység;
b/ (színes) televíziós készülék;
c/ botkormány(ok) és/vagy potméterek;
d/ játékprogramok:
- bővítő egységbe (cartridge) égetve;
- mágneses adathordozón;

e/ emellett egy kazettás vagy lemezegység beszerzése is szinte elkerülhetetlen.

2/ **Programfejlesztés**
Felhasználói programok készítéséhez már komolyabb hardver és szoftver feltételekre van szükség. A javasolt legkisebb konfiguráció, amelyik ezt lehetővé teszi a következő:

a/ C-64 központi egység;
b/ televíziós készülék;
c/ lemezegység (pl. VIC 1541);
d/ nyomtató;
e/ szoftver.

A C-64 BASIC interpreter önmagában nem igazán alkalmas felhasználói programok fejlesztésére. Egy C-64 programfejlesztői környezetnek legalább az alábbi szoftver elemeket kell tartalmaznia:

i/ gépi kódú monitor;
ii/ assembler;
iii/ a BASIC valamely kiterjesztése, vagy más programozási nyelv;
iv/ nagy adattömböket kezelő program esetén egy DOS (lemezegység) monitor;
v/ AMT feladatok megoldásához a grafikus lehetőségeket is támogató programok.

Ezek a szoftver elemek általában beszerezhetők. Külön kiemeljük a **CP/M kártyát**, amelyik lehetővé teszi **Z-80 bázisú** programok felhasználását. (A cartridge javítása azonban hazai környezetben nem megoldott, működése pedig - konstrukciós hiba miatt - bizonytalan.)

3/ Adatfeldolgozás

A C-64 adatelőkészítő és -feldolgozó mikrogépként csak akkor használható, ha legalább két kisebb, vagy egy nagyobb teljesítményű lemezegység csatlakozik hozzá. Ennek két oka is van. Egyrészt az adatáramlás a soros buszon meglehetősen lassú, másrészt a VIC 1541 minifloppyra kevés (kb. 170Kbyte) adat fér csak. Az ajánlott minimális konfiguráció ennek megfelelően a következő:

a/ C-64 központi egység;
b/ televíziós készülék;
c/ nagyobb teljesítményű lemezegység(ek):
két VIC 1541-es egység vagy
egy CBM 8250-es duál lemezegység;
d/ nyomtató(k);
e/ felhasználói szoftver.

4/ Intelligens terminál

A C-64 kiképzése lehetővé teszi, hogy intelligens terminálként egy nagyobb géphez, vagy önálló gépként hálózatba csatlakoztathassuk. Ebben az esetben már további hardver eszközökre is szükség van, amelyek lehetővé teszik az összeköttetés létrehozását. Ennek megfelelően az általunk ilyen esetre ajánlott minimális konfiguráció a következő:

a/ C-64 központi egység;
b/ televíziós készülék;
c/ lemezegység;
d/ RS-232-es kártya;
e/ modem;
f/ hálózati szoftver.

A modem megválasztása a kialakítandó kapcsolat típusától függ, és szakember bevonását igényli.

A C-64 BASIC interpreter önállóan elvégzi az RS-232-es csatorna kezelését. Ez azonban nem mindig elegendő. Lokális hálózatok kialakításához például nem kell az RS-232-es csatlakozás, ugyanakkor további szoftver/hardver igénye van. Másrészt az adatforgalom lebonyolításához, az adatkonverzióhoz mindenképp további szoftverre van szükség.

2. fejezet

A C-64 képernyő szerkesztője

Bevezetés

Ez a fejezet a C-64 személyi számítógép legegyszerűbb felhasználási lehetőségeit, a számítógéppel való kommunikációt ismerteti. A paragrafus olvasása közben célszerű a példákat azonnal ki is próbálni, ezért a következő konfiguráció használatát javasoljuk:

a/ C-64 központi egység,
b/ televízió.

Párbeszédes üzemmód

A C-64 számítógép bekapcsolása (lásd 1.3-at) után bejelentkezik, majd kiírja a 'READY.' üzenetet. A 'READY.' alatt egy villogó téglalap, az úgynevezett **kurzor** látható. A 'READY.' (KÉSZ.) üzenet azt jelenti, hogy a számítógép befejezte tevékenységét és további parancsra vár. (A bekapcsolás után ez a tevékenység a **számítógép inicializálása** volt.)

A kurzor azt a helyet mutatja, ahová a következő, általunk begépelt karakter kerül. Ha lenyomjuk például a <?> billentyűt, akkor a kurzor helyén megjelenik egy kérdőjel, és a kurzor egy hellyel jobbra lép. Ha már nincs hely a sorban, akkor a kurzor a következő sor elejére ugrik. Az utasítást - bármi legyen is az - karakterenként kell begépelnünk (nem úgy, mint például a ZX-81 esetén), rövidítésük azonban általában megengedett. Egy speciális billentyű, a <RETURN> szolgál annak jelzésére, hogy **befejeztük** a parancs gépelését. Ezután a számítógép végrehajtja a parancsot, majd a következő parancs begépelésére vár.

Ha például a ? 2+2 begépelése után megnyomjuk a <RETURN> billentyűt, akkor válaszként a következőt kapjuk:

```
4

READY.
■       villog!
```

A legegyszerűbb esetben az utasítás végrehajtása után a 'READY.' üzeneten kívül semmi egyéb üzenetet nem kapunk. Ez azt jelenti, hogy parancsunkat a gép hibátlanul végre tudta hajtani. Gépeljük azonban be a következőt: ? 2/0 <RETURN>. A képernyőn a következő üzenet jelenik meg:

```
?DIVISION BY ZERO ERROR
READY.
■       villog!
```

A 'READY.' üzenet kiírása előtt egy másik - úgynevezett - **hibaüzenetet** kaptunk, amelyik jelzi, hogy milyen típusú probléma miatt nem volt képes a számítógép az utasítást teljes egészében végrehajtani. Jelen esetben ez pusztán annyit jelent, hogy 0-val kíséreltünk meg osztani. A számítógép hibaüzeneteit az egyéb üzeneteitől az különbözteti meg, hogy kérdőjellel (?) kezdődnek.

A C-64 lehetőséget biztosít arra, hogy a képernyő tetszőleges helyén látható információt parancsként használjuk fel. Így például a fenti, hibás ? 2/0 sort a következőképpen javíthatjuk ki. A <CRSR FEL> billentyűt addig tartsuk lenyomva, amíg a kurzor a fenti hibás sorba nem kerül. Ezután a <CRSR JOBBRA> billentyű háromszori lenyomásával elérhetjük, hogy a kurzor a 0-ra kerüljön. Ha most megnyomjuk az <5> billentyűt, akkor a 0 eltűnik, és helyette az 5 jelenik meg. Ezután megnyomva a <RETURN> billentyűt a képernyőn a következőt olvashatjuk:

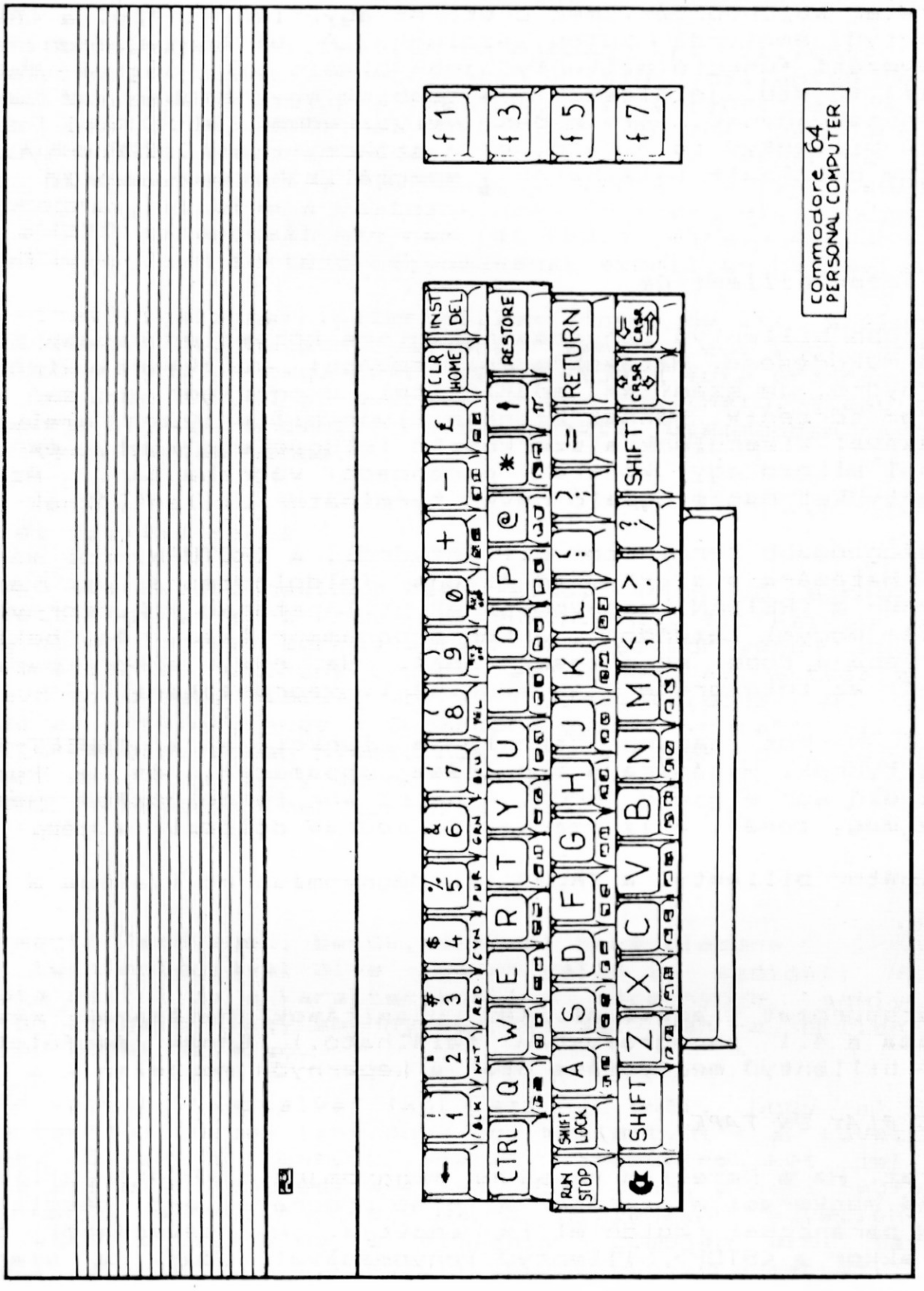
```
.4
?DIVISION BY ZERO ERROR
READY.
■       villog!
```

(A *'READY.'* nem törli a képernyőt, ezért az előző üzenet (*?DIVISION BY ...*) még látszik, de nincs hiba!)

A képernyő szerkesztő

A C-64 számítógép **teljes képernyős szerkesztővel** rendelkezik, melynek működésére már röviden utaltunk. Parancsok bevitelére a képernyő tetszőleges részét használhatjuk, illetve a képernyőn levő programsorokat tetszőleges sorrendben módosíthatjuk a kurzor megfelelő helyre pozícionálásával, s a kívánt karakterek felülírásával - innen az elnevezés.

A C-64 billentyúzete



A billentyűzet felosztása

A C-64 billentyűzetén összesen 65 billentyű található. Ezek közül 64 a hardver szempontjából teljesen egyenértékű, csak az interpreter különbözteti meg őket. Az egyetlen kivétel a <RESTORE> billentyű, amelyről külön szólunk. A billentyűzeten a négy úgynevezett **funkció** billentyű jobb oldalt elkülönítve található meg (f1,f3 stb. jelöléssel), a többiek egy tömbben, az amerikai írógépszabványnak megfelelően helyezkednek el. Alul egyetlen hosszú billentyű található, ez a **szóköz** (space) billentyű. Jobbfelette baloldalt találhatók a speciális hatású **vezérlő** billentyűk.

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, amelyek lenyomásával kikerülünk a szerkesztő felügyelete alól, és a gép azonnal elkezd egy konkrét parancsot végrehajtani. Az ilyen billentyűket összefoglaló néven **terminátor** billentyűknek hívjuk.

A legfontosabb terminátor billentyűről, a <RETURN>-ről már szóltunk. Hatására a szerkesztő átadja feldolgozásra azt a 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 <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

```
LOAD
RUN
```

parancssorozat kiadásával. (Az utasítások hatásának részletes leírása a 4.1 paragrafusban található.) Ennek megfelelően a <RUN> billentyű megnyomása után a képernyőn megjelenik a

PRESS PLAY ON TAPE

felirat. Ha a kazettás egységen megnyomjuk a <PLAY> billentyűt, a C-64 megkeresi a szalagon az első programfile-t, betölti, majd a RUN paranccsal rögtön el is indítja. Ha tévedésből nyomtuk meg, akkor a <STOP> billentyű lenyomásával tudunk a szerkesztő

módba visszatérni. Ne keverjük össze a <RUN> billentyűt és a RUN parancsot. Az előbbi hatására a fentebb vázoltak történnek, míg a RUN parancsot betűnként kell begépelnünk, és utána még a <RETURN> billentyűt is meg kell nyomnunk.

Az utolsó terminátor 'billentyű' valójában két billentyű egyidejű lenyomását jelenti. Pontosabban a <STOP> billentyű lenyomva tartása közben lenyomjuk a <RESTORE> billentyűt. Ezt a billentyűzést <STOP-RESTORE>-ral jelöljük. Ennek hatására az esetleg futó BASIC program futása megszakad, a képernyő pedig törlődik. A billentyűzés a VIC hanggenerátorát is kikapcsolja. A tárolt program, illetve a változók nem vesznek el, a program azonban a CONT paranccsal már nem indítható újra. A <RESTORE> billentyű önmagában történő lenyomásának semmilyen hatása sincs.

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 ezekután a CONT paranccsal tovább indítható. (Lásd a 4. fejezetben a STOP parancsot!) Ha szerkesztő üzemmódban nyomjuk le, semmilyen hatása sincs.

Logikai és fizikai sorok

A képernyő szerkesztő maximum két valódi képernyő sorból egy úgynevezett **logikai sort** képes létrehozni. Két fizikai sor akkor válik egyetlen logikai sorrá, amikor az első sor végén egy nem kurzor karakterrel 'átírunk'. Ebben az esetben az aktuális sor alatt levő sorok egy fizikai sorral lejjebb tolódnak. A logikai sornak az az értelme, hogy a Commodore gyári nyomtatók általában 80 karakter szélességben nyomtatnak. Így a képernyő *szimulálja* a nyomtatót. Azt, hogy mely sorok vannak összefűzve, a 217-242 ($00D9-$00F2) memóriacímek tartalma határozza meg.

Váltók

Az információ (szövegek, betűk, számok) begépelésére - írógépen használt terminológiával élve - három **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.

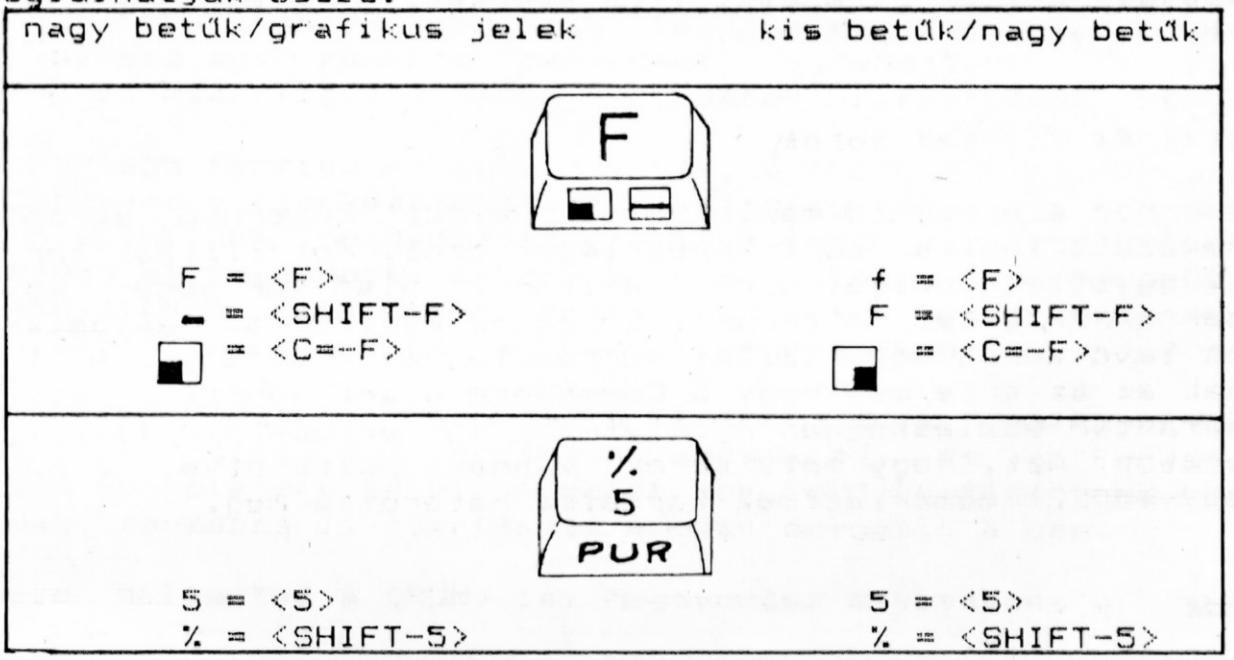
A <SHIFT> váltót használva (ami azt jelenti, hogy a SHIFT lenyomva tartása közben lenyomunk egy billentyűt!) a következők történnek. Olyan billentyűk esetén, amelyekre két jel van ráírva, a siftelés (emelés) a felső jelet adja, a neki megfelelő karakter kerül a képernyőre. Ha a billentyűn egyetlen jel látható, akkor a siftelés a billentyű gombján **elől látható jobboldali** karakter (grafikus jel) begépelését eredményezi.

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

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

A C-64 a képernyőn két karakterkészletet tud megjeleníteni (de nem egyidőben). Az egyiket '**nagy betűk/grafikus jelek**' karakterkészletnek hívjuk. Ha ezt használjuk, a betűket csak nagy betűs alakjukban tudjuk a képernyőre írni. A másikat '**kis betűk/nagy betűk**' karakterkészletnek hívjuk, mert ebben az esetben a betűk kis betűkként íródnak a képernyőre, míg a <SHIFT> váltót használva nagy betűket kapunk (ez az üzemmód felel meg az *írógépnek*)
Egyik karakterkészletről a másikra a <SHIFT> és a <C=> váltók **egyidejű** megnyomásával térhetünk át. (Ez is tekinthető teminátor billentyűnek, mert bármilyen körülmények közt végrehajtódik.)

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



Vezérlő karakterek

Bizonyos billentyűk megnyomásának nem az a hatása, hogy valamely karakter megjelenik a képernyőn. Ilyenek voltak például a <CRSR JOBBRA> illetve a <CRSR FEL> billentyűk.Ezeket a karaktereket összefoglaló néven **vezérlő karakter**eknek hívjuk. A következőkben ezekről lesz szó.

Színvezérlés

Lehetőség van a képernyőre kerülő karakterek színének beállítására. A <CTRL> vagy a <C=> lenyomva tartása közben egy színbillentyű leütése azt eredményezi, hogy az azt követően beírt valamennyi karakter a billentyűkombinációnak megfelelő színnel jelenik meg a képernyőn. (A billentyűzés után a kurzor színe *azonnal* megváltozik!) A színbillentyűk azonosak az 1-8 számbillentyűkkel. A <CTRL>-lal kiválasztható szín az oldalukra van írva.

Kurzor vezérlés

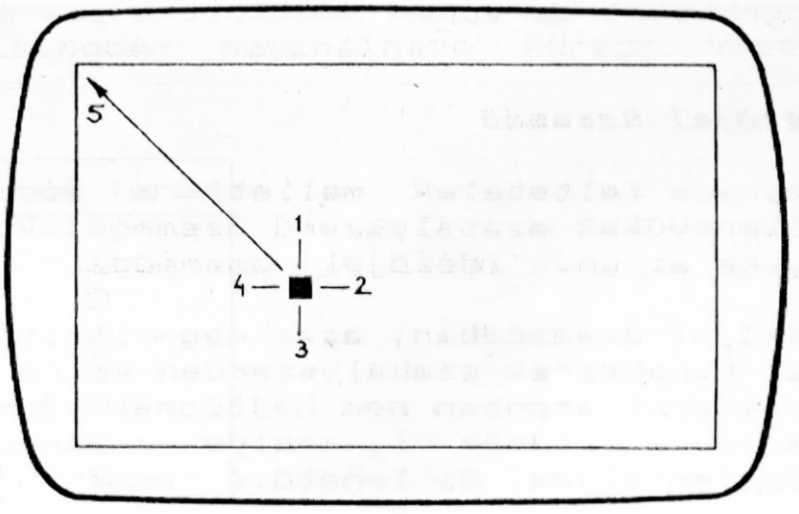
A kurzor billentyűk teszik lehetővé, hogy a kurzort - a képernyőn látható információ megváltoztatása nélkül - tetszőleges helyre pozicionáljuk. Hat olyan billentyű van, amelyeket ebbe a kategóriába sorolunk.

A <CLR> billentyű lenyomása **törli a képernyőt** és a kurzort a bal, felső sarokba viszi vissza. (Ez az un. **home** pozíció.)

A <HOME> lenyomása nem törli a képernyőt, csak a 'home' pozícióba viszi a kurzort.

A <CRSR JOBBRA>, a <CRSR BALRA>, a <CRSR FEL> és a <CRSR LE> billentyűk a feltüntetett iránynak 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> hatástalan. Ha a képernyő jobb vagy bal szélén lépünk át a <CRSR JOBBRA>, illetve a <CRSR BALRA> billentyűvel, akkor a képernyő másik szélére, egy sorral lejjebb, illetve feljebb lépünk vissza.

1 = <CRSR FEL>
2 = <CRSR JOBBRA>
3 = <CRSR LE>
4 = <CRSR BALRA>
5 = <HOME>



Első pillanatra meglehetősen bonyolultnak tűnhet mindez, de a gép mellé ülve, azonnal kipróbálhatjuk valamennyi vezérlő billentyű hatását. Arra kell csak ügyelnünk, nehogy valamelyik terminátor billentyűt megnyomjuk. Ha már megszoktuk a képernyőre való írást, előre tervezzük el, mit szeretnénk és hogyan kiírni, majd pedig próbáljuk tervünket meg is valósítani!

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 előzőekben említettük, hogy a <RETURN> billentyű megnyomása után a képernyő szerkesztő azt feltételezi, hogy a kurzort tartalmazó sort kívántuk feldolgozásra átadni az interpreternek. (Vannak olyan mikrogépek, ahol adatok és programsorok bevitelére csak a képernyő utolsó sorát használhatjuk.) Emiatt a tulajdonsága miatt hívjuk a C-64 szerkesztőjét *teljes képernyős* szerkesztőnek. Ez lehetővé teszi, hogy a képernyőn levő szövegrészeket felhasználjuk a parancs vagy programsor végleges 'megfogalmazásához'.

Különösen kényelmes ez hibás programsorok javításánál. Kilistázzuk a szóbanforgó programsort, majd a vezérlő karakterek segítségével kijavítjuk. Ha a <RETURN> megnyomásának pillanatában a kurzor - bárhol - a javított sorban volt, az interpreter a régi programsort az éppen javítottra cseréli.

Idézőjel üzemmód

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

Idézőjel üzemmódban, azaz megnyitott idézőjel után, a 'közönséges' karakterek szabályszerűen kiíródnak a képernyőre, a vezérlő karakterek azonban nem hajtódnak végre. Helyettük inverz grafikus jelek íródnak ki, amelyek a lenyomott vezérlő karaktereket reprezentálják. Ez lehetővé teszi, hogy a programokban a sztrin-

gek 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><CRSR LE><CRSR LE><CTRL-RVSON>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 harmadik sorába egy inverz A betűt ír.

Idézőjel üzemmódban a <CTRL-RVSON> nem 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 a kurzor mozgatására, csak a <DEL> billentyűt visszaléptetésre; vagy előbb a megkezdett sort a <SHIFT-RETURN> billentyű lenyomásával be kell fejeznünk. Egy másik lehetőség, (ha már nics szükségünk arra a sorra) a <STOP-RESTORE> billentyű lenyomása. (Valamennyi terminátor billentyű megszünteti az idézőjel üzemmódot.) A sztringben használható kurzor vezérlő karakterek az alábbiak:

Vezérlő karakter	Megjelenési forma	
<CRSR FEL>	▢	▯
<CRSR LE>	▯	▢
<CRSR JOBBRA>	▮	▮
<CRSR BALRA>	▮	▮
<CLR>	▙	▚
<HOME>	▚	▚
<INST>	▐	▌

Karakterek törlése/beszúrása

A szerkesztő lehetőséget biztosít arra, 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, un. 'inzert' vagy *beszúrási üzemmód*ban dolgozik. Az <INST> hatását a következő ábrán szemléltethetjük:

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

A beszúrási üzemmódot a <RETURN>, a <SHIFT-RETURN>, a <RUN>, a <STOP-RESTORE> billentyűk lenyomásával, vagy valamennyi hely betöltésével lehet megszüntetni.

Beszúrási üzemmódban a kurzor és a szín-vezérlő 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.)

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

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

Ha ezt a 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 a 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 idézőjel üzemmódba, ha a kurzorral átlépünk egy idézőjelen. Ugyanígy, ha a megnyitott idézőjelet a <DEL> billentyűvel töröljük, a szerkesztő továbbra is idézőjel üzemben dolgozik!
Az inverz alakú vezérlő karakterek a következők:

Funkció	Billentyűzés	Megjelenési forma
<RETURN>	M	
<SHIFT-RETURN>	<SHIFT> M	
kis betűk/nagy betűk	N	
nagy betűk/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>
<CRSR FEL><CRSR JOBBRA> nyolcszor!
<CTRL-RVSON><SHIFT-M><RETURN>
```

A LIST 10 parancs a programsort a következő alakban listázza:

```
10 REM "
FOPROGRAM
```

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
<RETURN>	13
<SHIFT-RETURN>	141
kis betűk/nagy betűk	14
nagy betűk/grafikák	142
<SHIFT-C=> letiltása	8
<SHIFT-C=> engedélyezése	9
Home	19
CLR	147
Kurzor le	17
Kurzor fel	145
Kurzor jobbra	29
Kurzor balra	157
Inverz karakterek	18
Normál karakterek	146
Törlés	20
Beszúrás	148
f1	133
f3	134
f5	135
f7	136
f2	137
f4	138
f6	139
f8	140
Fehér	5
Piros	28
Zöld	30
Kék	31
Narancs	129
Fekete	144
Barna	149
Világos piros	150
Szürke 1	151
Szürke 2	152
Világos zöld	153
Világos kék	154
Szürke 3	155
Bíbor	156
Sárga	158
Encián	159

3. fejezet

BASIC interpreter

3.1 BASIC: Összefoglalás

3.1.1 A C-64 BASIC felépítése

A C-64 személyi számítógép programozásához az alábbi - a gépbe beégetett - programkomponensek állnak rendelkezésre:

a/ képernyő szerkesztő;
b/ BASIC interpreter;
c/ a perifériák kezelését végző 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 BASIC interpreter két részből áll: egyrészt a programok szerkesztését biztosító **programszerkesztőből**, másrészt a BASIC **futtató rendszer**ből. Maga a futtató rendszer meglehetősen egyszerű, például semmilyen nyomkövetési lehetőséget sem biztosít. A programokat a

RUN (<sorszám>)
GOTO <sorszám>
GOSUB <sorszám>

parancsokkal, vagy a <RUN> billentyű lenyomásával indíthatjuk el. A GOTO illetve a GOSUB parancsok nem törlik a változók értékét, a RUN igen.

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 BASIC program futását a <STOP> billentyű megnyomásával bármikor megállíthatjuk. A CONT paranccsal ekkor is folytatható a program (vagy parancs) végrehajtása.

A programszerkesztő hagyományos. A BASIC sorok tetszőleges sorrendben beírhatók, a törlést egy üres - csak a sorszámot tartalmazó - sor beírásával érhetjük el. Egy meglévő sor újraírása a szóbanforgó sor módosítását eredményezi. A program szövegét (vagy annak egy részét) a LIST paranccsal listázhatjuk ki. A teljes programot a **NEW** paranccsal törölhetjük.

A C-64 BASIC-ből hiányzik számos programszerkesztő parancs. Ilyenek például: *APPEND* (programok összefűzése), *DELETE* (több programsor törlése), *RENUMBER* (a program újraszámozása) stb. A C-64 BASIC bővítései ezeket általában biztosítják.

A kész programokat a **SAVE** utasítás segítségével menthetjük lemezre, kazettára vagy egy másik számítógépre (az RS-232-es csatorna segítségével). A tárolt programot a **LOAD** utasítás segítségével tölthetjük be a memóriába.

Magának a C-64 BASIC-nek két érdekessége van. Egyik, hogy a perifériákra vonatkozó utasítások minden egyes perifériára megegyeznek. (Nincs például külön LPRINT utasítás. A nyomtatóra is a PRINT utasítás segítségével írhatunk.) Másik, hogy nem tesz szigorú különbséget parancs és utasítás között. Szinte valamennyi utasítás használható parancsként és programban is.

A C-64 BASIC nem teszi kötelezővé az értékadó utasításban a LET használatát. A **változók neve** akármilyen hosszú lehet, ennek csak a beírható sor maximális hossza szab határt. Megkülönböztetésük azonban csak az **első két karakter**ük alapján történik. Felismeri az **egész**, a **valós** és a **sztring típus**okat. Az egész illetve a sztring típusok jele a név után írt **%** illetve **$** jel. 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 szorzás, osztás jele: *, illetve /. A hatványozás jele a felfelé mutató nyil:↑. Logikai kifejezésekben a <, >, <=, >=, =, <> relációs jelek használhatók.

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

IF...THEN...
ON...GOTO...
ON...GOSUB...

utasítások. Az *ELSE* utasítást a C-64 BASIC nem ismeri.

A ciklusutasítás

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

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

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

A C-64 **ki-/bemeneti utasítás**ainak használatához ismerni kell az un. **elsődleges kimeneti illetve bemeneti eszköz** fogalmát. Elsődleges bemeneti **eszköz** mindig a **billentyűzet.** Innen várja az inputot a **képernyő szerkesztő** és az INPUT utasítás. Az elsúdleges inputról bevitt adatok a képernyőn mindig megjelennek. Az **elsődleges output eszköz a képernyő.** A rendszerüzenetek és a PRINT utasításban adott mennyiségek mindig ide íródnak ki. Az elsődleges output azonban tetszőleges megnyitott file-ba átirányítható a **CMD** utasítás segítségével. Programhiba után az elsődleges output-eszköz automatikusan a képernyő lesz.

További beviteli utasítás a **GET.** Hatására a billentyűzeten éppen lenyomott billentyű a GET-et követő sztringváltozóba kerül, mint egy 1 hosszúságú sztring. Ha nincs benyomva egyetlen billentyű sem, a változóba az üres sztring kerül.

Az elsődleges output-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 új sorba nyomtat.

Az **INPUT** utasítás a szokásos. Az input sor - a puffer mérete miatt - legfeljebb **80 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.

Adatok 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 nincs paramétere, ezért az olvasást csak a DATA-k elejére állíthatjuk vissza.

A file-ok használatára egységesen az **OPEN, CLOSE, INPUT#, GET# és PRINT#** utasítások szolgálnak. A file-ok használatához bizonyos azonosítókat meg kell adnunk. 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 utasításban adjuk meg, s attól kezdve a programban vagy parancsban ezzel azonosíthatjuk a file-t. A PRINT# utasítás ASCII alakban írja a számokat is a file-ba. Az INPUT# utasítás az első CHR$(13) karakterig olvassa az input sort, és azt azután az INPUT utasításhoz hasonlóan dolgozza fel. A GET# utasítás egyetlen byte-ot olvas a file-ból.

A C-64 hanggenerátorát, valamint a video chip legtöbb funkcióját a BASIC-ből csak a POKE/PEEK utasításpár segítségével használhatjuk. (Vagy gépi kódú alprogramok segítségével.) Hasonlóan tudunk csak a C-64-hez csatlakoztatott botkormányok és/vagy potméterek állapotáról is tájékozódni.

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 programnyelvét. De még ebben az esetben is lehetnek értelmezési problémák. Helyesnek tekinthetünk-e egy 10 GOSUB 132 utasítást, ha a 132-es sor nem 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-64 BASIC sokban hasonlít a Commodore cég nagyobb gépein futó BASIC-ekhez. Ezek valójában kompatibilisek, nem csak a forrásnyelv, hanem a gépi megvalósítás szintjén is. Ha valaki elsajátítja a C-64 programozását, az azonnal képes a VC-20, a PET 600/700-as típusú gépek használatára. Mindegyik BASIC valójában egy egyszerűsített **Microsoft BASIC**. Az egyszerűsítés elsősorban a perifériák kezelésére vonatkozik.

BASIC programok írására több, mint 38000 byte áll rendelkezésre. Ekkora területen már rendkívül bonyolult és összetett programokat lehet írni; ezért mindenképp szükség van a programok elemibb részekre való felbontására. Célszerű a programot minél bővebben kommentálni; erre a **REM** utasítás szolgál. Igaz, hogy a tul sok REM lassítja a program futását, de a program kipróbálása után ezek a megjegyzések törölhetők. A HELP+ külön parancsot is biztosít erre a célra (a #C-t, lásd a 9. fejezetben). Az alprogramok struktúráját úgy célszerű kialakítani, hogy a rutin listázáskor elférjen a képernyőn. Könyvünkben a szövegközi és a programlistán való dokumentálást egyszerre használtuk. Programfejlesztés során az előbbi nyilván nem jöhet szóba.

A C-64 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 4.1 paragrafusban felsoroljuk, hogy melyik alapszót, hogyan lehet rövidíteni.

3.2.2 Típusok

A számítástechnikában nemcsak számokkal dolgozunk, s ráadásul a számoknak is sokféle típusa lehet. A Microsoft BASIC interpreterek három **adat típust** 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 feldolgozását. A különböző típusokkal más és más műveleteket végezhetünk. A C-64 BASIC ehhez a három típushoz némileg hasonlóan kezeli a **függvénydefiníciókat** is.

3.2.3 Konstansok

A C-64 BASIC három fajta konstans használatát engedi meg. Ezek azonosak a három adat típussal:

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

A többször használt konstansokat célszerű külön változóban tárolni. Ez egyrészt meggyorsítja a program végrehajtását, másrészt a program könnyebben módosítható.

1/ Az *egész konstansok* 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 konstansok* a következő részekből állhatnak:

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

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

$$I + \frac{F}{10^{n}} * 10^{k},$$

ahol n az F törtrész jegyeinek száma. Az a/-c/ részek alkotják tehát 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, 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*) *konstansok* alfanumerikus és grafikus jelek sorozatát jelentik. A sztring konstansok hosszát csak az köti meg, hogy a képernyőről legfeljebb 80 karakter hosszú sort lehet bevinni. A sztring konstansokat idézőjelek (") közé kell írni. Így 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?"
" 0"
"" (üres sztring, nem egyenlő CHR$(0)-val!)
```

3.2.4 Változók

A 'változó' fogalma az algebrából került át a számítástechnikába. Eredetileg egy pontosan meg nem nevezett mennyiség jelölésére szolgált. Algebrai értelemben mi nem használjuk, hiszen X=X+1 lehetetlen. A korai BASIC interpreterek ezért tették kötelezővé a LET szó használatát az értékadó utasításban: LET X=X+1 (pl. ZX-81-en). Az Algol erre az X:=X+1 jelölést használja. A számítástechnikában éppen ezért célszerűbb a változókat **tároló egység**eknek felfogni. Ekkor az X=X+1 értékadás jobb oldalán szereplő X a tároló tartalmát jelenti az utasítás végrehajtása **előtt**, a baloldali X pedig a tároló tartalmát jelenti a végrehajtás **után**. A C-64 BASIC interpreter a változók három típusát különbözteti meg, attól függően, hogy egész, valós vagy sztring mennyiségek tárolására szolgál-e. Ezen kívül megkülönböztet **egyszerű** vagy **indexes** (tömb) változókat is. 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 elvégzi. Például az L%=L/256 értékadásban L/256-ot egészre kerekíti, ellenőrzi, hogy a (-32768, 32767) intervallumba esik-e, s ha igen, akkor 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 4.1 paragrafusban!)

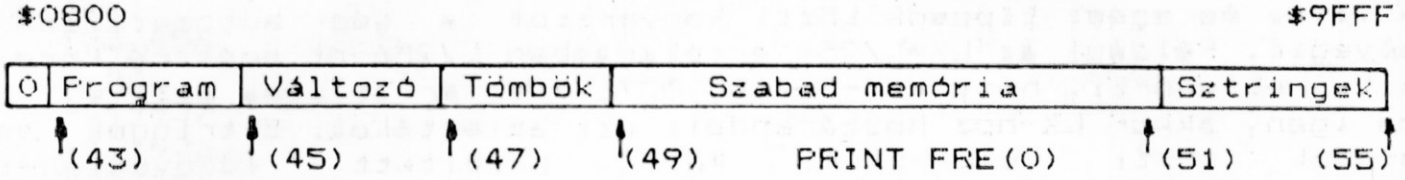
A változónevek képzésének szabályai a következők:

(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*I, *WOR*D nem megengedett nevek). A védett változók TI, ST, TI$ lehetnek változó nevek részei mert nem számítanak alapszónak (FAN*TI* 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 az eredmény tárolására 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 memóriában a következőképpen helyezkednek el:



A tárolt program egyes komponenseinek első byte-jaira mutatnak a 0.lapon levő mutatók:

BASIC program kezdete	(43)	($2B)	(általában $0801)
BASIC terület vége	(55)	($37)	(általában $9FFF)
BASIC változók kezdete	(45)	($2D)	
BASIC tömbök kezdete	(47)	($2F)	
BASIC tömbök vége + 1	(49)	($31)	
BASIC sztringek eleje	(51)	($33)	

A BASIC munkaterület elejét, végét a (43), (55) mutatókkal magunk is állíthatjuk. A többi mutató értéke a program szerkesztésétül, illetve futásától függően alakul.

A BASIC interpreter nem abban az alakban tárolja a programsorokat, ahogy beírtuk. A programsor számát két byte-os számmá konvertálja. Szokás szerint először az alsó, majd a felső byte kerül tárolásra. A programsorban előforduló BASIC alapszavakat az interpreter egy byte-on tárolja. Ezt a byte-ot az alapszó **token**-jének hívjuk. A BASIC utasítások tokenjeit az alábbi táblázat tartalmazza:

Token		Belépési pont		Alapszó	
$80	128	$a830	43056	end	eN
$81	129	$a741	42817	for	fO
$82	130	$ad1d	44317	next	nE
$83	131	$a8f7	43255	data	dI
$84	132	$aba4	43940	input#	iN
$85	133	$abbe	43966	input	input
$86	134	$b080	45184	dim	dA
$87	135	$ac05	44037	read	rE
$88	136	$a9a4	43428	let	
$89	137	$a89f	43167	goto	gO
$8a	138	$a870	43120	run	rU
$8b	139	$a927	43303	if	if
$8c	140	$a81c	43036	restore	reS
$8d	141	$a882	43138	gosub	goS
$8e	142	$a8d1	43217	return	reT
$8f	143	$a93a	43322	rem	rem
$90	144	$a82e	43054	stop	sT
$91	145	$a94a	43338	on	on

$92	146	$b82c	47148	wait	wA
$93	147	$e167	57703	load	lO
$94	148	$e155	57685	save	sA
$95	149	$e164	57700	verify	vE
$96	150	$b3b2	46002	def	dE
$97	151	$b823	47139	poke	pO
$98	152	$aa7f	43647	print#	pR
$99	153	$aa9f	43679	print	?
$9a	154	$a856	43094	cont	cO
$9b	155	$a69b	42651	list	lI
$9c	156	$a65d	42589	clr	cL
$9d	157	$aa85	43653	cmd	cM
$9e	158	$e129	57641	sys	sY
$9f	159	$e1bd	57789	open	oP
$a0	160	$e1c6	57798	close	clO
$a1	161	$ab7a	43898	get	gE
$a2	162	$a641	42561	new	nE

$b4	180	$bc39	48185	sgn	sG
$b5	181	$bccc	48332	int	int
$b6	182	$bc58	48216	abs	aB
$b7	183	$0310	784	usr	uS
$b8	184	$b37d	45949	fre	fR
$b9	185	$b39e	45982	pos	pos
$ba	186	$bf71	49009	sqr	sQ
$bb	187	$e097	57495	rnd	rN
$bc	188	$b9ea	47594	log	log
$bd	189	$bfed	49133	exp	eX
$be	190	$e264	57956	cos	cos
$bf	191	$e26b	57963	sin	sI
$c0	192	$e2b4	58036	tan	tan
$c1	193	$e30e	58126	atn	aT
$c2	194	$b80d	47117	peek	pE
$c3	195	$b77c	46972	len	len
$c4	196	$b465	46181	str$	stR
$c5	197	$b7ad	47021	val	vA
$c6	198	$b78b	46987	asc	aS
$c7	199	$b6ec	46828	chr$	cH
$c8	200	$b700	46848	left$	leF
$c9	201	$b72c	46892	right$	rI
$ca	202	$b737	46903	mid$	mI

A fenti táblázatban nem szerepelnek azok az alapszavak és jelek, melyeknek nincs önálló belépési pontja. A 'belépési pont' megnevezés - a vonal feletti alapszavak esetén - kicsit félrevezető. Az interpreter ezek esetén a belépési pontként megjelölt értékek alsó és felső byte-ját a verembe tölti, majd végrehajt egy gépi kódú RTS utasitást. (Az M6510-es processzor esetén ez megszokott módja az indirekt vezérlésátadásnak.) Ennek hatására a processzor a verem tetején levő két byte-os mutatót az utasításszámlálóba tölti, és az így kapott címet megnöveli

eggyel. Ettől a címtől folytatódik a program futása. Például a NEW táblázatban szereplő belépési pontja $A641 (42561). A helyes vezérlésátadást a következőképpen érhetjük el:

```
BASIC-ből:    SYS 42561+1
Gépi kódból: LDA #$A6/ PHA/ LDA #$41/ PHA/ RTS
```

(A többi alapszó esetén a belépési pont a rutin tényleges **kezdőcímét** jelenti.)

A külön belépési ponttal nem rendelkező alapszavak a következők:

$a3	163	tab(	$a4	164	to
$a5	165	fn	$a6	166	spc(
$a7	167	then	$a8	168	not
$a9	169	step	$aa	170	+
$ab	171	-	$ac	172	*
$ad	173	/	$ae	174	↑
$af	175	and	$b0	176	or
$b1	177	>	$b2	178	=
$b3	179	<	$cb	203	go

Az interpreter a <pi>-t, aminek tokenje 255 ($ff) egészen speciálisan kezeli.

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. Kivételt képeznek a programban szereplő szövegkonstansok. Ezek esetében a sztringváltozó mutatója a program megfelelő helyére mutat vissza. A tömbök az egyszerű változók után helyezkednek el, 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 változóknál 3 illetve 2 felesleges byte is van.

Tömbváltozók tárolása

Egy tömbváltozó tárolóterülete a tömbre vonatkozó legfontosabb (az ún. leíró) adatokkal (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 leírója tartalmaz még egy relatív mutatót, amelyik a következő tömbváltozó első byte-jára mutat. A tömbelemek oszlopfolytonosan helyezkednek el

a memóriában, a DIM A(1,2) deklaráció hatására például ilyen sorrendben:

A(0,0),A(1,0),A(0,1),A(1,1),A(0,2),A(1,2).

Ugyanezeket az elemeket mátrix alakban is elképzelhetjük:

A(0,0) A(0,1) A(0,2)

A(1,0) A(1,1) A(1,2)

A C-64 a változókat a következőképpen tárolja:

Típus	Név	Tartalom				
lebegőpontos	ASCII	ASCII vagy 0	exponens	mantissza M1	M2	M3 M4
egész	ASCII+128	ASCII+128 vagy 128	felső	alsó	0	0 0
sztring	ASCII	ASCII+128 vagy 128	hossz	mutató alsó	felső	0 0
függvény	ASCII+128	ASCII vagy 0	mutató (definició) alsó	felső	mutató (változó) alsó	felső 0

A név két byte-on való tárolását az teszi lehetővé, hogy az alfanumerikus jelek ASCII kódjai mind kisebbek 128-nál. A nevet tartalmazó byte-ok 7. bitjei - mint fent is látható - felhasználhatók a négy típus jelzésére.

Egy tömb a memóriában 5+2*dimszám+(dim$_1$+1)*...*(dim$_n$+1)*k helyet foglal, ahol k=2,3 vagy 5, attól függően, hogy a tömb egész, sztring vagy valós típusú. Sztringtömb esetén ehhez még hozzá kell számítani annyi byte-ot, amennyi a tömb sztringjeinek az össz hossza, hiszen a 'tömb' csak a sztringek hosszát és a mutatót tartalmazza. Maguk a sztringek a sztringterületen helyezkednek el. A tömb egyes elemei a következőképpen helyezkednek el:

tömbnév	rel. mutató		dimen. szám	dimn+1		dim1+1		...adatok
	alsó	felső		felső	alsó	felső	alsó	

A tömbnév két byte-jának 7. bitjei - az egyszerű változókhoz hasonlóan - jelzik a tömb típusát.

Mint már utaltunk rá, a BASIC munkaterület elejét és végét magunk is beállíthatjuk. A többi mutató értéke a program futásától függően alakul. Bizonyos esetekben a BASIC munkaterület végét át is kell állítanunk. Ilyen eset az RS-232-es csatorna használata, vagy a bittérképes üzemmód. A munkaterület átállításánál csak arra kell tekintettel lennünk, hogy a BASIC interpreter csak a következő CLR kiadásakor veszi tudomásul az új értékeket. Ennek megfelelően a munkaterület átállítása a program elején történik.

A következő program egyszerre demonstrálja a BASIC munkaterület átállítását és a változók memóriában való elhelyezkedését. A BASIC munkaterületet az alábbi program ugyanis a képernyő memóriával azonosítja, így a képernyőn pontosan látjuk, mi történik a változókkal. A futás után a SYS 64738 parancs kiadásával visszaállítjuk a 'normális' üzemmódot. (Vigyázat, a program ekkor elvész!)

```
10 PRINT"<CLR>";CHR$(14);CHR$(8)
20 POKE 45,40: POKE 46,4
30 POKE 55,232: POKE  56,7: CLR
40 J=13*4096+8*256
50 FOR K=J TO J+999: POKE K,14: NEXT K
60 A=1234: FOR I=1 TO 500: NEXT I
70 FOR I=0 TO 10
80 INPUT "<HOME>SZTRING=";SZ$(I)
90 NEXT I
```

3.2.6 Kifejezések

Az algebrában tanultakhoz hasonlóan konstansokból, változókból és egy-, illetve többváltozós műveletekből építhetünk 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 konstansok;
b/ sztring változók;
c/ minden olyan függvény, amelynek a nevében a $ jel szerepel. Ezek: CHR$, LEFT$, MID$, RIGHT$, STR$. (Természetesen a függvények argumentumainak megfelelő típusú kifejezéseknek kell lenniük.)
d/ két sztring kifejezés a + jellel összekapcsolva;

A sztring kifejezések tetszés szerint zárójelezhetők.

Példák:

```
"QW ER TY"
AA$+" ROSSZ"
CHR$(13)
STR$(25)+MID$(K$,3,6)
"NEW"+A$+B$
```

A sztring függvények hatásáról a 4.1 paragrafusban részletesen lesz szó. A + sztring-művelet a sztringek **egymáshoz fűzését, 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 sztring függvényeket **nem** tudunk definiálni.

Aritmetikai kifejezések

Az aritmetikai kifejezések esete nemcsak 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 (két byte-os egész számként tárolt) 0-t **hamis**nak, az ettől eltérő értéket **igaz**nak tekinti.

Az összehasonlítások eredménye azonban mindig 0-t vagy -1-et ad eredményül.

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 0-t kapunk.

A relációs jelek értelme, amikor számokat hasonlitunk össze, a szokásos:

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

Sztringek esetén = és <> jelentése értelemszerű. 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 kódszáma az A$ sztringben kisebb, mint a B$-ban. Elképzelhető, hogy az A$ sztring a B$ sztring kezdőszelete (pl. B$="hazmester", A$="haz"). Ebben az esetben A$<B$ mindig teljesül.

Példák:

1=4	hamis	(0)
14>=62	hamis	(0)
14<62	igaz	(-1)
5*5 <> 16	igaz	(-1)
"ABC"<"AX"	igaz	(-1)
"A"<"D"	igaz	(-1)
"A"<"9"	hamis	(0)
"XYZ">"XY"	igaz	(-1)
"XYZU"<="XY9"	hamis	(0)
""<=CHR$(0)	igaz	(-1)
CHR$(0)<="@"	igaz	(-1)

Az eredményről magunk is meggyőződhetünk. Adjuk ki például a PRINT 1=4 parancsot! Válaszul az interpreter az 1=4 állitás logikai értékének (ami hamis) megfelelő számot irja ki. Az eredmény tehát 0 lesz.

Logikai műveletek

A BASIC interpreter összesen három logikai műveletet ismer, egy egyváltozós (NOT) és két kétváltozós (AND, OR) műveletet. Hatásukról részletesen a 4.1 paragrafusban szólunk.

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, COS, EXP, FRE, INT, LEN, LOG, PEEK, POS, RND, SGN, SIN, SQR, TAN, VAL. A felsorolt 17 egyváltozós függvény néhány egészen speciális függvényt is tartalmaz. Ezek hatásáról a 4.1 paragrafusban szólunk részletesen. Az F.6 függelékben szerepelnek azok a matematikai függvények, amelyeket gyakrabban szokás használni, s közvetlenül nem számíthatók ki.

Megjegyezzük, hogy a DEF FN utasítás segítségével további - egyváltozós - aritmetikai függvényeket tudunk definiálni.

Ennyi előkészítés után definiáljuk, hogyan is épülnek fel az aritmetikai kifejezések:

a/ tetszőleges egész vagy valós változó vagy konstans 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
SC↑(D-E)*25
((X-C)↑(D-E)/2)*10)+1
K%=1 AND M<>X
NOT (D=E)
K%=2 OR ( (QA=B) AND (M<X) )
```

A fenti definició egy igen érdekes sajátosságára hívjuk fel a figyelmet. Aritmetikai kifejezések szerepeltetése a logikai kifejezések közt megszokott. A C-64 azonban azonosítja a logikai és aritmetikai kifejezéseket. Tekintsük például a következő értékadást:

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

L$ egy hexadecimális számjegyet 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 0-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

3.3 A BASIC interpreter működése

A C-64 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 (belépési pont = $A480). Az interpreter első lépésként meghív egy rutint (belépési pont = $A560), 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-$0258 címeken. A fenti rutin az inputsor végére egy 0 byte-ot is elhelyez.

A BASIC parancs- vagy programsor szövegét az interpreter byte-onként dolgozza fel. Ehhez a ($7A) mutatót használja, amelyik a BASIC szöveg éppen feldolgozás alatt álló byte-jára mutat. Ennek a byte-nak az akkumulátorba való töltésére szolgál a 0. lapon levő un. CHRGET rutin ($73-$8A). Ha a rutinba a $73 ponton lépünk be, az először eggyel megnöveli a ($7A) mutatót, s csak utána tölti be az akkumulátorba a byte-ot. Ha a $79 ponton lépünk be, akkor a mutató értékének növelésére nem kerül sor. A rutin a szóközöket **átlépi**. Közvetlenül a rutin vége előtt két ellenőrzésre kerül sor. A C jelzőbit alacsony lesz, ha a byte egy számjegy ASCII kódja. A Z jelzőbit abban az esetben lesz alacsony, ha a byte egy kettőspont ASCII kódja, vagy 0. A BASIC interpreter működését legegyszerűbben ennek a rutinnak az átírásával módosíthatjuk.

Miután az interpreter a bevitt sort elhelyezte az input pufferbe, megvizsgálja, hogy számmal kezdődik-e, vagy sem. Ha igen, akkor az inputsort programsornak értelmezi, és elhelyezi a memóriában. Ha nem, akkor parancsnak értelmezi, és végrehajtja. Ez utóbbi tény jelzésére a $58-as memóriába $FF kerül.

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 he-

lyettesítse. Ez a rutin ismeri fel a kulcsszavak rövidítését, a ? utasítást. 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 két 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 nics, a program végrehajtása ?OUT OF MEMORY ERROR hibaüzenettel megszakad, és a vezérlés visszaadódik a képernyő szerkesztőnek. Ha van hely, és az inputsor nem üres, a program 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 0 byte-ból áll), az utolsó lépésekre nem kerül sor. Egy üres sor, pl. 213 <RETURN> bevitele így 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 0, akkor END nélkül ért véget a program, és a vezérlés visszatér a szerkesztőbe. Ha nem, a BASIC program végrehajtása folytatódik.

Mindezek után kerül sor az alábbi rutin végrehajtására, amely egyben a parancs mód esetén az ellenőrző ciklus belépési pontja. Az interpreter 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 megfelelú rutinok végén - hacsak nem történt hiba - a vezérlés az ellenőrző ciklus legelejére adódik vissza - függetlenül attól, hogy parancs (direkt) vagy program módban vagyunk-e.

Külön szólunk a RUN végrehajtásáról. A RUN utasítás végrehajtása egy mutatónak a BASIC szöveg elejére való állításával kezdődik. Ezután a vezérlés az ellenőrzési ciklusba kerül.

A BASIC interpreter működésének ismerete elengedhetetlen a BASIC lehetőségeinek kiterjesztéséhez. A fent jelzett rutinok ugyanis általában egy JMP ($xxxx) utasitással kezdődnek, ahol $xxxx egy-egy RAM címet jelent (általában a 3. lapon). A címek - hacsak nem irtuk 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. A legegyszerűbb lehetőség erre a CHRGET rutin módosítása, amelyik a 0. lapon helyezkedik el. (A rendszer újrainditása esetén az első lapok értékeit az inicializáló program beállítja.)

Végül pár szót szólunk a C-64 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 restart 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 beállitására. 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.

Az M6510-es processzor NMI vonala a CIA#2 perifféria chip IRQ vonalához van kötve. Ez az utóbbi azonban maszkolható. Igy a C-64 számítógépben igazi 'nem maszkolható' megszakítások nincsenek. A CIA#2 megszakító rendszerét az RS-232-es csatorna, illetve a <RESTORE> billentyű használja. A <RESTORE> billentyű - egy kondenzátor segítségével - alacsonyra viszi a processzor NMI vonalát. A megszakító rutin ellenőrzi, hogy a megszakítást az RS-232-es csatorna generálta-e. Ha nem, akkor a megszakítást a <RESTORE> billentyű generálta. Megvizsgálja, hogy a <STOP> billentyűt is lenyomtuk-e. Ha nem, az interpreter befejezi a megszakító rutint. Ha igen, akkor inicializálja a képernyő szerkesztőt. Speciálisan inicializálja a hanggenerátort is.

A BASIC interpreter fontosabb rendszerváltozói

($20)	GOTO,GOSUB,SYS,RUN utasítások paramétere
($3B)	az utoljára végrehajtott utasítás sorszáma
($3D)	a végrehajtás alatt álló utasítás sorszáma
($7A)	az aktuális BASIC byte cime
$0200-$0258	input puffer

A BASIC interpreter fontosabb belépési pontjai

($0300)	BASIC hibaüzenet kiírása
($0302)	BASIC meleg start
($0304)	input puffer tokenizálása
($0306)	ellenőrző ciklus belépési pont
($0308)	BASIC token kiértékelés
($0314)	hardver megszakító rutin(IRQ)
($0316)	megszakító vektor(BRK)
($0318)	megszakító vektor(NMI)

$A480	Basic meleg start
$A49C	programsor elhelyezése a memóriában
$A533	programsorok mutatóinak újraszámozása
$A560	input sor átírása a pufferba
$A579	input puffer tokenizálása
$A7E1	BASIC token kiértékelés
$A871	RUN belépési pont
($FFFA)	nem maszkolható megszakítás(NMI)
($FFFC)	restart
($FFFE)	maszkolható megszakítás(IRQ)

A fenti ismeretekre akkor van szükség, ha magának az interpreternek a működését módosítani akarjuk, vagy gépi kódú programból akarunk BASIC programrészeket felhasználni. Ez utóbbira gyakori példa a BASIC aritmetikai műveleteinek a használata.

Az alábbi gépi kódú programrész, egyetlen - legfeljebb 90 karakterből álló - BASIC parancssort hajt végre - úgy, mintha parancsként adtuk volna ki.

A BASIC parancsot ASCII karaktersorozatként kell megadni, úgy, ahogy a billentyűzetről bevinnénk. A program gondoskodik a sor tokenizálásáról.

A programnak gondoskodnia kell a hibák saját kezeléséről is, hiszen nem biztos, hogy hiba esetén a BASIC hibakiíró rutinját akarjuk használni. Ezért a programban háromféle hibakezelés közt választhatunk:

- a BASIC eredeti hibakezelése: üzenet kiírása, s a program megállítása;
- a hibaüzenet kiírása, de a program fut tovább;
- a hibaüzenet nem kerül kiírásra, s a program fut tovább.

A program listája az alábbi:

```
; Az alprogram a (12*4096+4)-tól elhelyezett és
; a (12*4096+94)-ben tárolt hosszú
; ASCII basic szövegű utasítást hajtja végre.
;
; Hívása: SYS 12*4096
;
; Hiba esetén a hiba kódja (12*4096+95)-ben,
; de a program fut tovabb.
;
; Ha (12*4096+96) nem nulla a hiba szövege kiíródik, ha pedig
; (12*4096+97) nem nulla a BASIC utasítás tárolódik.
; (DEF FN esetén erre szükség van!
```

```
        .mac dcopy          ; szómásolás
       lda ?1
       sta ?2
       lda ?1+1
       sta ?2+1
      .mnd

       .mac dlet            ; szóértékadás
       lda #<?2
       sta ?1
       lda #>?2
       sta ?1+1
       .mnd

       .mac dinc            ; szónövelés
       inc ?1
       bne ?2
       inc ?1+1
       ?2 .mnd

       .mac let             ; értékadás
       lda #?2
       sta ?1
       .mnd


errlbl = $0300  ; hiba kiíró rutin címe
chrget = $73    ; chrget kezdőcíme
ermlbl = $a326  ; hibaüzenetek címei
retprt = $aad7  ; ? (return)
kerprt = $ab45  ; ? "?"
jelprt = $ab47  ; ? az aku
getmut = $7a    ; chrget mutatója
puffer = $0200  ; basic puffer
token  = $a57c  ; tokenizáló rutin
basic  = $a7ed  ; BASIC parancs végrehajtása
mut    = $fb    ; segédváltozó

*=$c000
beq kezdet              ; adatok atlepese
bne kezdet
par *=*+90              ; BASIC utasitas helye
```

```
  hossz .byte 0                 ; az utasitas hossza
hkod .byte 0                    ; a hiba kodja, 0=nincs
errprt .byte  0                 ; 0= nem kell a hibauzenetet nyomtatni
                                ; $ff = kell
jelzo  .byte 0                  ; jelzo 0    nem kell tarolni
                                ;       $ff= tarolni kell
mutato .word memori
errsav .word 0                  ; ide menti a hibarutin cimet
getsav .word 0                  ; ide menti a chrget mutatojat
spsav  .byte 0                  ; ide menti az sp-t


kezdet tsx                      ; sp elmentese
       stx spsav
       dcopy errlbl,errsav      ; hiba rutin kezdocimenek elmentese
       dcopy getmut,getsav      ; chrget mutato elmentese
       dlet mut,$0200
       dlet getmut,$0200
       dlet errlbl,errnew
       let hkod,0               ; puffer vegere egy 0 byte
       ldx hossz                ; utasitas atirasa az input pufferbe
       lda #0
       sta puffer,x
       dex
cikl1  lda par,x
       sta puffer,x
       dex
       bpl cikl1
       jsr token                ; tokenizalas
       lda jelzo                ; el kell-e tenni?
       beq tov1                 ; nem --> vegrehajtas
       dcopy mutato,mut
       dcopy mutato,getmut
       dinc mut
       ldx #0
       ldy #0
loop3  lda puffer,x             ; egy karakter elmentese
       sta (mut),y
       beq tov2                 ; 0 byte --> vegrehajtas
       dinc mut
       inx
       bne loop3
       beq loop3
tov2   dcopy mut,mutato
tov1
loop   jsr chrget               ; vegrehajtas
       jsr basic
       ldy #0
       lda (getmut),y
       beq vege                 ; 0 byte?
       cmp #':                  ; : utasitas elvalasztasa
```

```
        beq loop            ; igen --> vegrehajtas
        ldx #11             ; szintaktikus hiba

errnew  stx hkod            ; hibakod elmentese
        lda errprt          ; kell-e a hibauzenetet nyomtatni
        beq vege
        cpx #128            ; end,stop,next = nem hiba
        beq vege
        txa
        asl a
        tax
        lda ermlbl,x
        sta mut
        lda ermlbl+1,x
        sta mut+1
        jsr retprt          ; ?(return)
        jsr kerprt          ; ? "?"
        ldy #0
cikl2   lda (mut),y         ; hibaszoveg
        pha                 ;    kovetkezo karaktere
        and #$7f
        jsr jelprt
        iny
        pla
        bpl cikl2
        lda #$6a
        ldy #$a3
        jsr $ab1e           ; ?"error"
        jsr retprt


vege    dcopy getsav,getmut ; getchr vissaallitasa
        dcopy errsav,errlbl
        ldx spsav           ; veremmutato visszaallitasa
        txs
        rts                 ; vissza a hivo programhoz
memori  .byte 0
        .end
```

A fenti programrész kipróbálására bemutatunk egy BASIC programot. A program a 1050-ik sorban levő INPUT utasításban megadott függvény értékét számítja ki a 0-21 intervallum egész értékeire.

Ha a kérdésre pl. a X*X+6*X+1 választ adjuk, akkor ennek a függvénynek az értékeit számítja ki a program. Ezt a fenti gépi kódú programrész nélkül nem tudnánk megtenni, hiszen a DEF FN utasításokat előre be kell írni a programba.

Az 1000-1030 programrész a DATA sorokban megadott - s az előzőleg asszembler listán megadott - gépi kódú rutint tölti a helyére. Az 1050 sorban bekérjük a függvény alakját az f$ változóba, majd a "def fn f(x)="+f$ sztringet elhelyezzük a 12*4096+3 cimtől kezdődően. A sztring hosszát a 12*4096+94 címre helyezzük. A 12*4096+97 címre 1-et írunk, jelezve, hogy a feldolgozandó BASIC sort el kell tárolni. Erre azért van szükség, hogy a későbbi függvényhívások 'megtalálják' a függvény definícióját. Az 1140-1160 sorok kiírják a függvény értékeit.

```
1000 c=49152
1010 read a
1020 if a=-1 then goto 1050
1030 poke c,a: c=c+1: goto 1010
1040 :
1050 input "Fuggveny=";f$
1060 f$="def fn f(x)="+f$
1070 poke 12*4096+94,len(f$)
1080 cim = 12*4096+3
1090 for i=1 to len(f$)
1100 poke cim+i,asc(mid$(f$,i,1))
1110 next i
1120 poke 12*4096+97,1
1130 sys 12*4096
1140 for x=0 to 21
1150 print fn f(x)
1160 next x
1170 :
1180 data 240,103,208,101,68,69,70,32
1190 data 70,78,32,70,40,88,41,61
1200 data 88,42,88,42,88,102,124,0
1210 data 0,0,60,96,96,96,60,0
1220 data 0,6,6,62,102,102,62,0
1230 data 0,0,60,102,126,96,60,0
1240 data 0,14,24,62,24,24,24,0
1250 data 0,0,62,102,102,62,6,124
1260 data 0,96,96,124,102,102,102,0
1270 data 0,24,0,56,24,24,60,0
1280 data 0,6,0,6,6,6,6,60
1290 data 0,96,96,108,120,108,17,0
1300 data 0,1,133,193,139,227,173,8
1310 data 246,186,142,104,192,173,0,3
1320 data 141,100,192,173,1,3,141,101
1330 data 192,165,122,141,102,192,165,123
1340 data 141,103,192,169,0,133,251,169
1350 data 2,133,252,169,0,133,122,169
1360 data 2,133,123,169,8,141,0,3
1370 data 169,193,141,1,3,169,0,141
1380 data 95,192,174,94,192,169,0,157
1390 data 0,2,202,189,4,192,157,0
```

```
1400 data 2,202,16,247,32,124,165,173
1410 data 97,192,240,58,173,98,192,133
1420 data 251,173,99,192,133,252,173,98
1430 data 192,133,122,173,99,192,133,123
1440 data 230,251,208,2,230,252,162,0
1450 data 160,0,189,0,2,145,251,240
1460 data 11,230,251,208,2,230,252,232
1470 data 208,240,240,238,165,251,141,98
1480 data 192,165,252,141,99,192,32,115
1490 data 0,32,237,167,160,0,177,122
1500 data 240,61,201,58,240,240,162,11
1510 data 142,95,192,173,96,192,240,47
1520 data 224,128,240,43,138,10,170,189
1530 data 38,163,133,251,189,39,163,133
1540 data 252,32,215,170,32,69,171,160
1550 data 0,177,251,72,41,127,32,71
1560 data 171,200,104,16,244,169,106,160
1570 data 163,32,30,171,32,215,170,173
1580 data 102,192,133,122,173,103,192,133
1590 data 123,173,100,192,141,0,3,173
1600 data 101,192,141,1,3,174,104,192
1610 data 154,96,0,0,-1
```

4. fejezet

BASIC utasítások

4.1 BASIC alapszavak ABC sorrendben

Ez a fejezet tartalmazza a C-64 BASIC interpreter utasításait és függvényeit, azok leírását, működésük részletes ismertetését. 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 'kis betűk/nagy betűk' karakterkészlet használata esetén látható alakját adjuk meg. A 'nagy betűk/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 szóbanforgó alapszó tokenje. Az érvényes tokenek 128 és 203 közé esnek. A fenntartott változóknak nincs külön tokenje.

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

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

A kapcsos zárójelek közt, egymás alatt szereplő részek <u>kötelező választási</u> lehetőségeket sorolnak fel. (Lásd például az ON utasításnál.)

A szintaxis leírásánál szereplő **többi jel (",: stb.) az utasítás része.** A szintaxis leírásánál használt szögletes és kapcsos zárójelek természetesen **nem** részei az utasításoknak.

Példák: A példákat igyekeztünk úgy összeválogatni, hogy az utasítás legtipikusabb használati 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

Rövidítés: aB *Token:* $B6(182)

Mód: mind parancs -, mind program módban használható.

Az ABS jelet zárójelben követő aritmetikai kifejezés abszolút értékét számítja ki.

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

Példák:

```
(i)   465 IF ABS(X-X1)<EPS THEN PRINT "** VEGE **": STOP
(ii)  IF ABS(QY)>1E7 THEN PRINT"**QY TUL NAGY **"
(iii) 148 IF ABS(X%)<3 THEN GOTO 232
```

Az **első példa egyenletek** közelítő megoldásában igen gyakori. Ellenőrzi, hogy a gyök X és X1 közelítései elég közel vannak-e egymáshoz. Ha igen, a program megáll, ha nem, akkor tovább folytatja a számítást.

A második példa az ABS direkt módban való használatát szemlélteti. A program elindítása előtt ellenőrizzük QY értékét. Ha az meghaladja az 1E7 értéket, a gép egy **QY TÚL NAGY** üzenet kiírásával figyelmeztet. Az utasítás természetesen magába a programba is beépíthető, így egy kicsit erőltetett.

A harmadik példa egy NIM játékprogramból való. A gép véletlenszerűen generál gyufaszálakból álló kupacokat. Ha túl kevés (<3) gyufa van egy kupacban, akkor újból generál egy számot.

Hibalehetőségek Ha a zárójelben rossz vagy nem aritmetikai típusú kifejezés áll, akkor különféle hibajelzések generálódhatnak, például szintaktikus hiba, nem megfelelő típus jelzése stb. Ha

az aritmetikai kifejezés kiértékelése túlcsordulást okoz, egy ?OVERFLOW ERROR hibaüzenetet kapunk.

AND

Rövidítés: aN *Token*: $AF(175)

Mód: mind parancs -, mind program módban használható.

Két kifejezés értékének logikai 'és'-ét képezi. Az AND művelet egész számokra is értelmezhető. Az 'és' művelet ekkor bitenként, egymástól függetlenül hajtódik végre. Az AND művelet műveletáblája a következő

AND	igaz	hamis
igaz	igaz	hamis
hamis	hamis	hamis

Szintaxis: <aritmetikai kifejezés> **AND** <aritmetikai kifejezés>

Példák:
```
(i)   PRINT 750 AND -237 : REM = 514
(ii)  100 OK=X>Y AND Y↑Z>20 AND X>15
(iii) 150 IF 0<X AND X<6 THEN GOTO 200
```

Az első példában azt mutatjuk be mi történik, ha aritmetikai kifejezéseket kapcsolunk össze AND-del. A két szám 16-bites megfelelőjével az AND műveletet bitenként elvégezve az eredmény:

```
0000 0010 1110 1110
1111 1111 0001 0010
-------------------
0000 0010 0000 0010
```

Az első példa így végülis 514-et nyomtat.

A következő két példa azt mutatja be, hogyan használhatjuk az AND műveletet összetett logikai kifejezések képzésére. A második példában az OK egész változó értéke -1 (igaz) lesz abban az esetben, ha X>Y, Y↑Z>20, illetve X>15 egyaránt teljesül. Különben OK értéke 0 (hamis) lesz. Az utolsó példa az AND használatát egy összetett feltételes utasításban szemlélteti. A vezérlésátadásra csak abban az esetben kerül sor, ha mind a két feltétel teljesül.

Hibalehetőségek: Ha az operandusok kiértékelése közben hiba történik, a hiba típusának megfelelő jelzést kapunk. Gyakori hiba forrása a műveletek sorrendjének pontatlan ismerete. A C-64 BASIC műveleti hierarchiája megegyezik a FORTRAN és az ALGOL programnyelvekével. Először a NOT, azután az AND, legvégül az OR hajtódik végre. Ügyelnünk kell arra, hogy az interpreter először az aritmetikai műveleteket hajtja végre, s csak azután a logikaiakat.

ASC

Rövidítés: aS *Token*: $C6(198)

Mód: mind parancs -, mind program módban használható

Az ASC jelet zárójelben követő sztring-kifejezés első karakterének C-64 ASCII kódját adja. Az ASC függvény használata azokban az esetekben gyakori, amikor egy karakter kódját (ami egy szám) könnyebb kezelni, mint magát a karaktert.

Szintaxis: **ASC(**<sztring kifejezés>**)**

Példák:

```
(i)   PRINT ASC("A") : REM =65
(ii)  100 X=ASC("ZEBRA") : REM =90
(iii) 10 GET JEL$: IF JEL$="" GOTO 10
      20 JEL=ASC(JEL$): IF JEL=13 THEN RETURN
(iv)  200 JEGY=ASC(KOD$)-48+(ASC(KOD$)>64)*7
```

Az első példa az ASC függvény hatását mutatja. Az A betű helyére bármilyen karaktert (betűt, számot, grafikus jelet) téve megkapjuk annak ASCII kódját. A második példa eredményül 90-et ad, lévén Z ASCII kódja 90.

A (iii) alatti programrészlet azt mutatja, hogy az ASC függvény segítségével a billentyűzetről kapott információt (JEL$) hogyan elemezhetjük. Ezzel az eljárással a teljes billentyűzetet ellenőrizhetjük, kivéve a <STOP> billentyűt. Az ASC függvény helyettesíthető a CHR$ függvénnyel. A JEL=13 helyett rögtön JEL$=CHR$(13)-t kérdezhetnénk. Ekkor persze JEL kiszámítása teljesen felesleges. Más a helyzet, ha JEL$ nem egy 1 hosszúságú sztring. Ekkor csak a 20. sorban látható módszert választhatjuk.

A (iv) alatti példánk egy hexadecimális-decimális konvertáló program része. Ha a KOD$ első karaktere hexadecimális számjegy, akkor JEGY értéke ez a számjegy lesz. Ha például KOD$="B89A",

akkor JEGY=11 a B "számjegy" értéke.

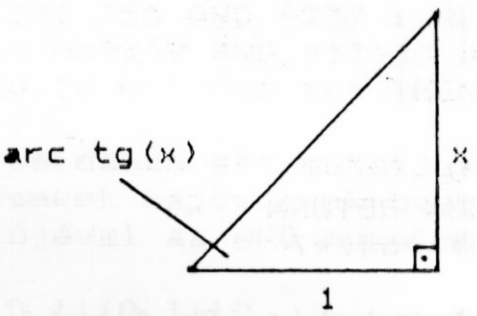
Hibalehetőségek: A kifejezés kiértékelése közben számos hiba fordulhat elő, köztük az, hogy az eredménysztring hossza meghaladja a 255-öt. Ha a kifejezés nem sztring típusú, akkor *nem megfelelő típus* (?TYPE MISMATCH ERROR) üzenetet kapunk. Gyakori hiba, hogy ASC-t hajtunk végre az üres ("") sztringre. Ekkor ?ILLEGAL QUANTITY ERROR hibajelzést kapunk. Ezt elkerülhetjük, ha ASC(J$+CHR$(0))-t használunk ASC(J$) helyett, vagy pedig előre lekérdezzük , hogy az argumentum sztring nulla-e.

ATN

Rövidítés: aT *Token*: $C1(193)

Mód: mind parancs -, mind program módban használható.

Az argumentum arkusz tangensének főértékét számítja ki radiánban. Az alábbi ábra illusztrálja x és arc tg x kapcsolatát:



Szintaxis: ATN(<aritmetikai kifejezés>)

Példák:

```
(i)    PRINT ATN(1): REM =pi/4
(ii)   PRINT ATN(TAN(2*pi+pi/4)): REM = pi/4
(iii)  100 ALFA=ATN(AV/BV): BETA=pi/2-ALFA
(iv)   120 FI=ATN(Y/X): R=SQR(X*X+Y*Y)
```

Az első két példában az ATN hatását mutatjuk be. Mivel az ATN függvény az arkusz tangens főértékét számítja ki, ezért a (ii) példában a végeredmény nem egyezik meg a TAN argumentumával.

Az utolsó két példa geometriai jellegű. A (iii) az AV, BV befogójú derékszögű háromszög szögeit számítja ki. A (iv) program-

sor egy pont (X,Y) derékszögű koordinátáit számítja át polárkoordinátákba. R a pont origótól mért távolsága, FI a pont helyvektorának az X tengely pozitív felével bezárt szöge.

A fenti példákban ALFA, BETA és FI értéke mindig a (-pi/2, pi/2) intervallumba esik. Ha ezt az értéket megszorozzuk 180/pi-vel az eredményt szögekben is megkaphatjuk.

Hibalehetőségek: Az aritmetikai kifejezés kiértékelése közben számos hibalehetőség adódik. Ha a kifejezés kiértékelhető, és nem haladja meg a gépben ábrázolható valós számok felső határát, akkor az ATN rutin már hibajelzés nélkül végrehajtódik.

CHR$

Rövidítés: cH (a $ már nem kell!) *Token*: $C7(199)

Mód: mind parancs -, mind program módban használható.

Egy tetszőleges, a 0<x<255 intervallumba eső számból előállít egy 1 hosszúságú sztringet, amelynek ASCII kódja éppen a szóbanforgó szám. A C-64 BASIC-ben ez az egyetlen kényelmes eljárás bizonyos speciális karakterek (például <RETURN> és ") kezelésére (CHR$(13) és CHR$(34)).

Szintaxis: CHR$(<aritmetikai kifejezés>)

Az aritmetikai kifejezés nem kell, hogy egész legyen. Ilyenkor a kifejezés egész részével számol tovább az interpreter.

Példák:

```
(i)   A$=CHR$(34)+CHR$(20)+CHR$(30)+CHR$(18)+
          "NEV"+CHR$(146)+CHR$(34): ?A$
(ii)  XE$=CHR$(160)+X$
(iii) PRINT#4,CHR$(7);
(iv)  PRINT#4,CHR$(27);"FOPROGRAM"
```

Példáink azt mutatják be, hogyan lehet a CHR$-t speciális szerkesztő karakterek nyomtatására felhasználni. (Ezeket a karaktereket különben igen nehéz előállítani.) Az első példa idézőjelek közé helyez egy sztringet, előtte azonban eléje és utána helyezi a <RVSON> illetve a <RVSOFF> karaktereket. A ?A$ hatására idézőjelek közt jelenik meg a NEV negatív (inverz) képe.

A (ii) példa az X$ sztring elejére egy siftelt szóköz karaktert illeszt. Bizonyos perifériák a sztring elején álló szóközöket nem nyomtatják. Ezt XE$=" "+X$ alakban is begépelhetnénk, de akkor nem látszik, hogy az idézőjelek közt siftelt szóköz van.

A harmadik példa elküld a nyomtatónak egy 'bell' jelet. Bizonyos típusú nyomtatók ilyenkor valóban adnak valamilyen hangjelzést.

A negyedik példa ugyancsak a nyomtatóhoz kapcsolódik; a vezérlő karakter hatására a FOPROGRAM karakterei dupla-széles formában nyomtatódnak ki (CBM típusú nyomtatókon legalábbis).

A CHR$ az ASC inverze. Speciális felhasználási területe a különböző karakterkészletek közti konverzió. A PRINT utasítást használó programok nem írhatóak át közvetlenül PRINT#4-re. Előbb a kinyomtatandó értékek részletes vizsgálata szükséges, hogy a nyomtatási kép megegyezzen a képernyővel.

CHR$(0) egy '**null**' karaktert jelent, de a hossza 1. A nyomtatási képben ennek azonban nincs nyoma. Nézzük például az Y$="123"+CHR$(0)+"321" utasítást. A PRINT Y$ utasítás hatására az 123321 sorozat íródik ki, de LEN(Y$)=7 és VAL(Y$)=123!

Hibalehetőségek: Az argumentum kiértékelése, illetve nem megfelelő értéke számos hibajelzést eredményezhet. A speciális karakterek kódjainak rossz használata esetén a kívánt hatás elmarad.

CLOSE

Rövidítés: cl0 *Token*: $A0(160)

Mód: mind parancs -, mind program módban használható.

Befejezi egy adott file feldolgozását, törli a file-t a három file-leíró táblából. (A háttértárakon levő file-ok tartalma természetesen nem vész el.) A billentyűzetre és a képernyőre megnyitott file-okkal a táblákból való törlésen kívül semmi egyéb nem történik. A többi file esetében a tevékenység attól függ, hogy milyen utasítással nyitottuk meg a file-t.

Szintaxis: CLOSE <aritmetikai kifejezés>

Példák:

```
(i)   OPEN 4,4 : PRINT#4,"HOGY VAGY?": CLOSE4
(ii)  OPEN 1,1,1,"FILE":PRINT#1,"HOGY VAGY?":CLOSE1
```

```
(iii) PRINT#4 :CLOSE 4: REM LEZARJA A NYOMTATOT CMD UTAN
(iv)  CLOSE 1,2,3,"$": REM UGYANAZ, MINT CLOSE 1
```

Az (i) példában megnyitjuk a nyomtatót, kiírjuk a "HOGY VAGY?" sztringet, majd a file-t lezárjuk. A (ii) példa ugyanezt a sztringet írja ki a szalagon éppen megnyitott "FILE" nevű file-ba. A (iii) példa mutatja, hogy CMD használata után hogyan kell egy file-t lezárni.

A (iv) példa a C-64 BASIC egy furcsaságára utal: a CLOSE utasítás szintaxisát ugyanaz a rutin ellenőrzi, mint az OPEN-ét. Ezért a CLOSE-ban szerepelhetnek további paraméterek, de azok semmire sem jók. Logikai hibát okozhat viszont a CLOSE 1,2 alak. Ez ugyanis **csak** az 1-es logikai file-t zárja le!

Hibalehetőségek: Elsősorban írásra megnyitott lemezes file-ok esetén lényeges, hogy ezeket a file-okat lezárjuk. Ha ezt nem tesszük meg, az utolsó rekord sáv/szektor mutatója nem a megfelelő helyre mutat, s előbb vagy utóbb két file egymásra íródik. Programhiba következtében gyakran maradnak nyitva file-ok. Ilyenkor célszerű a file-okat paranccsal lezárni. Előfordulhat, hogy a 'megnyitott file-ok száma' mutató ($98, 152) nulla lesz (pl. szerkesztés után). A file-t még ekkor is megkísérelhetjük lezárni a POKE 152,10 : CLOSE <file-szám> parancs kiadásával. Ha a táblák is törlődtek, akkor adjuk ki az OPEN 15,8,15,"I":CLOSE 15 parancsot. Amennyiben ez sem zárja le, akkor nincs mit tenni.

CLR

Rövidítés: cL *Token*: $9C(156)

Mód: mind parancs -, mind program módban használható.

Valamennyi egyszerű és tömb változót törli a memóriából, magát a BASIC programot azonban változatlanul hagyja. A CLR törli a GOSUB és FOR utasításokhoz tartozó hivatkozásokat is, és végrehajt egy RESTORE utasítást!

Szintaxis: **CLR**

Példák:

```
(i)   CLR
(ii)  CLR: PRINT"A VALTOZOKAT TOROLTUK"
(iii) POKE 55,0: POKE 56,48: CLR: REM A MUNKATERULET VEGE $3000
(iv)  100 CLR: DIM AX(5,67)
```

A példák önmagukért beszélnek. Bármi is volt a változók értéke eddig, a programsor vagy a parancs végrehajtása után csak kezdőértékükkel használhatjuk őket. Ez számok esetén 0, sztringek esetén pedig az üres sztring (""). A (iii) példában a BASIC munkaterület végét jelző byte-okat átállítottuk. Ilyen esetben a CLR mindenképp szükséges, mert csak ezután veszi az interpreter figyelembe az új munkaterületet.

A (iv) példában az egyszer már deklarált AX tömböt akarjuk újra deklarálni. Ehhez azonban a változókat törölnünk kell. Ha ezt nem tesszük, akkor ?REDIM'D ARRAY hibaüzenetet kapunk.

Hibalehetőségek: Az utasítás - helyes szintaxis esetén - mindig hiba nélkül végrehajtódik. Problémát csak az okozhat, hogy egyes változók értékei - amelyekre később szükség lenne - elvesznek. A CLR semlegesítése igen nehéz, ezért óvatosan bánjunk vele!

CMD

Rövidítés: cM *Token*: $9D(157)

Mód: mind parancs -, mind program módban használható.

A CMD utasítás két funkciót egyesít:

1/ Segítségével módosíthatjuk az elsődleges output file-t;

2/ A CMD utasítást különböző értékek kiírására éppúgy használhatjuk, mint a PRINT utasítást. Az egyes értékeket elválasztó, illetve a sor végén szereplő , és ; jeleknek ugyanaz a hatása.

Szintaxis: **CMD** <aritmetikai kifejezés> {,<nyomtatási kép>}

Példák:

```
(i)    10 OPEN 5,4
       20 CMD 5: REM A TOVABBI OUTPUT A NYOMTATORA MEGY.
       20 CMD 5,;: REM A TOVABBI OUTPUT A NYOMTATORA MEGY
                                                  CRLF NELKUL
       20 CMD 5,"HELLO"
       PRINT#5:CMD PRINT : REM SZINTAKTIKUSAN HELYES

(ii)   10 OPEN 4,4 : CMD 4 : INPUT"NEV";K$

(iii)  10 OPEN 4,4: CMD 4
       20 PRINT "EREDMENYEK"
       30 GET K$:IF K$=""THEN GOTO 30
       40 PRINT X,Y,Z
```

```
(iv)   PRINT#4:CLOSE4:END
(v)    OPEN 1,4: CMD 1: LIST
       PRINT#1: CLOSE 1
```

Az (i) alatt felsorolt példák önmagukért beszélnek.

Ha a CMD 5, "HELLO" utasítást összehasonlítjuk a PRINT#5, "HELLO" utasítással, rögtön világossá válik, mennyire hasonlóak. Zavaró azonban, hogy míg a CMD 5 parancs hatására a nyomtató *'hallgató'* állapotba kerül, addig a PRINT#5 hatása ennek épp az ellenkezője; az output eszköz *megszűnik 'hallagtó'* lenni. Ez jól látszik a nyomtató esetén. CMD utasítás után a 'READY.' nem jelenik meg a képernyőn, ehhez még egy üres PRINT# utasítást is ki kell adni.

A (ii) példa a CMD egy igen szerencsétlen hatását mutatja be; az input promt a CMD-ben specifikált file-ba íródik, nem a képernyőre.

Hasonló a célja a (iii) példának is. A GET hatására az elsődleges output file újból a képernyő lesz, és az X,Y,Z értéke a *képernyőre* kerül.A (iv) példa mutatja, hogyan kell a CMD-ben használt file-t helyesen lezárni.

Megjegyezzük, hogy programhiba esetén a CMD hatása megszűnik, és az elsődleges output egység újból a képernyő lesz. A fenti hatások miatt a CMD utasítást főleg listázásra célszerű használni, úgy ahogy azt az (v) példában látjuk. Eredmények kiírására a PRINT# utasítás használata egyszerűbb.

Hibalehetőségek: A példák során már utaltunk a CMD utasítás néhány furcsaságára. Célszerű csak listázásra használni.

CONT

Rövidítés: cO *Token*: $9A(154)

Mód: csak parancs módban használható.

A program továbbindítására szolgál, miután a program futása egy STOP vagy END végrehajtása során, vagy a <STOP> billentyű megnyomása miatt megszakadt.

Szintaxis: CONT

A BASIC program futása közben, a program végrehajtását ellenőrző ciklus beállít bizonyos mutatókat,amelyek segítségével az inter-

preter a CONT utasítást végrehajtja. Ezek a következők:

($39) = a feldolgozás alatt álló BASIC sor sorszáma,
($3B) = az utoljára végrehajtott BASIC sor sorszáma,
($3D) = a következő utasítás mutatója (CONT-hoz).

A CONT utasítás megléte természetesen lassítja a program futását. A program megállása után a változókat a PRINT utasítással kiírathatjuk, módosíthatjuk értéküket, és a CONT még ezután is végrehajtható. Néhány esetben (pl. CLR, NEW, ?SYNTAX ERROR, illetve a program átszerkesztése után) a program futása már nem folytatható.

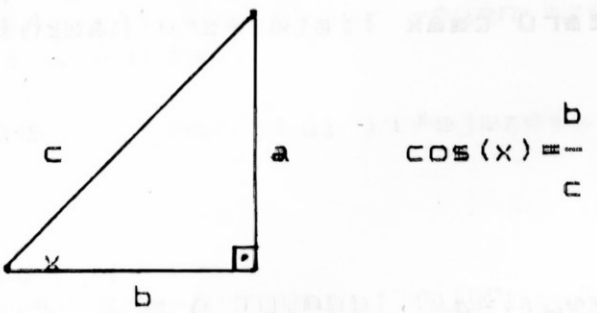
Hibalehetőségek A leírásban is jelzett esetekben a program futása nem folytatható. Ilyenkor ?CAN'T CONTINUE hibajelzést kapunk. (GOTO-val lehet kísérletezni.)

COS

Rövidítés: nincs *Token*: $BE(190)

Mód: mind parancs -, mind program módban használható.

A radiánban megadott szög koszinuszát számítja ki. Az alábbi ábra illusztrálja x és cos x viszonyát:



$$\cos(x) = \frac{b}{c}$$

Szintaxis: COS (<aritmetikai kifejezés>)

Példák:

```
(i)   PRINT COS(1.25)
(ii)  PRINT COS(90*<pi>/180): REM =0
(iii) 10 INPUT X
      20 PRINT COS(X)↑2+SIN(X)↑2
      30 GOTO 10
```

(iv) 100 X=ALFA-SIN(ALFA):Y=1-COS(ALFA)

Az első két példa önmagáért beszél. A (iii) példa a **Pitagorasz-tételt** 'ellenőrzi'. Az X értékétől függetlenül mindig nagy pontossággal 1-et kell a programnak nyomtatnia. A program futása a <STOP-RESTORE> lenyomásával állítható csak meg.

Az utolsó példában speciális trigonometrikus függvényeket számítunk ki. A (iv) egy **ciklois** X,Y koordinátáit adja meg.

Hibalehetőségek: Az aritmetikai kifejezés kiértékelése közben többfajta hibajelzést is kaphatunk. Ha a kiértékelés sikerrel befejeződött, maga a COS kiszámítása már hiba nélkül megtörténik.

DATA

Rövidítés: dA *Token*: $83(131)

Mód: csak program módban használható.

A fenti utasítás lehetővé teszi, hogy a programban számokat, sztringeket tároljunk, s ezeket a READ utasítás segítségével beolvassuk. A READ utasítások abban a sorrendben olvassák be ezeket az adatokat, ahogy azok a DATA utasításokban egymást követik.

Szintaxis: **DATA** <konstanslista>

A <konstanslista> elemei egymástól vesszővel (,) elválasztott konstansok. Mind szöveg- (sztring-), mind számkonstansok szerepelhetnek a listában. A szövegkonstansokat nem kell idézőjelek közé tenni, kivéve ha vesszőt (,) vagy grafikus jeleket tartalmaznak.

Példák:

```
(i)   151 DATA"KOVACS PETER,1957","NAGY ISTVAN,1964"
      152 DATA"ERDOS ILONA,1965"
(ii)  1055 DATA GEPI KOD,120,169,46,133,96
(iii) 2000 DATA 1,1,2,2,3,0: REM MUVELETEK RANGJA
      2010 DATA 0,1,2,3,4,5,6,7,8,9: REM ERVENYES SZAMJEGYEK
      2020 DATA ,+-,*,/,↑: REM MUVELETI JELEK
```

Az első példa szövegkonstansok DATA-ban való tárolását mutatja. A sztringekben szereplő vesszők miatt az idézőjeleket nem hagyhatjuk el. A FOR I = 1 TO 3 : READ X$: ? X$: NEXT utasítással

mind a három szövegkonstans kiíratható. A második példa illusztrálja, hogyan lehet bizonyos adatokat megtalálni a DATA-k közt. Addig hajtjuk végre a READ X$ utasitást, amig X$="GEPI KOD" nem teljesül. A "GEPI KOD" sztringben nem szerepel vessző, ezért az idézőjeleket elhagyhattuk. A (iii) példa mutatja, hogy a DATA-kban célszerű strukturáltan elhelyezni az adatokat, hogy könnyebben cserélhetők legyenek.

A DATA utasítás végrehajtása az utasítás kihagyását jelenti, azzal a különbséggel, hogy nem az egész programsor marad ki (mint a REM esetén), hanem csak az utasítás. A rutin megkeresi a következő kettőspontot, vagy a programsor végét, és onnan folytatja a program végrehajtását. Tekintsük a következő példát:

```
10 X=2
20 DATA 3: X=5
30 PRINT X
```

A program a 20. sorban átlépi a DATA-t és az X=5 utasítást hajtja végre. Ennek megfelelően a 30. sorban az 5 íródik ki.

Hibalehetőségek: A READ utasítás néhány rutinja megegyezik az INPUT utasítás hasonló rutinjaival. Emiatt a nem siftelt, sztring elején álló szóközök és bizonyos grafikus jelek elvesznek. ?SYNTAX ERROR hibaüzenet egy DATA utasítással kapcsolatban általában a hozzá tartozó READ utasítás hibáját jelenti. READ X$ sohasem okozhat problémát (kivéve, ha nincs több adat). READ X esetén nem mindegy, hogy mi a következő adat. Felesleges vessző is okozhat problémát. A DATA 1,2,3, utasítás négy konstanst tartalmaz, a negyedik egy üres sztring ("").

DEF FN

Rövidítés: dE (FN-nek nincs rövidítése) *Token*: DEF $96(150)
FN $A5(165)

Mód: csak program módban használható.

A DEF FN utasítás segítségével egyváltozós aritmetikai függvényeket definiálhatunk, amelyekre azután az FN segítségével hivatkozhatunk. A definicióban szerepel a függvény neve és a változója.

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

A <név> **a függvény neve**, a <változó> az **operandusa**. A függvényt definiáló aritmetikai kifejezés legfeljebb egy BASIC sornyi

lehet. A függvény definiálása után az

FN <név> (<aritmetikai kif.>)

alakban használhatjuk a függvényt. Mindkét változó csak **valós** lehet.

Példák:

```
(i)    10 A=3: B=4: C=-5
       20 DEF FN F(X)=A*X*X+B*X+C
       30 FOR I=0 TO 2 STEP .1
       40 PRINT X, FN F(I)
       50 NEXT I

(ii)   300 DEF FN DEEK(X)=PEEK(X)+256*PEEK(X+1)
(iii)  400 DEF FN MIN(X)=(A+B)/2-ABS(A-B)/2
       410 DEF FN MIN(X)=-(A>B)*B-(B>=A)*A
```

Első példánk egy másodfokú függvényt definiál, majd annak helyettesítési értékeit számítja ki a 0-2 intervallumban.

(ii) egy '**dupla pontosságú** PEEK'-et definiál. Ha két egymás utáni byte egy cím alsó illetve felső byte-ját tárolja (ebben a sorrendben), akkor az FN DEEK(X) függvény ennek a címnek az értékét számítja ki.

A harmadik példában két megoldást láthatunk a *minimum függvény* kiszámítására. Ez nem igazi függvény, hiszen az X csak álváltozó (dummy variable), a szintaxis kedvéért szerepel. Az A és B értéket a függvény kiszámítása előtt meg kell adni:

```
A=12: B=24: PRINT FN MIN(X)
```

Az X változó helyén így tetszőleges aritmetikai kifejezés szerepelhet: FN MIN (0), FN MIN (X*Y-10) értéke megegyezik.

A függvénydefiníciók további függvénydefiníciókat is tartalmazhatnak. Például

```
123 DEF FN EX(X)=1+X+X↑2/2+X↑3/2/3+FN E(X)
124 DEF FN E(X)=X↑4/2/3/4+X↑5/2/3/4/5+X↑6/1/2/3/4/5/6
```

megengedett.

A függvények, hasonlóan más változókhoz, újra definiálhatók:

```
175 DEF FN Y(X)=Y: DEF FN Y(X)=X
```

helyes.

Megjegyezzük, hogy a függvény kiértékelése közben a függvény definíciójában szereplő változó értéke nem változik meg:

```
10 X=77
20 DEF FN Q(X)=X*X-5*X+6
30 PRINT FN Q(3),X
```

A 30. sorban X értékeként nem 3, hanem 77 íródik ki. A függvény kiértékelésekor X-et ideiglenesen tárolja az interpreter, majd visszaállítja az eredeti értékét.

Függvényeket csak program futása közben definiálhatunk. A már definiált függvények azonban a program megállása után parancs módban is használhatók. Írjuk be és futtassuk le a következő egysoros programot:

```
10 DEF FN CS(X)=COS(X*(pi)/180)
```

Ezt követően a CS függvény segítségével fokokban adott szög koszinuszát is kiszámíthatjuk: PRINT FN CS(90) például 0-át ad eredményül.

Hibalehetőségek: Ha a DEF után az egyenlőségig valamilyen hibát követünk el, akkor ?SYNTAX ERROR hibajelzést kapunk. Ha az aritmetikai kifejezés kiértékelése közben történik hiba, akkor a hibajelzést a megfelelő FN-t tartalmazó sorban kapjuk. Ha az FN utasítást a hozzá tartozó DEF FN utasítás előtt hívjuk, akkor ?UNDEF'D FUNCTION ERROR hibajelzést kapunk. (A CLR a függvény-definíciókat is törli!)

DIM

Rövidítés: dI *Token*: $86(134)

Mód: mind parancs -, mind program módban használható.

Az utasítás a DIM-et követően megadott nevű, típusú, dimenziójú és méretű tömbelemeknek helyet foglal a memóriában. A tömb neve, típusa tetszőleges lehet, az egyes indexek értéke 0-tól a DIM utasításban megadottig terjedhet, de legfeljebb 32767 lehet. A tömbdeklaráció csak akkor válik érvényessé, ha a program végrehajtja a DIM utasítást.

Szintaxis: **DIM** <tömbváltozó lista>

A <tömbváltozó lista> egymástól vesszővel elválasztott tömbelemekből áll.

Példák:

```
(i)   10 DIM A%(12),B1%(100,12),B(210),KJ$(320)
(ii)  410 INPUT "N=";N: DIM TOMB(N*N+1)
(iii) 3000 FOR J=1 TO 3: READ A$: MES$(J)=A$:NEXT J
(iv)  DIM XF(23)
```

A DIM utasítás használata egyszerű. A problémát inkább az okozza, hogy nagyobb adatmennyiséget a programjaink által kezelhető módon már nehéz megszervezni. A tömbök **dinamikusan definiálhatók**; erre példa a fenti (ii) példa, amiben a TOMB dimenzióját az N*N+1 kifejezés adja meg. A (iii) példa **implicit tömbdefiníciót** tartalmaz. Ez azt jelenti, hogy tömbváltozókat a tömbök definiálása nélkül is lehet használni, feltéve, hogy indexeik nem haladják meg a 10-et. A MES$(1) kiértékelésekor a memóriában a tömb elhelyezésére is sor kerül. A hatás ekvivalens a DIM MES$(10) utasítással.

A legkisebb index értéke 0 lehet. Több számítógép nem teszi lehetővé ennek az indexnek a használatát. Ugyanakkor egy tömb 0. elemét különféle számításokban jól fel lehet használni. Például a

```
10 DIM A(20): FOR X=1 TO 20
20 INPUT N: A(X)=N: A(0)=A(0)+N: NEXT
```

programrész 20 elemet olvas be az A(X) tömbbe, összegüket pedig elhelyezi az A(0) tömbelemben.

A CLR utasítás mind az egyszerű, mind a tömbváltozókat törli. A tömbváltozók **külön törlése** a következő paranccsal (ami programból is használható) egyszerűen elvégezhető:

```
POKE 49,PEEK(47): POKE 50,PEEK(48)
```

A 3.2 paragrafusban részletesen leírtuk, hogy egy tömb elemei hogyan helyezkednek el a memóriában. A következő BASIC parancscsal egyszerűen kiírathatjuk a tömb által foglalt helyet:

```
F=FRE(0): DIM SI(14,20): PRINT F-FRE(0)
```

Hibalehetőségek: Az indexhatárok kiértékelése közben ?OUT OF MEMORY ERROR hibajelzést kaphatunk, ha nincs elegendő hely a tömb elhelyezésére. Ugyanazt a tömböt csak egyszer definiálhatjuk egy programban. A második kísérletre ?REDIM'D ARRAY ERROR hibaüzenetet küldi. Ugyanezt a hibajelzést kapjuk abban az esetben is, ha implicit módon definiáltuk a tömböt, s azt követően újra definiáltuk:

```
X$(0)="ABCD": DIM X$(22)
```

A leggyakoribb hiba, ha szintaktikus vagy tervezési hiba miatt egy tömbváltozó indexei nem esnek a megfelelő intervallumba. Ilyenkor ?BAD SUBSCRIPT ERROR hibaüzenetet kapunk.

END

Rövidítés: eN *Token*: $80(128)

Mód: mind parancs -, mind program módban használható.

Az END végrehajtása a program futásának azonnali befejezését jelenti; az interpreter a 'READY.' üzenet kiírásával visszaadja a vezérlést a képernyő szerkesztőnek. A CONT parancs kiadásával folytathatjuk a program futását.

Szintaxis: **END**

Példák:

```
(i)   100 PRINT#4: CLOSE 4: GOSUB 200: END
(ii)  130 IF SAV<>TR THEN PRINT "** NEM JO SAVSZAM **":END
(iii) 2000 GOSUB 1000:END:GOSUB 2000:END
(iv)  59999 END
      60000 <utasítások>
```

Négy példánkban az END különböző célú alkalmazásait igyekeztünk bemutatni. Az első példa a nyomtató lezárása után meghívja a 200 alatti alprogramot, majd befejezi a program futását. A második példában az END egy hibaág végén jelzi az interpreternek, hogy a program futását fel kell függeszteni.

A harmadik példa nem egy kész program részlete. A program ellenőrzésére két töréspontot is elhelyeztünk. A program megállása, a változók ellenőrzése után a CONT parancscsal folytathatjuk a program futását. Az utolsó példában a 60000 sorszámmal kezdődő alprogramok elé egy END utasítást tettünk, nehogy 'rácsorogjon' a program vezérlése az alprogramokra.

Hibalehetőségek: Nincs. (Leszámítva azt az esetet, hogy vezérlési hiba folytán rossz helyen áll meg a program.) Az END végrehajtásakor az interpreter nem írja ki a BREAK IN ... üzenetet.

EXP

Rövidítés: eX *Token*: $BD(189)

Mód: mind parancs -, mind program módban használható.

Az e (természetes) alapú exponenciális függvény kiszámítására szolgál (e=2.7182818..). x értékének megközelítőleg a (-88,88) intervallumba kell esni.

Szintaxis: EXP (<aritmetikai kifejezés>)

Példák:

```
(i)   PRINT EXP(10): REM=22026.4658
(ii)  Y=EXP(1): REM Y=2.71828183.
(iii) PRINT EXP(LOG(X)): REM=X
(iv)  100 FOR N=0 TO 10: P(N)=EXP(-M)/ FN FACT(N): NEXT N
(v)   50 NT=NE*EXP(-B*EXP(-K*T))
```

Az SQR függvényhez hasonlóan az EXP függvény is a hatványozás speciális esete: EXP(Q)=2.7182818↑Q. Sok esetben van szükség rá, ezért kényelmesebb, ha külön is használhatjuk.

Első két példánk az EXP közvetlen hatását mutatja. (ii) azt szemlélteti, hogy az EXP függvény a LOG függvény inverze, leszámítva természetesen a kerekítési hibákat. Az utolsó két képlet valószínűségi, illetve statisztikai programokban használható. Az első a Poisson-elosztást számítja ki, felhasználva az előzetesen már definiált FACT(N) faktoriális számító függvényt. Az utolsó képletet egy un. logisztikus növekedési függvényt definiál; a népesedési folyamatok modellezésében lehet felhasználni.

Magát az e számot sokféleképpen lehet definiálni. Egyik definíció sem elemi, lévén az e irracionális szám. e↑x-et legegyszerűbben az 1 + x + x/2! + x/3! + ...hatványsorral lehet kiszámítani. Az e↑x függvény deriváltja önmaga.

Hibalehetőségek: Ha az argumentum értéke túl nagy, akkor ?OWERFLOW ERROR hibajelzést kapunk. Ha az argumantum értéke túl kicsi, akkor hibajelzés nélkül, a gépi nullával számol tovább az interpreter.

FOR..TO..{STEP..}

Rövidítések: fO *Tokenek:* FOR $81(129)
stE TO $A4(164)
STEP $A9(169)

Mód: mind parancs -, mind program módban használható.

Lehetővé teszi a FOR..TO..{STEP..} és a megfelelő NEXT utasítások közti programrész többszöri végrehajtását. A NEXT utasítás végrehajtásakor az interpreter ellenőrzi a változó értékét, megkeresi a hozzá tartozó FOR..TO..{STEP..} utasítást, a STEP részben definiált értékkel megnöveli a ciklusváltozó értékét, és ellenőrzi, hogy beleesik-e a TO-ban definiált értéktartományba. Ha igen, a vezérlés a FOR..TO..{STEP..} utasítást követő utasításra adódik; ha nem a NEXT utáni elsőre.

Szintaxis:
FOR<változó>=<arit. kif.> **TO** <arit. kif.> {**STEP**<arit. kif.>}

A FOR utáni változót **ciklusváltozónak**, az egyenlőségjel utáni kifejezés értékét **kezdőértéknek**, a TO utáni kifejezés értékét **végértéknek**, a STEP utáni kifejezés értékét pedig **növekménynek** hívjuk. Ha a STEP hiányzik, az interpreter a növekményt 1-nek tekinti. A FOR és a NEXT közti programsorokat **ciklusmagnak** hívjuk.

Példák:

```
(i)   FOR J=1 TO 1000: PRINT ".";: NEXT
      FOR J=1 TO 1000: PRINT J;: NEXT
      FOR XJ=1 TO 1000: NEXT
```

Az (i) alatt felsorolt példák a legegyszerűbb ciklusokat mutatják be. A ciklusváltozó értékét a cikluson belül nem változtatjuk. Valahányszor a NEXT végrehajtódik, J, illetve XJ értéke 1-gyel nő, ez lesz ugyanis (a STEP utasítás hiányában) a növekmény értéke. Amikor a ciklusból kilépünk, a J, illetve XJ értéke 1001.

```
(ii)  FOR I=0 TO 255:POKE 1024+I,I:POKE 55296+I,14:NEXTI
(iii) FOR I=255 TO 0 STEP -1:POKE 1024+I+1,PEEK(1024+I):NEXT I
```

A (ii) parancs a képernyő memóriába írja be a számokat 0-tól 255-ig. Ennek hatására a képernyő tetején az összes lehetséges karakter kiíródik. A ciklusban használt változó mutatja, hogyan lehet biztosítani, hogy a ciklus ismétlődő végrehajtása során egyes mennyiségek értéke más és más legyen.

A (iii) parancssor a képernyő memória tartalmát tolja el egy pozícióval jobbra. Az egyes karakterek másolását hátulról előre kell végrehajtanunk, különben a képernyőn csak az első karakter maradna. Ezért kell használnunk a STEP -1 utasítást.

A ciklusok használata nem szokott különösebb problémát okozni, mégis néhány megjegyzést teszünk helyes használatukról. Először a szintaxisról. A ciklusváltozónak mindig **egyszerű valós** változónak kell lennie. FOR X%=1 TO 9 és FOR X(0)=0 TO 2 STEP -1 egyaránt helytelen. A ciklus a ciklusváltozó alsó és felső határaival is lefut. Például a FOR I=0 TO 5 ciklusutasitás az I=0,1,2,3,4,5 értékekkel hajtódik végre.

Ha a program a ciklusváltozó és a növekmény értékét pontosan tárolta, akkor a ciklus - néhány extrém esettől eltekintve - annyiszor fut, ahányszor matematikailag futnia kell. Általában az egész változók értékeit és a decimális törteket az interpreter pontosan tárolja. Ennek megfelelően a

```
FOR X=1 TO 1000 illetve a
FOR X=1 TO 10 STEP .0125
```

utasitások esetén a ciklusmag 1000-szer illetve 800-szor kerül végrehajtásra. A FOR X=1 TO 1000 STEP 1/3 utasitással már probléma lehet, célszerü a FOR X=1 TO 1000.1 STEP 1/3 utasitással helyettesiteni.

Megjegyzendő, hogy a ciklusmag egyszer akkor is végrehajtódik, ha a növekmény > 0, és a végérték kisebb mint a kezdőérték.

A BASIC interpreter megengedi a ciklusok egymásba ágyazását is. A következő program-séma illusztrálja ezt:

```
FOR X=X1 TO X2
 FOR Y=Y1 TO Y2
  FOR Z=Z1 TO Z2
      ...
    <ciklusmag>
      ...
  NEXT Z
 NEXT Y
NEXT X
```

A FOR utasitás végrehajtásakor 18 byte-nyi információ kerül a verembe. A FOR utasitáson kivül a GOSUB is használja a vermet, együttes használatuknak igy határt szab a verem nagysága (256 byte). Az interpreter minden egyes FOR utasitás végrehajtásakor ellenőrzi, hogy ez a ciklusváltozó szerepel-e a veremben. Ha igen, a 'verem teteje' mutató erre a változóra fog mutatni, a verem 'tetejének' többi része elvész.

A ciklusok tetszőleges strukturában egymásba ágyazhatók:

```
10 FOR X=XA TO XB: FOR Y=YA TO YB: NEXT Y
20 FOR A=AA TO AB: FOR C=CA TO CB: NEXT C,A,X
```

Az egyes programnyelvekben megtalálható *DO <utasítások> UNTIL <teszt>* konstrukció a következő ciklussal érhető el:

```
FOR J=-1 TO 0
    <utasítások>
    J=NOT <teszt>
NEXT J
```

A *DO <utasítások> WHILE <teszt>* konstrukció a következő ciklussal érhető el:

```
FOR J=-1 TO 0: <utasítások>: J=<teszt>: NEXT J
```

Hibalehetőségek: Az aritmetikai kifejezések kiértékelése számos hibát eredményezhet. A ciklusutasítás használatában már kódolási hibák is előfordulhatnak. Ezek közül a leggyakoribbak:

(i) A negatív növekmény lemarad.
(ii) NEXT hiányzik.
(iii) Az egymásba ágyazott ciklusváltozók felcserélődnek.
(iv) Ugyanazt a változót két egymásba ágyazott ciklusban használjuk.
(v) Egy RETURN nélküli GOSUB könnyen okozhat ?NEXT WITHOUT FOR hibaüzenetet.
(vi) Ugyanezt a hibaüzenetet kapjuk abban az esetben, ha a NEXT utasításban egy nem létező ciklusváltozót használunk.
(vii) Ha egész, sztring vagy tömbváltozót használunk ciklusváltozónak, ?SYNTAX ERROR hibaüzenetet kapunk.

FRE

Rövidítés: fR *Token:* $B8(184)

Mód: mind parancs -, mind program módban használható.

Kiszámítja, hány **szabad byte** található a BASIC munkaterületen, azaz mennyi a különbség a tömbök vége és a sztringek eleje közt. Ezt megelőzően azonban egy ún. **szemétgyűjtést** végez.

Szintaxis: FRE(<kifejezés>)

A FRE alakját tekintve függvény, de az argumentum kiszámítására vagy ellenőrzésére nem kerül sor. A FRE valójában egy nulla-változós függvény. Általában PRINT FRE(0) vagy F=FRE(0) alakban használjuk. A FRE(X$), FRE(A+X%), FRE(X) kifejezések szintaktikusan helyesek.

Példák:

```
(i)  PRINT "SZABAD BYTE-OK SZAMA=";FRE(0)
(ii) X=FRE(0):DIM X%(278):PRINT X-FRE(0)
```

Az első példa egyszerüen kiirja a szabad memóriakapacitást. A második esetben a FRE függvényt annak kiszámitására használjuk, hogy egy sztring tömb definiálása a memóriában hány byte-ot foglalt le.

A BASIC munkaterületen a program egyes részei a következőképpen helyezkednek el:

BASIC program	egyszerű változók	tömbök	szabad memória	sztringek

eleje ← BASIC munkaterület → vége

Az alábbi példa egy beolvasó rutin, amely 20 egymás utáni karaktert olvas be a billentyüzetről:

```
10 FOR I=1 TO 20
20 GET X$: IF X$="" GOTO 20
30 I$=I$+X$: NEXT
40 X=FRE(0)
```

Az X$ minden esetben 1 byte hosszuságu és I$ minden egyes kiszámitása I$ számára a sztringek közt ujabb helyet foglal le. A 30. sor I-ik végrehajtásakor a memória I*(I+3)/2 byte-ját foglalják le X$ és I$ előző értékei. A 20 karakter beolvasása a memóriában összesen 230 byte-ot foglal el. A 40. sorban a FRE(0) kiszámitásakor X$ és I$ felesleges darabjai 'eltünnek'.

Hibalehetőségek: Nincs. Alkalomadtán azonban 10 percig is eltarthat a 'szemétgyüjtés'.

GET és GET#

Rövidítés: gE és gE# *Token*: $A1(161)

Mód: csak program módban használható.

Az utasitásban megadott input perifériáról egyetlen byte-ot olvas be. Billentyüzet esetén, ha egyetlen karakter sincs a billentyüzet-pufferben, az eljárás az üres sztringgel tér vissza.

Szintaxis: GET(# <aritmetikai kifejezés>,) <változólista>

A <változólista> legalább egy elemet kell, hogy tartalmazzon; elemei általában sztring változók. Az aritmetikai kifejezés az input file logikai file számát definiálja. A GET és a # jel közt nem lehet szóköz.

Példák:

```
(i)  5 GET X$:IF X$="" THEN GOTO 5
     10 PRINT" ";X$;"  ";ASC(X$):GOTO 5

(ii) 200 GET A$,B$,C$:PRINT A$+B$+C$:GOTO 200
```

Az első példánk segitségével kipróbálhatjuk, hogy az egyes billentyüket a GET hogyan is olvassa be. Az 5-ös sorszámu utasitásban addig várunk, mig legalább egy karakter nem kerül a billentyüzet pufferbe (**dinamikus megállás**), azután ezt 'visszairjuk' a képernyőre és kiirjuk az ASCII kódját is. érdemes kipróbálni a <RETURN>, <DEL>, <CTRL-RVSON> stb. billentyüket!

A (ii) példában három, a billentyüzet pufferbe kerülő, legfeljebb 3 karakternyi információt olvassuk ki végtelen ciklusban és irjuk ki a képernyőre. Jól érzékelhető, hogy a GET nem várakozik a billentyü leütésére, hanem a billentyüpuffer esetleges tartalmát üriti ki.

Megjegyezzük, hogy az 50 GET A utasitás szintaktikusan helyes. Ha azonban a byte értéke nem a 0-9 intervallumba esik, akkor egy ?SYNTAX ERROR hibaüzenetet kapunk. Ha a következő byte ; : vagy ,, akkor ?EXTRA IGNORED hibaüzenetet kapunk, és A értéke 0 lesz. Célszerübb a GET A$ utasitás alkalmazása, majd azt követően annak ellenőrzése, hogy számjegyet olvastunk-e be.

Billentyüzet puffer A GET utasitás az input puffer első karakterét olvassa be. Ezeket a karaktereket a hardver megszakitó rutin helyezi el a pufferba. (Másodpercenként kb. 50-szer hajtódik végre.) A billentyüzet-puffer a $277-$280 (631-640) cimeken található. A $C6 (198) cimet használja az

interpreter a pufferben levő karakterek (byte-ok) számának tárolására. A puffert 'kiürithetjük' a POKE 198,0 utasitással, vagy a

```
10 GET X$ : IF X$>"" THEN 10
```

utasitással. A billentyüzet-puffer léte a következő program végrehajtásával érzékeltethető:

```
10 FOR J=0 TO 8000: NEXT
20 FOR J=0 TO 20: GET X$: PRINT X$: NEXT J
```

A RUN parancs kiadása után - amig az első sor fut - billentyüzzünk be néhány karaktert. A képernyőn csak az elsőként beirt tizet látjuk viszont.

A GET utasitás - szemben az INPUT utasitással - byte-onként olvassa be az adatokat, ezért alkalmas ismeretlen strukturáju rekordok olvasására is. Lemezes file-ok esetén ezt legegyszerübben az alábbi ciklussal tehetjük meg:

(iii)
```
1000 GET#3,X$:IF ASC(X$)=13 GOTO 3000
1010 IN$=IN$+X$:REM SZTRING OLVASASA
1020 GOTO 1000
```

Szalagos file Abban az esetben, ha a GET# utasitásban definiált logikai filo szám egy kazettás file-ra utal, a karaktert a kazetta pufferből olvassa a GET# rutin, nem pedig a billentyüzet pufferből. Ha a kazetta puffert már teljes egészében beolvastuk, a program futását az interpreter felfüggeszti, és a következő rekordot betölti a szalagról. A 'szalag vége' jelnek egy nulla byte felel meg. Ennek olvasása az ST állapotjelző byte-ot 64-re állitja. Ha ST értékét nem ellenőrizzük, akkor a következő GET# (vagy bármilyen, a szalagra vonatkozó input utasitás) ST értékét nullázza és a kazettáról további adatok olvashatók. Ugyanez áll **lemezes file-okra** is.

Hibalehetőségek: A GET# utasitásban szereplő logikai file-t előbb meg kell nyitnunk. Problémát szokott okozni, ha az ST vizsgálata elmarad, és igy esetleg az EOF jel után is tovább olvassuk a file-t. Ekkor a lemezegység teljes egészében 'lemerevedhet'.

GO

Rövidítés: nincs *Token*: $CB(203)

Mód: mind parancs -, mind program módban használható.

A GO utasitás lehetővé teszi a GOTO utasitás GO TO alakban való irását.

Szintaxis: **GO** <szóközök> **TO** <sorszám>

Hibalehetőségek: Ugyanazok, mint a GOTO utasitásnál. A GOTO utasitás GO TO alakban való irása lassitja a program futását, ezért inkább az egybeirt alak használatát javasoljuk.

GOSUB

Rövidítés: goS *Token*: $8D(141)

Mód: mind parancs -, mind program módban használható.

Végrehajtja a GOSUB után megadott sorszámú programsorban kezdődő **alprogram**ot. Ez azt jelenti, hogy feltétel nélküli vezérlésátadás jön létre a GOSUB utasításban megadott számú sorra. Amikor ezt követően a legközelebbi RETURN utasításhoz ér az interpreter, a vezérlés visszakerül a GOSUB utasitást követő első utasításra.

Szintaxis: **GOSUB** <sorszám>

A <sorszám> csak előjel nélküli egész konstans lehet.

Példák:

```
(i)   FOR V=0 TO 24: FOR H=0 TO 39: GOSUB 600: NEXT H,V
      600 <utasitasok>

(ii)  100 E=0: FOR I=1 TO LEN(S$)
      110 IF MID$(S$,I,1)=":" THEN E=E+1
      115 NEXT I
      120 GOSUB 6000
      125 <utasitások>
          ...
      6000 IF E=0 THEN A$=UZEN$(0)
      6010 IF E>0 THEN A$=UZEN$(1)
```

```
      6020 PRINT"<CLR>";A$: FOR I=1 TO 2000: NEXT
      6030 RETURN

(iii) 5000 GOSUB 5010
      5010 REM *** CSIPOGO ***
      5020 <csipogo program utasitasai>
```

Első példánk azt mutatja, hogy hogyan használható a GOSUB utasítás parancs módban. A 600-as sorszámmal kezdődő (a példában konkrétan nem szereplő) szubrutin a kurzor helyét változtatja a képernyőn. A parancs minden lehetséges értékre ellenőrzi a rutin helyességét.

A (ii) példa egy ellenőrző rutin része. Megvizsgálja, hogy az inputként beolvasott S$ sztring tartalmaz-e kettőspontot (:). Ezután kerül sor a 6000 alatt kezdődő alprogram végrehajtására, ami a vizsgálat eredményétől függően beállítja A$ tartalmát, és kiírja a képernyőre. A program ezekután folytatódik.

Az utolsó példa egy olyan szubrutint mutat be, aminek két belépési pontja van: 5000, illetve 5010. Ha a főprogramból az alprogramot GOSUB 5000-rel hívjuk meg, akkor az 5020 alatti programrész - az 5000 alatti újbóli GOSUB miatt - kétszer hajtódik végre, a program kétszer 'csipog'. Ha a GOSUB 5010 utasítást használjuk, a program csak egyszer 'csipog'.

Még abban az esetben is, ha egy programrészt csak egy helyen akarunk a programban használni, előfordulhat, hogy célszerű alprogramként kidolgozni. Ilyenek például a következő programrész alprogramjai:

```
7000 IF S$="S" THEN GOSUB 5000: GOTO 6000
7010 IF S$="L" THEN GOSUB 5100: GOTO 6000
7020 IF S$="@" THEN GOSUB 5200: GOTO 6000
7030 IF S$="$" THEN GOSUB 5300: GOTO 6000
7040 GOTO 6000
```

Általában, ha egy ismétlődő programrész hosszabb, mint egy sor, vagy többszörösen egymásba ágyazott IF utasításokat tartalmaz, célszerű megfelelően kialakított alprogramokat használni.

A GOSUB utasítások **egymásba ágyazhatók.** Ennek azonban határt szab nemcsak a program áttekinthetősége, hanem a verem nagysága is. Ilyen esetekben ügyelnünk kell arra, hogy a RETURN utasítás mindig csak a *legutolsó* GOSUB után tér vissza, s ezért a vezérlést pontosan meg kell terveznünk. Példa erre a következő program, mely elsősorban az ilyen programok *strukturáját* mutatja be.

```
   10 GOSUB 5000
   20 <SZAMITASOK>
      ...
 5000 <SZAMITASOK>
      GOSUB 10000
      IF E=0 THEN RETURN
      ...
      <SZAMITASOK>
      E=3: RETURN
10000 <SZAMITASOK>
      IF <TESZT> THEN E=0: RETURN
      <SZAMITASOK>
      RETURN
```

Az 5000. sorban kezdődő alprogram egy további alprogramot (10000) hív meg. Ez az utóbbi, hiba esetén, az E értékét 0-ra állítja. Ebben az esetben a vezérlés azonnal visszatér a 10.sor alá. Mivel az 5000 alatti alprogram sikeres visszatérés esetén E értékét maga is beállítja, ezt a 20-as sorral kezdődő programrészben is felhasználhatjuk.

Hibalehetőségek: Ha a GOSUB utasításban megadott programsor nem létezik, akkor ?UNDEF'D STATEMENT ERROR hibajelzést kapunk. A sorszám előállítása az első nem numerikus karakterig tart. Így például GOSUB 12I0 hibajelzés nélkül végrehajtódik, és ekvivalens a GOSUB 12 utasítással. Amennyiben a veremben már nincs elég hely, ?OUT OF MEMORY ERROR hibaüzenetet kapunk.

100 GOSUB 100 például mindig ezt a hibajelzést adja, szemben a 100 GOTO 100 programrésszel, ami hibajelzés nélküli végtelen ciklust eredményez.

GOTO

Rövidítés: gO *Token*: $89(137)

Mód: mind parancs -, mind program módban használható.

Feltétel nélküli vezérlésátadás. A program a GOTO utasításban megadott számú sor végrehajtásával folytatódik.

Szintaxis: **GOTO** <sorszám>

Példák:

(i) D$="1234": GOTO 1500

```
(ii)   100 GET X$: IF X$="" GOTO 100
       110 IF X$=CHR$(13) GOTO 300
       120 <utasítások>
(iii) GO TO 2000
```

Az első példa a GOTO direkt (parancs) módban való használatát mutatja. A D$ beállítása után a program futása az 1500-as sorszámú programsorral folytatódik. Az utasítás hatása nem azonos RUN 1500-al, mert az elúbb töröl majdnem mindent, például a D$ értékét is.

A második példa egy rövid ciklust tartalmaz. Amint megnyomunk egy billentyűt, az IF feltétel hamissá válik, és a GOTO 100 utasítás helyett a következő sorra(110) kerül a vezérlés.

A (iii) példa egyszerűen illusztrálja a GO TO alak használatának lehetőségét.

Hibalehetőségek: A <sorszám> előállítása az első nem numerikus karakterig tart. Az így előállított sorszámra kísérli meg átadni a vezérlést az interpreter. Például GOTO 1X ekvivalens a GOTO 1 utasítással. Nem létezú sorszám esetén ?UNDEF'D STATEMENT ERROR hibajelzést kapunk.

Rosszul tervezett programok esetén könnyen előfordulhat, hogy nem megfelelő a vezérlés. Ilyen esetben a program strukturáját újból kell ellenőrizni (majd a szükséges változtatásokat végrehajtani).

IF...THEN...

Rövidítés: nincs *Token*: IF $8B(139)
THEN $A7(167)

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 a következő programsor.

Szintaxis:

IF <aritmetikai kifejezés> { THEN { <sorszám> / <utasítás> } / GOTO <sorszám> }

<sorszám> csak előjel nélküli konstans lehet!

Példák:

```
(i)   FOR I=1 TO 600: 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

Rövidítés: nincs *Token:* $85(133)

Mód: csak program módban használható.

Az utasítás lehetővé teszi, hogy a billentyűzetről közvetlenül **adatokat adjunk át egy futó BASIC programnak.** Az INPUT utasítás a billentyűzetről bevitt értékeket mindig **visszaírja** a képernyőre. Az eljárás a <RETURN> megnyomásával fejeződik be.

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

A <szöveg> tetszőleges karaktersorozat lehet; a <változólista> elemei egymástól vesszővel elválasztott egyszerű vagy tömbváltozók. Az opcionális <szöveg> karaktersorozat, majd azt követően egy kérdőjel jelenik meg a képernyőn, végül ezt követi a villogó kurzor. A bebillentyűzött értékek feldolgozása a következőképpen történik:

1/ Az interpreter a legtöbb karaktert az input változók részének tekint, bizonyos jeleket azonban ettől eltérően értelmez. Ilyenek elsősorban az " , : jelek. A " jel utasítja az interpretert, hogy az azt követő jeleket egy sztring részének tekintse. A <RETURN> billentyű lenyomása mindig befejezi az input sor bevitelét.

2/ Az INPUT utasítást végrehajtó rutinnak közös részei vannak a GET és a DATA utasítások hasonló részeivel, így a vesszőt (,) és a kettőspontot (:) **elválasztó jel**ként értelmezi.

3/ Az INPUT utasítás az input sort a <szöveg> és a kérdőjel (?) kinyomtatásától a logikai sor végéig (maximum 80 karakter) dolgozza fel.

Példák:

(i)
```
100 INPUT "HONAP";H$
200 INPUT "CIM (VESSZOT NEM HASZNALHAT!)";C$
300 INPUT X,Y: REM KEZDOPONT
```

A fenti példák az INPUT utasítás legegyszerűbb használatát mutatják; használata ilyenkor nem okozhat problémát. *Helytelen adatbevitel* esetén azonban furcsa dolgok történhetnek. A <HOME> lenyomása a képernyő bal felső sarkába állítja a kurzort. A <RETURN> azonnali terminálást eredményez, a <STOP> elkezd betölteni a szalagról egy új programot stb.

(ii)
```
1000 INPUT AA$ BB,C%: REM KEZDOERTEKEK
```

(iii)
```
2000 FOR J=0 TO 10:INPUT X$(J):NEXT
2010 FOR J=0 TO 10:PRINT X$(J):NEXT
```

Az 1000 sorszámú utasítás három input változót tartalmaz. A

KUTYA, 56.83,7.1 <RETURN>

helyes válasz. A program futása az AA$="KUTYA", BB=56.83, C%=7 értékekkel folytatódik. (Az egész változóra vonatkozó értékadás a szokásos kerekítési szabályok segítségével történik.) A következő válasz egy ?EXTRA IGNORED hibaüzenetet eredményez:

KUTYA,18,56.83,7.1 <RETURN>

A KUTYA <RETURN> válasz a következő sor elején két kérdőjel (??) és a kurzor kinyomását eredményezi; jelezve, hogy **további** értéket kell bevinnünk a gépbe.

Következő példáink azt mutatják, hogyan lehet az INPUT utasítást néhány egyszerű trükkel biztonságosabbá és kényelmesebbé tenni.

```
(iv) 10 INPUT "<CLR>N=";N: PRINT N
     20 INPUT "NEV=   <USPC><B><B><B>";X$:PRINT X$
(v)  30 INPUT"ALFA  2<B><B><B>";A:PRINT A
(vi) 200 POKE 198,3:POKE 631,34:POKE 632,34:POKE 633,20
     210 INPUT X$:PRINT X$
```

Az első, (iv) alatti példában az input promptban elhelyezett vezérlő karakter hatására a képernyő törlődik, s utána kéri a program N értékét.

A 20. sorban levő példában az input sztringbe a következő vezérlő karaktereket helyeztük el: <B>=<CRSR BALRA>, <USPC>=<SHIFT-szóköz>. A <RETURN> azonnali megnyomásának a hatására az X$-ba nem az előző eredmény, hanem egy egyetlen karakatert tartalmazó sztring kerül.

Az (v) példában a három <CRSR BALRA> hatására az INPUT utasítás kurzora éppen a 2 felett fog villogni. Ha ez az érték megfelel, akkor csak a <RETURN>-t kell megnyomni, magát az értéket nem kell beírni.

A 200-210-es sorokban található rutin a billentyűzet pufferbe három karaktert (""<DEL>) helyez el. A ? kiírása után a három karaktert az interpreter beolvassa. Ez azt eredményezi, hogy az inputsor már nem lehet üres, és a második " utasítja az interpretert, hogy sztringnek tekintse az inputot. így az vesszőt (,) és kettőspontot (:) is tartalmazhat. A két utasítás együtt valójában egy INPUTLINE X$ utasítást szimulál.

Hibalehetőségek: Ez a meglehetősen hosszú rutin párhuzamosan dolgozza fel az input sort és az INPUT utasításban levő változólistát. Az inputsorból előállított értékeket megkísérli

hozzárendelni a soronkövetkező input változóhoz. Ha a típus nem egyezik, akkor ?REDO FROM START üzenettel az interpreter újra végrehajtja a **teljes** INPUT utasítást. Ha az inputsor kevesebb értéket tartalmaz, mint ahány INPUT változó van, akkor a következő sor elejére két kérdőjel (??) íródik ki, és a rendszer a **hiányzó** értékek bevitelét várja. Ha az inputsor több értéket tartalmaz, mint ahány változó van az INPUT utasításban, akkor ?EXTRA IGNORED üzenetet kapunk és a program futása folytatódik.

Az INPUT utasítás hátránya, hogy egy hibás adatbevitel a bizonylatot utánzó, vagy a szerkesztett képernyőt azonnal tönkreteszi. Ezt egyetlen módon kerülhetjük el: a GET utasítás segítségével; amelyik semmit sem ír ki a képernyőre, csupán az utoljára lenyomott billentyű ASCII kódjával tér vissza. Az így kapott karaktereket magunknak kell a képernyőre kiírnunk. Ilyenre példa a következő programrész, aminek segítségével legfeljebb N hosszúságú inputsort vihetünk be, és amelyik csak számjegyet, betűt vagy vesszőt tartalmazhat. A bevitelt szokás szerint a <RETURN> lenyomása fejezi be. Szerkesztésre egyedül a <DEL> billentyű szolgál. Kurzor helyett egy nem villogó ▌ jel jelenik meg.

A bevitel befejezése után a bevitt X$-t külön fel kell dolgoznunk.

```
   5 MAX=5:GOSUB 1000:PRINT: PRINTX$:END
1000 N=0: I=1: X$="":REM X$ TAROLJA A SZTRINGET
1010 PRINT CHR$(191);:REM KURZOR
1020 GET A$: IF A$="" THEN GOTO 1020
1030 IF N>0 AND A$=CHR$(20) THEN GOTO 1200:REM <DEL>
1040 IF N=MAX THEN GOTO 1100:REM TELE
1050 IF ASC(A$)>64 AND ASC(A$)<91 THEN GOTO 1080:REM BETU
1060 IF ASC(A$)>47 AND ASC(A$)<58 THEN GOTO 1080:REM SZAMJEGY
1070 IF A$=CHR$(13) THEN RETURN
1075 GOTO 1020
1080 N=N+1: X$=X$+A$:REM KOVETKEZO KARAKTER BE
1090 PRINT CHR$(157);A$;: GOTO 1010
1100 IF A$=CHR$(13) THEN RETURN
1110 GOTO 1020
1200 N=N-1
1210 X$=MID$(X$,1,N):REM TORLES VEGREHAJTASA
1220 GOTO 1090
```

Az INPUT utasítást nem használható parancs módban, mert a végrehajtás igénybeveszi az input puffert. Ha mégis megkíséreljük, akkor ?ILLEGAL DIRECT ERROR hibaüzenetet kapunk.

INPUT#

Rövidítés: iN (a # már nem kell!) *Token*: $84(132)

Mód: csak program módban használható.

Lehetővé teszi a háttértárakon rögzített adatok visszaolvasását.

Szintaxis: **INPUT#** <aritmetikai kifejezés>, <változólista>

Az aritmetikai kifejezés értékének az 1-255 intervallumba kell esnie. Ez annak a file-nak a **logikai file száma**, amelyikből az adatokat be kívánjuk olvasni. A <változólista> egymástól vesszővel elválasztott egyszerű-, vagy tömbváltozókat tartalmazhat. Az INPUT és a # között nem lehet szóköz.

A file-ból beolvasott karaktereket az interpreter a következő szabály szerint dolgozza fel:

Az alfanumerikus jelek a beolvasott értékek részét képezik, hasonlóan, mint az INPUT utasítás esetében. A <RETURN> (azaz CHR$(13)) olvasásának ugyanaz a hatása, mint a <RETURN> billentyű benyomásának az INPUT utasítás esetén. Hasonlóan a vessző (,) és a kettőspont (:) az egyes tételek végét jelzik (kivéve idézőjel után). Bár az ?EXTRA IGNORED hibajelzéshez hasonló üzenetet nem kapunk, de ha a pufferben levő és CHR$(13)-mal végződő rekord tagoltsága nem felel meg a változólistának, a fölös adatok éppúgy elvesznek, mint az INPUT utasítás esetén. Az input-puffer határt szab az egyetlen INPUT# utasítással beolvasható értékeknek; ezek maximum 79 adatbyte-ból és a CHR$(13) jelből állhatnak.

Példák:

(i)
```
10 OPEN 4,1,1
20 FOR J=1 TO 10:INPUT X$:PRINT#4,X$:NEXT
30 CLOSE 4

40 OPEN 5,1,0
50 FOR J=1 TO 10:INPUT#5,Y$:PRINT Y$:NEXT
60 CLOSE 5
```

A fenti példa egy -, a kazettás egységre vonatkozó - ki/beviteli utasitáspár, amelyen nem tüntettük fel a szalag visszacsévélési, az ST ellenőrző részeket. A file számok (ami az írás esetén 4, az olvasás esetén 5) természetesen tetszőlegesek lehetnek, de hogy jobban látszódjék, hogy különböző célra használjuk őket, nem azonosak az egységszámmal. Az (i) alatti program a billentyűzetről 10 számot olvas be, majd ezeket egy név nélküli szalagos file-ba írja ki. A (ii) alatti programrész ezeket a

számokat visszaolvassa az INPUT#5 utasítás segítségével, és kiírja a képernyőre.

```
(ii)   10 OPEN 1,0: REM FILE#1=A BILLENTYUZET
       20 OPEN 3,3: REM FILE#3=A KEPERNYO
       30 INPUT#1,X$: PRINT#3,X$: GOTO 30
```

A (ii) alatti példa azt mutatja, hogyan kezelhetjük a billentyűzetet file-ként. Ebben az esetben nem az INPUT, hanem az INPUT# utasítást kell az adatok beolvasására használni.

```
(iii)  5 OPEN 1,8,2,"@0:PROBA,S,W"
       10 FOR J=1 TO 10: PRINT#1,STR$(J): NEXT
       20 CLOSE1
       30 OPEN 2,8,3,"PROBA,S,R"
       40 FOR J=1 TO 10: INPUT#2,X$: PRINT X$: NEXT
       50 CLOSE2
```

Az utolsó, (iii) programrész egy lemezes file-ba való írást, majd ugyanennek a file-nak a visszaolvasását mutatja be.

Az INPUT és INPUT# utasítások a billentyűzetről, illetve a megfelelő file-ból kapott karaktereket az input-pufferbe ($0200) másolják. A pufferba való másolást a <RETURN> billentyű lenyomása, illetve egy CHR$(13) olvasása fejezi be. A <RETURN> byte helyett azonban egy CHR$(0) kerül az input-pufferbe. A $01FF címen egy vessző található, ez lehetővé teszi, hogy az inputsor első tételét az interpreter ugyanúgy dolgozza fel, mint a többit. Az INPUT-ról szóló rész egyik példáját az interpreter a pufferben így helyezi el:

$1FF	$200	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0	1	2
,	K	U	T	Y	A	,	1	8	,	5	6	.	8	3	,	7	.	1	null

A tételek feldolgozása hasonló, mint az INPUT és a READ utasítások esetén.

Hibalehetőségek: A szintaktikus hibákon kívül leggyakoribb, hogy a file végén túl akarunk olvasni, vagy a rekordokat nem megfelelően tagolva olvassuk vissza. Ha az input változó típusa nem felel meg a file-ból beolvasott adatnak, akkor ?BAD DATA hibajelzést kapunk. (Az INPUT utasítás ilyenkor küldi a ?REDO FROM START üzenetet. File-ok esetében ennek nincs értelme.)

INT

Rövidítés: nincs *Token*: $B5(181)

Mód: mind parancs -, mind program módban használható

Az argumentum egész részét számítja ki.

Szintaxis: INT(<aritmetikai kifejezés>)

Példák:

```
(i)   PRINT INT(X+.5)
(ii)  PRINT INT(123456.789): REM=123456
(iii) PRINT INT(-123456.789): REM=-123457
(iv)  35 INPUT X: IF X<>INT(X) THEN GOTO 35
```

A legtöbb kerekítési eljárás az INT függvényt használja. Az (i) sor egyszerű példát mutat a kerekítésre. Ha egy számot a legközelebbi egész számra akarunk felkerekíteni, akkor .5-et kell hozzáadni, és utána az egész részét képezni.

A (ii) és a (iii) példa azt illusztrálja, hogy mi a különbség a pozitív és a negatív számok egész részének képzése között. Utolsó példánk azt ellenőrzi, hogy egész számot adtunk-e be inputként.

Hibalehetőségek: Az argumentum kiértékelése közben számos hiba adódhat. Magának az INT-nek a kiszámítása már hiba nélkül megtörténik.

LEFT$

Rövidítés: leF (a $ már nem kell!) *Token*: $C8(200)

Mód: mind program -, mind parancs módban használható.

LEFT$ kétváltozós sztringfüggvény, amelyik az első argumentumként megadott sztring **baloldali karaktereiből** egy új sztringet képez. Az ehhez felhasználandó karakterek számát a második argumentum adja meg. Legyen például

```
     X$="KOVACS JANOS"
Pozíció: 123456789012
```

Ekkor LEFT$(X$,6)="KOVACS"

Szintaxis: LEFT$(<sztring>, <aritmetikai kifejezés>)

Példák:

```
(i)    PRINT LEFT$("HOGY VAGY",4): REM="HOGY"
(ii)   PRINT LEFT$("HOGY VAGY",123): REM NEM VALTOZIK
(iii)  PRINT LEFT$(X$+"                    ",15)
       PRINT LEFT$(STR$(L)+"                    ",15):
       PRINT L;LEFT$("                    ",15-LEN(STR$(L)))
```

(i) a LEFT$ hatását mutatja be. A második példa illusztrálja, hogy ha az aritmetikai kifejezés meghaladja a sztring hosszát, akkor eredményül az eredeti sztringet kapjuk (minden hibajelzés nélkül). (iii) alatt példákat adtunk arra vonatkozólag, hogyan egészíthetünk ki egy-egy sztringet szóközökkel éppen 15 karakterre. A példa első sorában ez egy 'valódi' sztring (X$), a következő kettőben az L valós változó karakteres alakja STR$(L).

LEFT$(X$,N) mindig helyettesíthető a MID$(X$,1,N) kifejezéssel.

Hibalehetőségek: A paraméterek kiértékelése közben előforduló hibákon túl, az N=0 érték ?ILLEGAL QUANTITY ERROR hibajelzést okoz.

LEN

Rövidítés: nincs *Token*: $C3(195)

Mód: mind parancs -, mind program módban használható.

LEN aritmetikai függvény; az argumentum sztring **hosszát** adja meg.

Szintaxis: LEN(<sztring kifejezés>)

Példák:

```
(i)    10 PRINT LEN("HAHO OCSI!"):REM=10
(ii)   20 X$="ELJEN":PRINT LEN(X$+"MAJUS")+2
(iii) 312 FOR J=1 TO 10
       314 PRINT SPC(40/2-LEN(UZENET$(J))/2);UZENET$(J)
       316 NEXT J
(iv)   500 IF LEN(BE$)<>13 THEN PRINT "** HIBA **":GOTO 3456
(v)    780 X$="*#!%↑@"
       790 FOR J=1 TO LEN(X$)
       800 IF B$=MID$(X$,J,1) THEN RETURN
       810 NEXT:PRINT "** FELISMERHETETLEN **"
```

Az első két példa a függvény hatását illusztrálja. A (iii) alatti néhány soros rutin az UZENET$(J) sztringeket írja ki a képernyő közepére. Negyedik példánk egy input rutin része, amelyik ellenőrzi, hogy a bevitt sztring az előírt hosszúságú-e. Az (v) példában egy egyszerű programot írtunk, amelyik ellenőrzi, hogy G$ egy előre megadott karakterkészlet eleme-e vagy sem. A ciklus végértékének természetesen 6-ot is írhattunk volna, de a program módosítása ekkor nehezebb lenne.

Hibalehetőségek: A sztring-kifejezés kiértékelése után egy legfeljebb 255 hosszú sztringet kell kapnunk, ebben az esetben a LEN már hiba nélkül végrehajtódik.

LET

Rövidítés: 1E *Token:* $88(136)

Mód: mind parancs -, mind program módban használható.

Szintaxis:

{LET} { <egész változó> = <aritmetikai kifejezés>
<valós változó> = <aritmetikai kifejezés>
<sztring változó> = <sztring kifejezés> }

értékadó utasítás. A változók egyszerű és tömb változók egyaránt lehetnek. A LET szó kiírása **nem kötelező.** Ha egy utasítás első karaktere nem token, akkor az interpreter LET-et tételez fel.

Példák:

```
(i)   LET X=1234: X$="QWE": LET X%=12.45
(ii)  20 FOR J=1 TO 10: READ X%: LET A%(J)=X%:NEXT
(iii) 30 IF A+B>C THEN D%=A/B
```

Az (i) alatti parancs a különböző típusú változók használatát mutatja be. Egész változó esetén az aritmetikai kifejezés értékének egész része lesz a változó új értéke. Így a parancs végrehajtása után X%=12. A (ii) példában egy összetett értékadás szerepel, egy ciklus a DATA sorokból olvas be, majd ezek értékét egy mátrixba írja. Utolsó példánk egy feltételes értékadást mutat be.

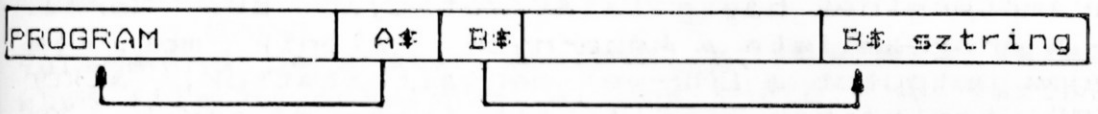
A változók értékei akárhányszor, és a parancs vagy a program bármely részéből megváltoztathatók. A BASIC nem ismeri a 'lokális' és a 'globális' változók fogalmát. Ez különösen az alprogramok megtervezésekor okozhat problémát. Ha egy rutint a program

több részéből is meghívunk, akkor változóit más célokra már nem célszerű használni.

A C-64 BASIC a sztringeket kétféleképpen tárolja. A két tárolási mód közti különbség néha fontos lehet. Tekintsük a következő programrészt:

```
10 A$="HELLO":B$="HELLO"+""
```

Az A$ illetve B$ értékét a program eltérően tárolja:



A változók területén a sztring neve, hossza, illetve a sztring első elemére mutató 2 byte kerül tárolásra. Amennyiben egy sztring-konstans kerül a változóba, a mutató magába a **programba mutat vissza** (helykímélés céljából). A sztringkifejezések eredményeiként előálló sztringek a BASIC munkaterületen kerülnek tárolásra. Ha most egy új programot töltünk be a PROGRAM-ból, akkor A$ értéke elvész, míg B$ értéke megmarad.

Hibalehetőségek: Amennyiben a változó és a kifejezés típusa nem felel meg egymásnak, ?TYPE MISMATCH ERROR hibajelzést kapunk. Ha a változó, amelynek értékét a kifejezés definiálja, tömbváltozó és indexei, nem esnek a megfelelő értékhatárok közé, akkor ?BAD SUBSCRIPT ERROR hibajelzést kapunk. A kifejezés kiértékelése közben további hibaüzeneteket is kaphatunk. Ha egy egész változónak adunk értéket, akkor ?ILLEGAL QUANTITY ERROR hibajelzést kaphatunk, feltéve, hogy a kifejezés értéke nem esik a megengedett határok közé.

LIST

Rövidítés: LI *Token*: $9B(155)

Mód: mind program -, mind parancs módban használható.

A BASIC munkaterületen tárolt programot, illetve annak a LIST paramétereiben specifikált részét írja ki a képernyőre (pontosabban az elsődleges outputra).

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

Példák:

```
(i)   LIST: REM AZ EGESZ PROGRAMOT KILISTAZZA
(ii)  LIST 5: REM CSAK AZ 5. SORT IRJA KI
(iii) LIST 5-135: REM AZ 5-TOL 135-IG ESO SOROKAT IRJA KI
      REM BELEERTVE 5-T ES 135-T IS
(iv)  LIST -135: REM A 135. SORIG LISTAZ
(v)   LIST 135-: REM A 135. SORTOL LISTAZ
(vi)  10 PRINT "** HIBA **":LIST 456
```

A példák a LIST paramétereinek használatát mutatják be. Normál körülmények között a programlista a képernyőn jelenik meg. Ha azonban az elsődleges outputot a CMD-vel megváltoztattuk, akkor a programlista a CMD utasításban specifikált file-ba íródik. Az **OPEN 3,4: CMD 3: LIST** parancs a programot a sornyomtatóra listázza ki. A program kazettára vagy lemezre is kilistázható. (Ekkor azonban LOAD-dal már nem tölthető vissza!)

Az utolsó példa mutatja, hogy a LIST program módban is használható; hatása ugyanaz, mint a STOP utasításé, azzal a különbséggel, hogy a program futása a CONT-tal nem folytatható, és a LIST paramétereinek megfelelő programsorokat az interpreter kilistázza.

A program listája nem betű szerint azonos azzal, ahogyan beírtuk az egyes programsorokat. A ? X utasítás a listán PRINT X alakban jelenik meg. A REM alapszó utáni grafikus jeleket (hacsak nem idézőjelben szerepelnek) a listázó rutin tokeneknek tekinti, és a nekik megfelelő alapszavakat írja ki. A megfelelő memóriahelyek tartalmának módosításával elérhetjük, hogy a megjegyzés sor képernyő vezérlő karaktereket is tartalmazzon. Ilyenkor a LIST hatására igen furcsa kiírási képet kaphatunk. Hasonló a probléma a sztring konstansokban szereplő vezérlő karakterekkel is. Gépeljük be a 10 ?"GO AWAY" utasítást, listázzuk ki a sort és álljunk a kurzorral a második idézőjel fölé, szúrjunk be az <INS> billentyű segítségével nyolc szóközt, majd ezt töltsük fel <DEL> jelekkel (inverz T-ként jelennek meg). A listázás eredménye ezután: 10 PRINT.

A leghosszabb sor, amit ki tudunk listázni, egy öt decimális jegyű sorszámból és 251 RESTORE tokenből áll. (Ezt a sort a szokásos módon nyilván nem tudjuk beírni a programba.)

Hibalehetőségek: Gyakorlatilag nincsenek. A paramétereknek nem **kell létező** sorszámoknak lenniök, a LIST 100- utasítás az első, 100-nál nem kisebb számú sortól kezdi a program listázását.

LOAD

Rövidítés: LO *Token:* $93(147)

Mód: mind program -, mind parancs módban használható.

Az utasítás a paramétereiben megadott nevű programot az adott egységről és az adott módon betölti.

Szintaxis:
LOAD {<sztring kif.> {,<arit. kif.> {,<arit. kif.>}}}

Valamennyi paraméter opcionális, hiszen a kazettás egységen az első file egyértelműen azonosítható. A <sztring kifejezés> a **program neve.** Az első <aritmetikai kifejezés> az **egységszám;** ha elmarad, akkor a LOAD mindig a kazettás egységre vonatkozik (egységszám=1). A második <aritmetikai kifejezés> a töltés módját határozza meg (secondary adress). Kazettás file-ok esetén semmilyen hatásta nincs.

Lemezes file-ok esetén az egységszám (8-11) mindig kötelező; ilyenkor a harmadik paraméter jelentése a következő:

0 a töltés a BASIC munkaterület elejéről kezdődik;
1 a töltés a program-file első két byte-ja által meghatározott címtől kezdődik.

Lemezes file-ok esetén a névmegadásnak speciális formái is megengedettek (lásd az 5.fejezetet!). Ha a név *-gal végződik, például PROG*, akkor az utasítás a lemez katalógusában szereplő és "PROG"-gal kezdődő nevű *első* file-t tölti be. A file-névben szereplő kérdőjelek (?) tetszőleges karaktert jelentenek. LOAD "HE??O",8 például az első pontosan öt karakteres nevű file-t tölti be, amely nevének első két karaktere HE, ötödik karaktere pedig O.

Példák:
```
(i)   LOAD :REM A SZALAGROL AZ ELSO PROGRAM
      LOAD "PROG":REM A SZALAGROL A PROG NEVU PROGRAM
      LOAD "*",8 :REM LEMEZROL A KATALOGUS ELSO PROGRAMJA
      LOAD "AS*",8 :REM LEMEZROL AZ ELSO AS-SEL KEZDODO PROGRAM
(ii)  90 LOAD "PROG",8
(iii) 45 PRINT "VARJ, TOLTOK!":LOAD "PROG2",8
```

Utolsó két példánk BASIC programból használja a LOAD utasítást. Program módban a LOAD utasítás végrehajtása eltér a parancs módban kiadottól. A töltés befejezése után a program futása *elölről* kezdődik. Amennyiben BASIC programot töltöttünk be, az új, éppen betöltött program kezd el futni. Ezt a lehetőséget **overlay** technika kialakítására is felhasználhatjuk. A betöltés az eredeti program változóit nem törli, feltéve, hogy a másod-

szorra betöltött program rövidebb az elsőnél. A sztring konstansokkal végrehajtott értékadások eredményei és a függvény definíciók azonban mindenképpen elvesznek.

Kazettás file esetén a következő információk kerülnek a képernyőre:

```
LOAD "PROGRAM",1 <RETURN>
PRESS PLAY ON TAPE  <PLAY>
OK
SEARCHING FOR PROGRAM
FOUND ...
FOUND PROGRAM
LOADING PROGRAM
READY.
```

A 'PRESS PLAY ON TAPE' üzenet megjelenése után kell a PLAY gombot lenyomnunk. Az interpreter addig keres a szalagon, míg az utasításban specifikált file-t meg nem találja, vagy egy EOT fejlécet nem olvas.

Lemezes file esetén az információ kevesebb:

```
LOAD "PROGRAM",8  <RETURN>

SEARCHING FOR PROGRAM
LOADING PROGRAM
READY.
```

Hibalehetőségek: Elsősorban kazettás egységről való töltés esetén hardver hibák is történhetnek. ?FILE NOT FOUND hibajelzést kapunk, ha a szóban forgó file nem szerepel a lemez katalógusában, illetve a kazettán az EOT jelzőn túl olvasott az interpreter. ?DEVICE NOT PRESENT hibajelzést kapunk, ha az egységszámnak megfelelő egység nincs a C-64-hez csatlakoztatva és bekapcsolva (ez a hibajelzés a kazettás egységgel kapcsolatban nem fordulhat elő). Gépi kódú programok töltése esetén a harmadik paraméter általában kötelező, s elhagyása hibát okozhat. Egy több gépi kódú programból álló program töltőprogramját mutatjuk be végül; az A értékének változtatása nélkül a program egy végtelen ciklusban mindig csak az első programot töltené be:

```
10 IF A=0 THEN A=1: LOAD"SKOT1",8,1
20 IF A=1 THEN A=2: LOAD"SKOT2",8,1
30 IF A=2 THEN A=3: LOAD"SKOT3",8,1
40 IF A=3 THEN A=4: LOAD"SKOT4",8,1
50 SYS 4960
```

LOG

Rövidítés: nincs *Token*: $BC(188)

Mód: mind parancs -, mind program módban használható.

Egyváltozós aritmetikai függvény, amelyik argumentumának e alapú logaritmusát számítja ki. Az EXP függvény inverze. (Lásd a (iv) példát.)

Szintaxis: LOG(<aritmetikai kifejezés>)

Példák:
```
(i)   PRINT LOG(10): REM =2.3026
(ii)  PRINT LOG(2.718281): REM =1 (gyakorlatilag)
(iii) PRINT LOG(X)/LOG(2): REM 2 ALAPU LOGARITMUS
(iv)  PRINT LOG(EXP(X)): REM =X
(v)   50 DEF FN LG(X)=LOG(X)/LOG(10):REM 10 ALAPU LOGARITMUS
      60 PRINT FN LG(100): REM =2
```

Példáink a LOG függvény használatának legegyszerűbb eseteit mutatják be. Magára a függvényre főleg statisztikai, tudományos számítások során van szükség.

A LOG kiszámítása az előjel ellenőrzésével kezdődik, csak **pozitív számok**nak van logaritmusa. A LOG kiszámítása egy olyan hatványsorral történik, amelynek összesen 4 (!) tagja van.

Hibalehetőségek Ha az argumentum értéke nem pozitív, ?ILLEGAL QUANTITY ERROR hibajelzést kapunk.

MID$

Rövidítés: mI(a $ már nem kell!) *Token*: $CA(202)

Mód: mind parancs -, mind program módban használható.

Két-, vagy háromváltozós sztring-függvény, amelyik az első paraméterként megadott sztring adott pozíciójánál kezdődő egymás után következő karaktereiből egy új sztringet állít elő.

Szintaxis:**MID$**(<sztring kif.>,<arit. kif.> [,<arit. kif.>]).

Az aritmetikai kifejezések értékei nem lehetnek 255-nél nagyobbak. A második paraméter a sztring azon karakterhelyét adja meg, **ahonnan** az új sztring kezdődik. A harmadik paraméter (ha van) az új sztring **hosszát** (az átmásolandó karakterek számát) adja meg. Ha nincs ennyi karakter a paraméterként szereplő

sztringben, akkor a sztring végéig másolódnak át a karakterek. Ha a harmadik paraméter hiányzik, akkor az 255-nek felel meg.

Legyen X$="KOVACS PETER".
123456789012

Ekkor MID$(X$, 3,6)="VACS P"
MID$(X$, 8,10)="PETER"=MID$(X$,8).

Példák:
(i) N$=MID$(STR$(N),2)
(ii) MO$=MID$("JANFEBMARAPRMAYJUNJULAUGSEPOCTNOVDEC",3*M-2,3)

Az (i) példánk a STR$(N) első karakterét törli. Ha például N=23, akkor STR$(N)=" 23", de N$="23". Természetesen N=-23 esetén is N$="23" lesz!

A második példa azt mutatja be, hogyan lehet MID$-t 'tömbként' használni. Az utasítás végrehajtása után MO$ az M. hónap hárombetűs (angol) rövidítését tartalmazza.

A LEFT$ és a RIGHT$ függvények a MID$ segítségével kifejezhetők:

LEFT$(X$,N) = MID$(X$,1,N)
RIGHT$(X$,N)= MID$(X$,LEN(X$)-N+1)

Hibalehetőségek: ?ILLEGAL QUANTITY ERROR hibajelzést kapunk, ha a sztring hossza, vagy a paraméterek nagyobbak 255-nél. Hasonló jelzést kapunk, ha az eredmény egy nullsztring lenne. Így például MID$("",I,J) mindig hibás lesz.

NEW

Rövidítés: nincs *Token:* $A2(162)

Mód: mind parancs -, mind program módban használható.

Törli a memóriában tárolt BASIC programot és a változókat.

Szintaxis: NEW

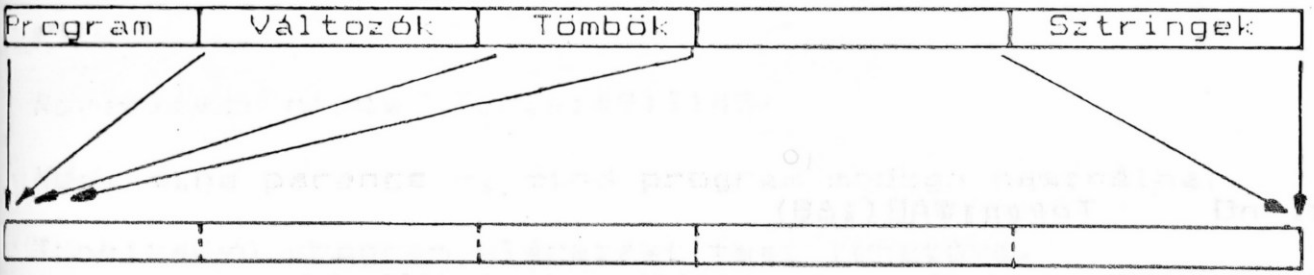
A NEW utasításnak nincsenek paraméterei.

Példák:

(i) NEW
(ii) 1000 PRINT "ODA AZ EGESZ":NEW

Mind parancs módban, mind program módban a hatás ugyanaz. A program és a változók törlődnek.

Az utasítás valójában *nem* törli a memória tartalmát, csupán a BASIC munkaterülethez kapcsolódó mutatókat állítja kezdőértékükre, ahogy az alábbi ábrán is látható:



Az interpreter ezek után úgy érzékeli, hogy üres a memória.

Hibalehetőségek: Nincs (,de elveszhet a program).

NEXT

Rövidítés: nE *Token*: $82(130)

Mód: mind parancs -, mind program módban használható.

A FOR ciklus magját zárja le. Az utasítás hatására a vezérlés a NEXT-nek megfelelő FOR-t követő első utasításra kerül; feltéve, hogy a STEP-ben specifikált értékkel megnövelt ciklusváltozó még mindig a megengedett értéktartományba esik. Ha nem, a NEXT-et követő első utasítás hajtódik végre.

Szintaxis: **NEXT** <változólista>

A változólista egymástól vesszővel elválasztott egyszerű, valós változókat tartalmazhat. NEXT I,J és NEXT I:NEXT J egymással ekvivalensek. Amennyiben a NEXT nem tartalmaz további paramétereket, akkor a legutoljára végrehajtott FOR utasítás alá tér vissza a vezérlés. Példákat a FOR utasítás leírásakor adtunk.

A FOR utasítás szintaxisát a FOR utasítás végrehajtásakor ellenőrzi az interpreter. Az összes szükséges paraméter értéke (összesen 18 byte) a verembe kerül; legvégén a FOR tokenje: 129. A NEXT utasítás végrehajtása a ciklusváltozónak megfelelő 18 byte megkeresésével kezdődik. Ha az interpreter nem találja meg, akkor ?NEXT WITHOUT FOR hibajelzést kapunk. Ha megtalálta a megfelelő FOR tokent, akkor a ciklusváltozó értékét megnöveli a lépésközzel. Ha a ciklusváltozó új értéke átlépi a FOR utasításban megadott végértéket, akkor a NEXT utáni utasításra kerül

a vezérlés. Ha nem, akkor a FOR utáni utasításra.

Hibalehetőségek: A NEXT I utasítás *törli* a veremből a felesleges NEXT-eket. Helytelenül strukturált programok esetén ez később ?NEXT WITHOUT FOR hibajelzést eredményezhet. Nem valós változó használata ?SYNTAX ERROR hibaüzenetet eredményez.

NOT

Rövidítés: nO *Token:*$A8(168)

Mód: mind parancs -, mind program módban használható.

Egyváltozós logikai függvény. Az argumentum értékét két byte-os egész számmá konvertálja, és bitenként végrehajtja a 'nem' logikai műveletet. Az argumentumot nem kell zárójelbe tenni. A NOT műveletáblája a következő:

	NOT
hamis	igaz
igaz	hamis

Szintaxis: NOT <aritmetikai kifejezés>

Példák:

```
(i)   105 IF PEEK(X)=34 THEN ID = NOT ID
(ii)  PRINT NOT 23456: REM =-23457
(iii) 300 IF NOT OK THEN PRINT "** HIBA **":END
```

Az első sor egy listázó rutinban szerepel. Ha a következő byte egy idézőjel ("), akkor az idézőjel jelenlétét ellenőrző ID változó (logikai) ellentettjére változik.

A második példa a NOT használatát aritmetikai argumentum esetén mutatja be. 23456=$5BA0, így a NOT műveletet *bitenként* elvégezve a $A45F hexadecimális számot kapjunk, amelyik a -23457 számnak felel meg. A nyomtatás eredményeként tehát ennyit kell kapnunk.

Utolsó példánkban a NOT használatát feltételes vezérlő utasításban mutatjuk be. Ha az OK változó nem igaz (azaz 0), akkor a HIBA szöveg kiíródik a képernyőre, és a program megáll.

Hibalehetőségek: A NOT művelet a logikai műveletek közt a legmagasabb rangú, ezért először a NOT hajtódik végre. Ennek megfelelően tartsuk szem előtt, hogy például a NOT A AND B ezt

jelenti: (NOT A) AND B. A NOT után álló kifejezés értékének a -32768 - 32767 intervallumba kell esnie, különben ?ILLEGAL QUANTITY ERROR hibajelzést kapunk.

ON

Rövidítés: nincs *Token*:$91(145)

Mód: mind parancs -, mind program módban használható.

Többirányú program elágazást tesz lehetővé.

Szintaxis:

ON <aritmetikai kifejezés> { GOSUB / GOTO } <sorszámlista>

A <sorszámlista> egymástól vesszővel elválasztott előjel nélküli egész konstansokat tartalmaz. Az aritmetikai kifejezést értékeli ki először az interpreter (az értéknek a 0-255 intervallumba kell esnie), majd a lista annyiadik elemének megfelelő programsorra adódik át a program vezérlése. GOSUB esetén a veremba még a visszatérést biztosító információk is bekerülnek.

Példák:

```
(i)   1000 ON SGN(X)+2 GOTO 2000,3000,4000
(ii)  360 ON 1+RND(1)*10 GOTO 10,20,30,40,50,60,70,80,90,100
(iii) 200 ON Q GOSUB 150,200,250
```

Az (i) példa egy előjel szerinti elágazást mutat. Ha X<0 a 2000., ha X=0 akkor a 3000., ha pedig X>0, akkor a 4000. sorral folytatódik a program. (ii) egy véletlen programelágazást mutat be, ilyen programrészek főleg játékprogramokban fordulnak elő. A vezérlés - véletlenszerűen - a 10., 20. stb. programsorok egyikére adódik.

Az interpreter nem ad hibajelzést, ha például a sorszámlista három értéket tartalmaz, de az aritmetikai kifejezés értéke 4. Ha a (iii) példában Q értéke 0 vagy 4, nem kapunk hibajelzést, a program futása a következő sorral folytatódik.

Hibalehetőségek: ?ILLEGAL QUANTITY ERROR hibajelzést kapunk, ha az aritmetikai kifejezés egész része nem esik a 0-255 intervallumba. ?SYNTAX ERROR üzenetet kapunk, ha az aritmetikai kifejezést nem GOTO vagy GOSUB token követi. A GO TO alak használata nem megengedett! Végül ha a szóban forgó számú sor nem létezik, akkor ?UNDEF'D STATEMENT ERROR hibajelzést kapunk.

OPEN

Rövidítés: oP *Token:* $9F(159)

Mód: mind parancs -, mind program módban használható.

Az utasítás első paramétereként szereplő számot, mint logikai file számot felveszi egy táblázatba az esetleges egységszámmal, illetve megnyitási móddal együtt. Amikor egy BASIC utasítás, például egy PRINT# erre a logikai file számra hivatkozik, a hozzá tartozó egységszám és megnyitási mód paramétereket a táblázatból előkeresi az interpreter, és ezek értékének megfelelően hajtja végre az illető utasítást. Az OPEN utasítás hatására azoknál az egységeknél, ahol ez szükséges, az egységen fizikai értelemben is megnyílik a file. Kazettás file-ok esetén ez a címke írását/olvasását jelenti. Lemezes file-ok esetén a paraméterek parancsként a soros buszra kerülnek, az egység megnyitja a file-t, és egy puffert köt le a file-lal való adatcserére.

Szintaxis:
OPEN <arit.kif.> {,<arit.kif.> {,<arit.kif.> {,<sztring kif.>}}}

Az első paraméter kötelező. értéke az 1-255 intervallumba kell, hogy essen. Ez a paraméter a **logikai file szám.** A második kifejezés az egység **hardver számát** adja, ez a 0-15 intervallumba eshet. A harmadik kifejezés a **megnyitási mód**, értéke ugyancsak a 0-15 intervallumba esik. A <sztring kifejezés> a file nevét és egyéb jellemzőit adja meg.

Példák:
```
(i)    100 OPEN 10:REM=OPEN 10,1,0
(ii)   110 OPEN 15,8,15 :REM A HIBACSATORNA MEGNYITASA
       120 OPEN 1,8,4,"#":REM A # PSZEUDO-FILE MEGNYITASA
       130 OPEN 2,8,2,"FILE,S,W":REM A FILE MEGNYITASA IRASRA
(iii)  200 OPEN D,D
       220 <AZ EREDMENYEK KIIRASA>
       230 CLOSE D
```

Az első példa egy kazettás file-t nyit meg. (A kazettán egyszerre csak egy file lehet megnyitva.) A fenti utasítás hatására a fejléc a kazetta pufferbe kerül és ott (pl. PEEK-kel) megvizsgálható. A második példában a 8-as lemezegységen három file-t (15,1,2 logikai fileszámokkal) nyitunk meg. A harmadik példában, D értékétől függően (0,4) vagy a képernyőt, vagy a nyomtatót nyitjuk meg. Az eredmények kiírására célszerű olyan szerkesztő karaktereket használnunk, amelyek mindkét egységen ugyanúgy hatnak.

A C-64 a következő megnyitási módokat értelmezi:

Periféria neve	száma	Megnyitási mód	Parancs
Billentyűzet	0		
Kazettás magnó	1	0=input 1=output 2=output+EOT	a file neve
Modem	2	0	kontrol regiszterek
Képernyő	3	0,1	
Nyomtató	4,5	0=grafikus jelek 7=normál	kiírandó szöveg
Lemezegység	8-11	2-14 adatcsatorna 15=hiba csatorna 0=SAVE 1=LOAD	meghajtó a file neve írás/olvasás file típusa

Az OPEN utasítás hatására az első három paraméter értéke bekerül a nyitott file-okat leíró táblázatokba. Ezek a következők:

1. Logikai file-számok ($259-$262, 601-610)
2. Egységszámok ($263-$26C, 611-620)
3. Megnyitási módok ($26D-$276, 621-630)

($98) tartalmazza a nyitott file-ok számát. Az OPEN utasítás kiadásakor először ellenőrzi az interpreter, hogy a táblázatban szerepel-e ilyen sorszámú logikai file. Ha igen, ?FILE OPEN hibaüzenettel térünk vissza a szerkesztőbe. Ha mind a 10 logikai file nyitott, akkor ?TOO MANY FILES hibaüzenetet kapunk. Az OPEN utasítás még a következő címeket használja:

($B8) = logikai file-szám
($B9) = megnyitási mód
($BA) = egységszám
($BB) = a parancs sztring kezdete
($B7) = a parancs sztring hossza

Hibalehetőségek: A paraméterek kiértékelése közben különféle hibajelzéseket kaphatunk. Programhiba után célszerű az összes file-t lezárni, különben újraindítás után ?FILE OPEN ERROR hibajelzést kaphatunk. Nem megfelelő működést eredményezhet a megnyitási mód, illetve a parancs sztring szerkezetének pontatlan ismerete.

OR

Rövidítés: nincs *Token*: $B0(176)

Mód: mind parancs -, mind program módban használható.

Két aritmetikai kifejezés logikai 'vagy'-át számítja ki. Az OR művelettáblája a következő:

OR	hamis	igaz
hamis	hamis	igaz
igaz	igaz	igaz

Szintaxis:
<aritmetikai kifejezés> **OR** <aritmetikai kifejezés>

Példák:

```
(i)   100 IF A<0 OR A>2000 THEN A$="ROSSZ"
(ii)  200 PRINT -1 OR 1234: REM =-1
(iii) 210 PRINT 380 OR 75: REM =383
(iv)  400 A%=12+(A<0 OR A>7)*(X-9)
```

Az (i) és (iv) példában az OR tipikus használatát látjuk. Az első egy egyszerű feltételes értékadás. (iv)-ben a feltételes értékadás az aritmetikai kifejezésbe épül bele, felhasználva, hogy a logikai kifejezések egyben aritmetikai kifejezések is. A (ii),(iii) példák az OR aritmetikai kifejezésekkel való használatát mutatják be. A második sor eredménye -1, hiszen -1=$FFFF alakban kerül tárolásra, és itt mindegyik bit magas. Az OR művelet eredménye így $FFFF=-1 lesz. A harmadik példa a 00000001 01111111 bit-képet adja, ami 383.

Az 'OR' művelet a műveletek közt a legalacsonyabb rendű, így utoljára hajtódik végre.

Hibalehetőségek: Megegyeznek az AND utasításnál leírtakkal.

PEEK

Rövidítés: pE *Token*: $C2(194)

Mód: mind parancs -, mind program módban használható.

A PEEK egyváltozós aritmetikai függvény, amelyik az argumentumaként megadott sorszámú memória címen levő byte értékét adja.

Szintaxis: PEEK(<aritmetikai kifejezés>)

A kifejezés értékének a 0-65535 intervallumba kell esnie.

Példák:

```
(i)   PRINT PEEK(198)
(ii)  CIM=PEEK(43)+256*PEEK(44)
(iii  PRINT "<CLR>"; CHR$(34);:FOR J=CIM TO CIM+100:
        PRINT CHR$(PEEK(J));: NEXT
(iv)  200 SID=13*4096+4*256
      210 POKE SID+24, PEEK(SID+24) OR 15
(v)   100 DEF FN DEEK(X)=PEEK(X)+256*PEEK(X+1)
      120 PRINT FN DEEK(43)
```

Az (i) példa a PEEK hatását mutatja be, a parancs kiírja a képernyőre a 198. memória tartalmát. Ez általában 0, lévén, hogy az interpreter ebben a memóriában tárolja a billentyűzet pufferben levő karakterek számát.

A (ii) példa a 43-44 címen levő két byte-os mutató értékét számítja ki. Ez a mutató adja meg, hol kezdődik a memóriában a BASIC program. A (iii) parancs (amelyet *egyetlen* sorba gépelve kell kiadni) kiírja a képernyőre a BASIC program első 100 byte-ját. Természetesen nem a LIST-tel megszokott alakban, hanem abban a formában, ahogy a gép tárolja. Ezért kell az elején egy idézőjelet (CHR$(34)) kiírni, nehogy a vezérlő karakterek elrontsák a szöveget.

A (iv) példában a SID hanggenerátorának hangerejét 15-re állítjuk be, anélkül, hogy a memória felső négy bitjét megváltoztatnánk. (Azok szabályozzák a szűrést.)

Az (v) példa egy *dupla-pontosságú* PEEK-et definiál, amelyik egy két byte-os mutató értékét számítja ki. Ezzel a függvénnyel a (ii) példában szereplő számítás a 120. sorban látható.

Hibalehetőségek: Az aritmetikai kifejezésnek a 0-65535 intervallumba kell esnie. Ha nem oda esik, akkor ?ILLEGAL QUANTITY ERROR hibajelzést kapunk.

POKE

Rövidítés: pO *Token*: $97(151)

Mód: mind parancs -, mind program módban használható.

Az utasítás segítségével egy tetszőleges memória címre közvetlenül beírhatunk egy 0-255 közti egész számot.

Szintaxis: **POKE** <aritmetikai kifejezés>,<aritmetikai kifejezés>

Az első kifejezés a memória címét, a második a tárolandó byte-ot adja meg. Ennek megfelelően értékeiknek a 0-65535, illetve a 0-255 intervallumba kell esniük.

Példák:

```
(i)   10 FOR J=0 TO 255:POKE 1024+J,J:POKE 55296+J,6: NEXT J
(ii)  30 DATA 162,0,138,157,0,30,232,208,249,96
      40 FOR J=828 TO 837:READ X:POKE J,X:NEXT
      50 SYS 828
(iii) 600 FOR J=2048 TO 9E9:POKE J,170
      610 IF PEEK(J)=170 THEN NEXT
(iv)  700 POKE SID+4,65
```

A négy példában megpróbáltuk érzékeltetni azokat a lehetőségeket, amit a POKE utasítás biztosít. Fontosságát mi sem bizonyítja jobban, mint az a tény, hogy a C-64 BASIC-ből zenélni csak a POKE utasítás segítségével lehet.

Az első példa a képernyő tetejére kiírja mind a 256 karaktert. A második példa azt mutatja, hogyan lehet a DATA utasításban tárolt gépi kódú programot 'betölteni' a POKE utasítás segítségével a memória megfelelő részébe, ezúttal a kazetta pufferba. Ezután a SYS 828 kiadásával hívhatjuk a gépi kódú alprogramot. Hatására ugyanaz történik, mint ami az első példában.

A (iii) példa a memória megadott részét ellenőrzi. Amikor a visszaolvasott érték már nem 170, az vagy a memória végét, vagy hibás címet jelent.

Utolsó példánk a hanggenerátor egyik **hangját** kapcsolja be.

Egy 'dupla-pontosságú' POKE függvényként nem írható fel. Egy egyszerű alprogram, amelyik ezt elvégzi, a következő:

```
POKE Z1,Z2-INT(Z2/256)*256: POKE Z1+1, Z2/256
```

Hibalehetőségek: A POKE paramétereinek kiértékelése közben számos hiba történhet. Ha a memóriacím, vagy a byte értéke nem megfelelő, akkor ?ILLEGAL QUANTITY ERROR hibajelzést kapunk.

POS

Rövidítés: nincs *Token:* $B9(1[illegible])

Mód: mind parancs -, mind program módban használható.

Kiszámitja a kurzor helyzetét az aktuális képernyő sorban. A lehetséges érték a 0-255 intervallumba esik. Ez nem a képernyő 40 karakteres sorában levő pozició, hanem annak a mértéke, amennyivel a kurzort a sor elejétől elmozgattuk. Programsorok esetén ez maximum 80 lehet.

Szintaxis: POS(<kifejezés>)

A POS-nak ál-argumentuma van, amelynek értéke, tipusa lényegtelen. Általában POS(0)-t használunk.

Példák:
```
(i)   IF POS(0)>39 THEN PRINT CHR$(13)
(ii)  PRINT TAB(10)POS(0);: PRINT SPC(10)POS(0)
(iii) PRINT LEFT$("          ",12-POS(1));K$
```

A POS talán a leghasznavehetetlenebb BASIC alapszó. Az (i) példában ellenőrizzük, hogy nem vagyunk-e a sor végén, s ha igen, egy <RETURN>-t irunk ki. A (ii) példa a POS hatását szemlélteti. A parancs eredményül 5-öt és 13-at nyomtat. A harmadik példa a kurzor helyzetétől a 12. pozicióig szóközöket nyomtat, utána pedig kiirja a K$ sztringet.

A POS ugyanazokat a paramétereket használja, mint a TAB. Igy a POS utasitás nyomtatókkal kapcsolatban csak egy és ugyanazon **fizikai soron** belül használható.

Hibalehetőségek: Nincs.

PRINT

Rövidítés: ? *Token:* $99(153)

Mód: mind parancs -, mind program módban használható.

Kiszámitja a paraméterek értékét, és az utasitásban megjelölt formában kinyomtatja az elsődleges output file-ba (általában a képernyőre).

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

A <nyomtatási kép> aritmetikai és/vagy sztring kifejezések sorozata, amelyeket egymástól a következő **szeparátor**okkal választhatunk el:

SPC(<aritmetikai kifejezés>)
TAB(<aritmetikai kifejezés>)
szóköz
vessző (,)
pontosvessző (;)
(Ahol ez megengedett, nem kell szeparátor).

Példák:

```
(i)   FOR J=0 TO 255: PRINT CHR$(J);:NEXT
(ii)  FOR J=0 TO 100: PRINT J,:NEXT
(iii) PRINT X+Y;124;P*G*(1-V)
(iv)  PRINT "HELLO";A$B$U$U1$;MID$(X$,2)
(v)   PRINT TI;TI$;ST;<pi>
(vi)  PRINT: PRINT
```

A fenti hat példa a PRINT utasitás szinte valamennyi lehetőségét bemutatja, kivéve a SPC és TAB használatát (ezek a megfelelő kulcsszónál találhatók meg). Az első példa 256 karaktert ir ki a képernyőre. Mivel ezek közt vezérlő karakterek is vannak, a kiirás közben a képernyő törlődhet, szines karakterek jelenhetnek meg stb. (ii) a vessző (,) használatát mutatja be. A PRINT utasitásban szereplő értékek kiírása a **legelső tabulálási ponttól** kezdődik. A 40 karakter hosszu sorban **négy tabulálási pont** van az 1., 11., 21, illetve 31. oszlopban. Ennek megfelelően a (ii) példa a 100 számot 25 sorban irja ki, minden sorba négyet-négyet.

Ha a kinyomtatandó mennyiségeket pontosvesszővel (;) választjuk el egymástól, akkor a nyomtatás a **következő karakterhely**en kezdődik. (iii) és (iv) példa az aritmetikai és a sztring kifejezések használatát mutatja be. Az aritmetikai kifejezések értéke a STR$(X) alakban nyomtatódik ki, ami után még egy <CRSR JOBBRA> is kinyomódik. Ilyen módon még ha pontosvesszővel (;) elválasztott számokat is nyomtatunk ki, egy szóköz mindig kerül közéjük. Pozitiv számok esetén a pozitiv előjel helyett egy

szóköz nyomtatódik ki. A (iv) példa a szeparátorok elhagyását is bemutatja. A $, % és a tömbváltozók végét jelző záró zárójelet az interpreter **terminátor**nak tekinti, ezért ezek után a **pontosvessző elhagyható.**

A szeparátorok elhagyása több esetben is lehetséges, de ezeknek a használatát nem javasoljuk:

```
10 PRINT (2)(7) :REM 2  7
20 PRINT 1.2..3 :REM 1.2  0  .3
30 PRINT KESZ.  :REM 0  0
40 PRINT ;,L*K;4:REM          0  4
```

Az (v) példánk azt mutatja be, hogy a PRINT utasitás felismeri a fenntartott változókat (TI, TI$, ST) és a pi-t.

A (vi) példa az üres PRINT utasitás használatát mutatja be: a (vi) parancs két soremelést nyomtat.

Vezérlő karakterek használata

Mint már a 2. fejezetben emlitettük, a PRINT utasitásban szereplő sztring kifejezések vezérlő karaktereket is tartalmazhatnak, ezek 'kinyomtatása' a megfelelő vezérlő funkció végrehajtásával jár. Ugyanezt a hatást a CHR$(X) alaku sztringek használatával is elérhetjük. Például a

```
PRINT CHR$(18)"HAHO" CHR$(146)"HAHO"
```

parancs az első HAHO-t inverz alakban irja ki. A képernyőn nem biztos, hogy megjelenik a PRINT utasitással kiirt szöveg, mert az, amit a képernyőn látunk, nemcsak a képernyőmemória tartalmától függ. Részletesen a 7. fejezetben szólunk a C-64 grafikus lehetőségeiről.

A C-64 BASIC-ből hiányoznak a számok formázott kiiratását biztositó vezérlő karakterek. Ezen a CBM kompatibilis nyomtatók általában segitenek, mert lehetővé teszik a számok formázott kiirását a központi egységtől függetlenül. Ha ilyennel nem rendelkezünk, akkor a sztringműveletek segitségével a számokat megadott alakban irathatjuk ki. (Szöveg típusú változót nem használó nyelvek esetén különféle FORMAT utasitást kell használnunk.) Tételezzük fel, hogy egy pozitiv számot 4 tizedesre kerekítve, egy 9 hosszu mezőben jobbra tömörítve szeretnénk kiiratni:

```
#  #  #  #  .  #  #  #  #
```

Első lépésként ellenőrizzük, hogy a szám a megadott intervallumba esik-e, majd 4 tizedesre kerekitjük. Ezt könnyen elvégezhetjük. Ezután kell ellenőriznünk, hogy a STR$(X) alakban van-e tizedespont, hány 0-val kell a végén a számot kiegészíteni

stb. Ezt a feladatot oldja meg például a következő programrész:

```
5 INPUT X
10 IF X>9999.9999 THEN PRINT"** TUL NAGY **":END
20 X=INT(X*1E4+.5) :REM KEREKITES
30 XE%=INT(X/1E4): REM EGESZRESZ
40 XT%=X-XE%*1E4 : REM TORTRESZ, MINT EGESZ
50 XE$=RIGHT$("  "+STR$(XE%),4)
60 XT$=RIGHT$("0000"+MID$(STR$(XT%),2),4)
70 PRINT XE$+"."+XT$
100 GOTO 5
```

Hibalehetőségek: A SPC és TAB paraméterei egész részének a 0-255 intervallumba kell esnie. Amennyiben az utasitásban nincs szintaktikus hiba, végrehajtódik. Ez persze messze nem jelenti azt, hogy olyan képet is kapunk, amilyent megterveztünk.

PRINT#

Rövidités: pR (a # már nem kell!) *Token*: $98(152)

Mód: mind parancs -, mind program módban használható.

Kiszámitja a paraméterek értékét és a megadott formában kiirja az utasitásban specifikált logikai file-ba.

Szintaxis: **PRINT#** lf {,<nyomtatási kép>}

lf a logikai file szám. A <nyomtatási kép> alakja azonos a PRINT utasitásban szereplővel. A PRINT és a # jel közé nem irhatunk szóközt!

Példák:

```
(i)    100 OPEN 4,4
       110 PRINT#4,"ARAJANLAT"
(ii)   200 OPEN 1,4,0: OPEN 2,4,7
       210 PRINT#1,"ARAJANLAT";: PRINT#2,"ARAJANLAT"

(iii)  1000 OPEN 5,8,5,"ADATOK,S,W"
       1010 FOR J=1 TO 10: READ A: PRINT#5,J;A: NEXT J
       1020 CLOSE 5
       1030 DATA 1,2,3,4,5,6,7,8,9,10
```

A fenti példák csak a nyomtató és a lemezegység használatát mutatják be, de hasonlóan lehet a többi perifériás egységet is programozni. Az (i) példa a nyomtatóra irja ki az "ARAJANLAT" szöveget.

A (ii) példában két csatornát nyitottunk meg a nyomtatón, és felváltva használjuk ezeket a PRINT# utasitásban. Ennek hatására az egyik "ARAJANLAT" nagy, a másik kisbetükkel nyomtatódik ki. A #1 logikai file ugyanis a nagybetük/grafikus jelek karakterkészlet, mig a #2 file a kisbetük/nagy betük karakterkészlet segitségével nyomtat.

Utolsó példánk az ADATOK nevü lemezes file-ba ir 20 számot.

Hibalehetőségek: A SPC, a TAB és a vezérlő karakterek a kiirás során nehézségeket okozhatnak; előfordulhat, hogy a file tartalma nem egyezik meg a kivánttal.

READ

Rövidités: rE *Token:* $87(135)

Mód: mind parancs -, mind program módban használható.

A program soraiban elhelyezett DATA utasitásokban definiált értékeket olvassa, és rendeli hozzá az utasitásban megadott változókhoz.

Szintaxis: **READ** <változólista>

A <változólista> tetszőleges tipusu, egymástól vesszővel elválasztott egyszerü- vagy tömbváltozókat tartalmaz. Legalább egy változó megadása kötelező.

Példák:

```
(i)    10 DATA 5,23,45,87,23,98
       20 READ N
       30 FOR I=1 TO N: READ X1%(N):NEXT I

(ii)   110 READ CIM:IF CIM<0 THEN LO%=(CIM+U) AND 255:
              HI%=(CIM+U)/256

(iii)  FOR I=1 TO 10:READ A:PRINT A,:NEXT
```

Az első példa azt illusztrálja, hogyan lehet a DATA utasitások első elemeként megadni a tárolt értékek számát. A READ utasitás segitségével először beolvassuk N értékét, majd egy 1..N ciklusban az adatokat.

A (ii) példa egy áthelyezhető gépi kódu program töltését mutatja. Amennyiben az érték negativ, a cimet U hozzáadásával kell előállitani.

(iii) a READ parancs módban való használatát mutatja; a parancs a tárolt program következő 10 adatát olvassa be, és irja ki a képernyőre.

Hibalehetőségek: Ha a változó és a DATA utasitásban talált érték tipusa nem felel meg egymásnak, akkor ?TYPE MISMATCH ERROR vagy ?SYNTAX ERROR hibajelzést kapunk. Ha a DATA-ban levő adatok elfogytak, ?OUT OF DATA ERROR hibajelzést kapunk.

REM

Rövidités: nincs *Token*: $8F(143)

Mód: mind parancs -, mind program módban használható.

Lehetővé teszi megjegyzések beirását a program szövegébe. Az interpreter a REM észlelésétől a sor maradék részét már nem hajtja végre, hanem a **következő sor** elejéről folytatja a végrehajtást.

Szintaxis: **REM** <karaktersorozat>

A <karaktersorozat> tetszőleges karaktereket tartalmazhat; beleértve a kettőspontot is (:). A REM után - a következő sorig - bármi állhat.

Példák:

(i)
```
7000 REM *** FOPROGRAM ***
7010 GOSUB 7040: REM *** KEZDOERTEKEK
7020 GOSUB 7200: REM *** SZAMITASOK
7030 GOSUB 8150: END: REM *** EREDMENYEK
7040 REM *** SZUBRUTINOK
```

(ii)
```
10 REM "

*** FOPROGRAM ***
```

(iii)
```
X$=A$+CHR$(13)+A$+B$:GOSUB 2234:REMUB 2367
```

Az (i) példa a REM leggyakoribb felhasználását mutatja, amikor az utasitások hatását irja le. A (ii) példában a 10. sorban a REM-mel kapcsolatos egyik legegyszerübb trükköt mutatjuk be; a REM után két <SHIFT-RETURN> karaktert POKE-oltunk be, aminek hatására a listázáskor a megjegyzés uj sorban jelenik már meg. Ennek billentyüzése a következőképpen lehetséges:

```
10 REM "  *** FOPROGRAM *** <SHIFT-RETURN>
<CRSR FEL>, nyolcszor <CRSR JOBBRA> <CTRL RVSON>
<SHIFT-M> <SHIFT-M> <RETURN>
```

Listázáskor a sor a fenti alakban jelenik meg.

A (iii) példa a REM parancs módban való használatát mutatja be. A program egy sorát kilistáztuk, töröltük a sorszámot és a nem kívánt rész elé egy REM-ot helyeztünk el. A <RETURN> megnyomása után a parancs a REM-ig hajtódik végre. (Az eredeti sor egy GOSUB 2367 utasítást tartalmazott.)

A LIST parancs a REM sorokban talált tokeneket nem grafikus jelként, hanem eredeti alapszavuknak megfelelően listázza ki.

Hibalehetőségek: Ha a REM-sor <SHIFT-L> karaktert tartalmaz, a listázás ?SYNTAX ERROR hibaüzenettel megszakad. (Az interpreter hibája.)

RESTORE

Rövidítés: reS *Token*: $8C(140)

Mód: mind parancs -, mind program módban használható.

Az utasítás hatására a READ utasítások előről kezdik el olvasni a DATA sorokban tárolt adatokat.

Szintaxis: **RESTORE**

Példák:

(i)
```
950 DATA ....
980 DATA ....
1000 DATA GEPI KOD1,169,0,141,....
1020 DATA ....
1340 RESTORE: FOR I=1 TO 9E9: READ X$
1345 IF X$<>"GEPI KOD1" THEN NEXT
1350 FOR L=828 TO 912:READ X:POKE L,X:NEXT
```

(ii)
```
2010 READ X$: IF X$="END" THEN RESTORE
```

Az első példa azt mutatja, hogyan lehet a DATA utasítások közt egy adott részt megtalálni. Először végrehajtatjuk a RESTORE utasítást, majd addig olvassuk a DATA-kat, míg a "GEPI KOD1" sztringet meg nem találjuk. Ezt követik a gépi kódú rutin byte-jai, amiket az 1345-ös ciklusban olvasunk be.

A második példában az utolsó DATA tétel egy "END" sztring. Ennek az olvasása után a RESTORE a DATA mutatót az első DATA utasításra állítja.

A NEW, RUN, CLR utasítások egyben egy RESTORE-t is végrehajtanak. Program indítása előtt a RESTORE-t így csak a GOTO <sorszám> esetében érdemes használni.

Hibalehetőségek: Nincs.

RETURN

Rövidítés: reT *Token:* $8E(142)

Mód: mind parancs -, mind program módban használható.

Az utasítás hatására a program vezérlése az utolsó GOSUB hívást követő első utasításra kerül.

Szintaxis: RETURN

Példák:

```
(i)   10 INPUT X : GOSUB 1500 : GOTO 10
      1500 RQ=0.5: X=INT((X-.005)/RQ+1)*RQ
      1510 PRINT X;: RETURN
```

Egyetlen példánk az 1500. sorban kezdődő kerekítési eljárás ellenőrzését mutatja be. A 10. sorban különböző értékeket adhatunk a programnak és kerekített alakját ellenőrizhetjük.

Az utasítás szintaxisának ellenőrzése után a rutin végigolvassa a vermet FOR és GOSUB tokeneket keresve. A FOR tokeneket és a hozzájuk tartozó információt törli a veremből. A GOSUB megtalálása után visszatölti a megfelelő értékeket, majd megkeresi a következő kettőspontot (:) vagy a sor végét, és onnan folytatja a futást. Ha például az

```
ON L GOSUB 10,20,30: PRINT X
```

utasítás segítségével hajtottunk végre egy szubrutinhivást, akkor a veremből való visszatöltés után még a PRINT X is végrehajtódik.

Hibalehetőségek: Ha az eljárás nem találja meg a visszatérési címet, ?RETURN WITHOUT GOSUB hibajelzést kapunk. A RETURN végrehajtása törli a verem tetején levő FOR-ra vonatkozó információkat. A

```
10 GOSUB 1000:NEXT J
1000 FOR J=0 TO 10:RETURN
```

program végrehajtása közben ezért ?NEXT WITHOUT FOR hibajelzést kapunk.

RIGHT$

Rövidítés: rI (a $ már nem kell!) *Token:* $C9(201)

Mód: mind parancs-, mind program módban használható.

A sztring függvény az argumentum sztring **jobboldali** karaktereiből egy új sztringet képez. A felhasználandó karakterek **számát** a függvény második paramétere határozza meg.

Szintaxis: **RIGHT$(**<sztring kifejezés>, <aritmetikai kifejezés>**)**

Az aritmetikai kifejezés értéke nem lehet 255-nél nagyobb.

```
Legyen    X$="KOVACS PETER"
    Pozíció= 123456789012

Ekkor RIGHT$(X$,5)="PETER"
      RIGHT$(X$,25)=X$.
```

Példák:

```
(i)  10 PRINT RIGHT$("ABRAKDABRA",3):REM=BRA
(ii) 20 PRINT RIGHT$("          "+STR$(N),10)
```

Az első példa a RIGHT$ hatását mutatja be, a második az N értékének megfelelő számot jobbra tömörítve, 10 karakterpozíción írja ki.

A RIGHT$ függvény helyettesíthető a MID$ függvénnyel:

```
RIGHT$(X)=MID$(X$,LEN(X$)-N+1)
```

Hibalehetőségek: A második paraméter értéke nem eshet a megadott értékhatárokon kívül.

RND

Rövidítés: nincs *Token*: $BB(187)

Mód: mind parancs -, mind program módban használható.

A 0-1 zárt intervallumba eső **pszeudo-véletlen számok**at generál.

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

Az aritmetikai kifejezés értékének csak az **előjele** számít.

Példák:

```
(i)   10 FOR J=0 TO 3000*RND(1):NEXT
(ii)  700 DATA 11,22,33,44,55,66,77,88,99
      710 FOR J=1 TO 9*RND(1):READ X$: NEXT J
(iii) 200 X=(B-A)*RND(1)+A
```

Az **első** példa a program futását maximum három másodpercre - de véletlen hosszúságú ideig - felfüggeszti. A második példa a kilenc elemű DATA listát véletlenszerű eleméig olvassa. Így tudunk diszkrét értékek közül véletlenszerűen választani.

Utolsó, (iii)-as példánk az (A,B) intervallumba **eső** véletlen számot állít elő.

Az RND utasítás természetesen nem állít elő 'igazi' véletlen számokat. Az argumentum előjele határozza meg, hogy milyen eljárás segítségével számítja ki az interpreter a következő számot. Az interpreter az utoljára előállított véletlen számot külön tárolja, és bizonyos számelméleti függvények segítségével kapja meg a következőt. Amennyiben az RND függvény argumentuma pozitív, az interpreter az így képzett véletlen számot generálja. **Negatív** argumentum az **első véletlen szám**ot állítja elő; X=RND(-1):PRINT X értéke **mindig ugyanaz**. Végül RND(0) a CIA chipek óra-regisztereit használja fel véletlen számok előállítására. Ez 'inkább' véletlen lesz, mintha csak a számelméleti függvényeket használnánk, de ez sem teljesen véletlen. Ha például egy cikluson belül ismételten használjuk a RND(0) utasítást, akkor bizonyos szabályosság azonnal jelentkezik a végrehajtási időben is.

Hibalehetőségek: Az aritmetikai kifejezés kiértékelése okozhat csak hibát.

RUN

Rövidítés: rU *Token*: $8A(138)

Mód: mind parancs -, mind program módban használható.

A memóriában tárolt programot az elejéről, vagy egy adott ponttól kezdve végrehajtja. A változók előző értéke elvész, a RUN ugyanis végrehajt egy CLR és egy RESTORE utasítást is.

Szintaxis: RUN [<sorszám>]

Példák:

```
(i)   RUN
(ii)  RUN 1000
(iii) 5000 IF X<>0 THEN RUN
```

Az első két sor a RUN parancs módban való használatát szemlélteti. Az utolsó példában, ha X<>0, akkor a RUN utasítás törli az összes változót és a program elöről kezd el futni. Ha a RUN utasítást nem követi egy kettőspont (:), akkor az interpreter sorszámot tételez fel. Igy RUN X és RUN "PRG" hatása egyaránt RUN 0 (lásd a GOTO utasítást!). Az X=80: RUN X hatására a program nem a 80., hanem a 0. sortól kezd el futni, de nem kapunk hibajelzést!

A <SHIFT-RUN> billentyű lenyomása ekvivalens a

```
LOAD <RETURN>
RUN  <RETURN>
```

billentyűzésével, tehát a szalagról betölti az első programot, és azonnal el is indítja.

Hibalehetőségek: Ha a RUN utasításban megadott sorszámú programsor nem létezik, akkor ?UNDEF'D STATEMENT ERROR hibajelzést kapunk.

SAVE

Rövidítés: sA *Token:*$94(148)

Mód: mind parancs -, mind program módban használható.

Az utasításban megadott nevű file-ba **átmásolja** (**elmenti**) a memóriában tárolt BASIC programot.

Szintaxis:
SAVE [<sztring kif.> [,<arit. kif.> [,<arit. kif.>]]]

A második <aritmetikai kifejezés> egyedül a kazettás egység esetén érdekes. Ha értéke 2, akkor a file elmentése után még egy EOT fejléc (header) is kiíródik a szalagra. (Lásd a kazettás egységről szóló fejezetet.)

Példák:

```
(i)   SAVE      : REM NEV NELKUL A KAZETTAN
(ii)  SAVE "",8: REM HIBAJELZEST KAPUNK, "" NEM NEV
(iii) SAVE "PROG",8:REM A PROGRAM NEVE PROG LESZ A LEMEZEN
(iv)  SAVE "@0:REGI",8:REM LEMEZEN MAR MEGLEVO FILEBA
(v)   12 SAVE "TEST"+TI$,8
```

A fenti példák - úgy hisszük - önmagukért beszélnek. Az (i) példa a kazettás egységen egy **név nélküli** file-t hoz létre. Ilyen a lemezegységen nem lehet, ezért a (ii) példa hibás.

A (iii) példa a lemezegység leggyakoribb használatát mutatja. A memóriában tárolt program kiíródik a lemezre egy "PROG" nevű file-ba. Ha egy ilyen nevű file már létezett, akkor az elmentés nem történik meg. A (iv) példa mutatja, hogyan érhetjük el, hogy a már meglevő file-t felülírjuk. A (iv) alatti parancs kiadása **mindig** elmenti a tárolt programot. Ha a lemezen volt már egy "REGI" nevű program, annak az előző tartalma elvész, és helyére a jelenleg a memóriában levő program kerül.

Az utolsó példa a SAVE utasítás programból való használatát mutatja; a programrész időről időre a tárolt programot kiírja a memóriából. Hogy ezek a file-ok különbözőek legyenek, a név utolsó hat karakterét a TI$ adja, ami mindig más és más. (A dolognak csak akkor van persze értelme, ha a program folyamatosan módosítja önmagát...)

Hibalehetőségek: Ha az utasításban megjelölt egység nincs bekapcsolva, vagy hibásan működik, akkor ?DEVICE NOT PRESENT hibaüzenetet kapunk. A kazettás egységre való kimentés aszinkron. Ha például csak a <PLAY> billentyűt nyomjuk meg, az utasítás hibajelzés nélkül végrehajtódik, bár a szalagra a program természetesen nem kerül kiírásra.

Ha a lemezen már meglevő file-ba kísérelјük meg kimenteni a programot (a "@:"-nélkül), akkor nem kapunk hibajelzést, csak a lemezegység **lámpája gyullad ki, vagy villog.**

SGN

Rövidítés: sG *Token*: $B4(180)

Mód: mind parancs -, mind program módban használható.

Az aritmetikai függvény az argumentum előjelétől függően a következőt számítja ki:

$$SGN(X)=\begin{cases} +1 & \text{ha } X>0; \\ 0 & \text{ha } X=0; \\ -1 & \text{ha } X<0. \end{cases}$$

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

Példák:

```
(i)   10 IF SGN(X)>0 THEN PRINT X;"POZITIV"
(ii)  20 IF ABS(SGN(X))<>0 THEN PRINT X;"NEM NULLA"
(iii) 30 ON SGN(X)+2 GOSUB 100,110,120
```

Az első két sor az SGN egyszerű használatát mutatja. A harmadik példa a FORTRAN aritmetikai GOTO utasítását szimulálja. (FORTRAN ekvivalens: IF(X) 100,110,120). A vezérlés a 100., 110. illetve 120. sorba kerül, attól függően, hogy X negatív, nulla vagy pozitív volt-e.

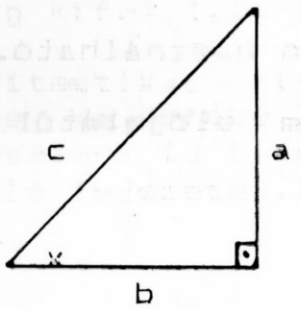
Hibalehetőségek: Csak az argumentum kiértéklése okozhat hibát.

SIN

Rövidítés: sI *Token*: $BF(191)

Mód: mind parancs -, mind program módban használható.

A radiánban megadott szög szinuszát számítja ki. Az alábbi ábra illusztrálja x és sin(x) viszonyát:



$$\sin(x)=\frac{a}{c}$$

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

Példák:

```
(i)    10 PRINT SIN(1)      : REM =.841470985
(ii)   20 PRINTSIN(360*<pi>/180): REM =0
(iii) 100 PRINT"<CLR>":FOR I=1 TO 22
       110 PRINT TAB(SIN(I/3)*9+10);"*":NEXT I
(iv)  200 X=A+SIN(A):Y=A+SIN(A/2)*2
```

Az első két példa önmagáért beszél. A harmadik program egy szinuszgörbét rajzol a képernyőre. A negyedik példa egy rajzoló program része.

Az argumentum nagysága a végeredmény pontosságát nem befolyásolja; a számítás során az interpreter ugyanis elosztja 2*pivel az argumentumot, és a maradékkal számol tovább.

Hibalehetőségek: Az aritmetikai kifejezés kiértékelése közben számos hibalehetőség adódik. Ha ez hiba nélkül befejeződött, a SIN rutin már hibátlanul lefut.

SPC (

Rövidítés: sP (a (már nem kell) *Token*: $A6(166)

Mód: mind parancs -, mind program módban használható.

Adott számú szóközt, vagy <CRSR JOBBRA> karaktert nyomtat.

Szintaxis: SPC(<aritmetikai kifejezés>)

Az utasítás csak a PRINT utasításban szerepelhet. Az SPC és a (-jel közt nem lehet szóköz. Az <aritmetikai kifejezés> értékének - lefelé kerekítés után - a 0-255 intervallumba kell esnie.

```
10 PRINT"<CLR>": FOR J=0 TO 20: PRINT"*";SPC(38)"*";:NEXT
20 FOR J=1 TO 19:PRINT SPC(J)"*"SPC(38-J*2)"*":NEXT
```

A fenti példák azt illusztrálják, hogy egy ciklusban használt SPC utasítás segítségével hogyan rajzolhatunk a képernyőre. Az első program egy keretet rajzol, a második pedig egy V alakú alakzatot.

Az, hogy a SPC szóközt vagy <CRSR JOBBRA> karaktert nyomtat, a programból állítható. Ha a 19.($13) címen 0 található akkor a SPC a kurzort mozgatja, ha ettől eltérő az érték, akkor szóközöket ír ki, ahogy ezt az alábbi program illusztrálja:

```
10 INPUT A
20 POKE 19,A
30 PRINT "<CLR>";:FOR J=0 TO 21: PRINT "X";:NEXT
40 PRINT "<HOME>**"TAB(7)"**"SPC(5)"**"
50 POKE 19,0
```

Az eredmény:

```
**      **      **XXXXX (A=1 esetén)
**XXXXX**XXXXX**XXXXX (A=0 esetén)
```

Hibalehetőségek: Ha az aritmetikai kifejezés értéke nem esik a 0-255 intervallumba ?ILLEGAL QUANTITY ERROR hibajelzést kapunk.

SQR

Rövidítés: sQ) *Token*: $BA(186)

Mód: mind parancs -, mind program módban használható.

Az argumentum négyzetgyökét számítja ki.

Szintaxis: SQR(<aritmetikai kifejezés>)

Példák:
```
(i)   PRINT SQR(2): REM = 1.4142...
(ii)  PRINT SQR(9)↑2: REM = 9
(iii) X1=(-B+SQR(B*B-4*A*C))/(2*A)
      X2=(-B-SQR(B*B-4*A*C))/(2*A)
(iv)  2000 T=SQR(X*X+Y*Y+Z*Z)
```

A SQR függvényre - éppen úgy, mint az EXP-re valójában nincs szükség, hiszen SQR(X)=X↑.5.

Az első két példa az SQR hatását mutatja be. A következő két sorban az A,B,C együtthatójú másodfokú egyenlet megoldásait számítottuk ki. Utolsó példánk az (X,Y,Z) pontnak az origótól vett távolságát adja.

Hibalehetőségek: Ha az argumentum értéke negatív, ?ILLEGAL QUANTITY ERROR hibajelzést kapunk. Vigyázzunk, sok programnyelvben az SQR - a C-64-től eltérően - a szám négyzetét számítja ki!

ST

Fenntartott BASIC változó.

ST értéke a perifériális egységek mindenkori állapotáról tájékoztat. ST jelentése az egység típusától függően más és más. A következő táblázat ST lehetséges értékeit és azok jelentését foglalja össze:

Bit érték	Kazettás egység I/O	Soros busz	Kazettás egység Verify+Load
0 1		idő túllépés	
1 2		idő túllépés	
2 4	rövid blokk		rövid blokk
3 8	hosszú blokk		hosszú blokk
4 16	hiba olvasási hiba		
5 32	ellenőrző összeg hiba		ellenőrző összeg hiba
6 64	file vége jel	file vége jel	EOT
7 -128		nincs bekapcsolva	

Az ST értéke a GET, INPUT és a PRINT, továbbá a CMD, GET#, INPUT# és a PRINT# utasítások megkezdése előtt nulla lesz. Ha a korábbi értékére a későbbiekben szükség lesz, akkor ST értékét egy másik változóba kell tárolni. Az ST-ben tárolt információt **valamennyi** I/O utasítás után ellenőrizni lehet.

ST nullától különböző értéke nem feltétlenül jelent hibát (pl. ST=64). Az ST értékét az interpreter egy byte-os számként a 144(\$90) címen tárolja.

STOP

Rövidítés: sT *Token*:\$90(144)

Mód: mind parancs -, mind program módban használható.

Megállítja a program futását és kiírja, hogy hányadik sorban állt meg. A program futása a CONT utasítással folytatható.

Szintaxis: STOP

Példák:

A STOP utasitást elsősorban a programfejlesztés stádiumában használjuk, hiszen kész programok esetén érdektelen, hogyan dolgozik a program, és hol is állt meg. A STOP utasítás segítségével töréspontokat helyezhetünk el a programban:

(i) 10 GOSUB 2000: STOP: GOSUB 3000: STOP
(ii) 1254 IF G\$="" THEN STOP

A STOP végrehajtása után a program változóit a PRINT utasítás segítségével ellenőrizhetjük. Ezután a CONT még működik. Ha egy régi vagy uj változónak adunk értéket a CONT általában még mindig használható. Ha újraszerkesztjük a programot, a CONT már nem használható, és a változók értéke is elvész.

Hibalehetőségek: Nincs.

STR$

Rövidítés: stR (a $ már nem kell!) *Token*: $C4(196)

Mód: mind parancs -, mind program módban használható.

Az argumentumként megadott aritmetikai kifejezés értékének sztring alakját adja meg.

Szintaxis: **STR$(**<aritmetikai kifejezés>**)**

Az STR és a $ közt nem lehet szóköz. Az <aritmetikai kifejezés> értéke a PRINT formátumának megfelelő alakban áll elő, leszámítva a szám végén levő <CRSR JOBBRA> karaktert, ami a STR$-ba nem kerül bele. Igy például STR$(.005) nem " .005", hanem " .5E-03".

Példák:
```
(i)   PRINT STR$(0.0024): REM =" 2.4E-03"
(ii)  10 PRINT STR$(555)+".00": REM =" 555.00"
(iii) 20 N$="0"+MID$(STR$(N),2): PRINT N$
```

Az utasítást elsősorban a számok alakjának szerkesztésére használjuk. Az első két példa egyszerűen csak szemlélteti a STR$ hatását. A (iii) példa az 1-nél kisebb számokat előnullázva írja ki. Például .05-t a program 0.05 alakban írja ki.

Hibalehetőségek: A számok normális írása és a STR$(N) alak nem mindig egyezik meg. Ebből szemantikus hibák adódhatnak. Az argumentum kiértékelése további hibák forrása lehet.

SYS

Rövidítés: sY *Token*: $9E(158)

Mód: mind parancs -, mind program módban használható.

A vezérlés a SYS argumentumaként megadott címen kezdődő **gépi-kódú alprogram**ra adódik át. A RTS gépi kódú utasítás végrehajtása után a vezérlés a BASIC-be kerül vissza.

Szintaxis: **SYS** <aritmetikai kifejezés>

Példák:
A SYS az egyik utasítás, amelyik segítségével gépi-kódú alprogramokat hívhatunk meg. A 4. fejezetben adunk egy példát, amelyik gépi kódú alprogram segítségével olvassa a billentyűzetet. A SYS 64738 parancs kiadása ekvivalens a számítógép ki-, és bekapcsolásával.

Hibalehetőségek Az utasítás hatására a program futásának ellenőrzése kikerül a BASIC interpreter hatálya alól. Ha a gépi-kódú rutint nem jól írtuk meg, a C-64 teljesen 'lemerevedhet'. Ilyenkor nincs mást tenni, mint kikapcsolni a gépet. SYS-t tartalmazó program kipróbálása előtt feltétlenül mentsük el a programot!

TAB(

Rövidítés: tA (a (már nem kell!) *Token*: $A3(163)

Mód: mind parancs -, mind program módban használható.

Amennyiben a kurzor pozíciója az adott sorban kevesebb, mint a TAB paraméterében megadott érték, addig a pozícióig szóközöket, vagy <CRSR JOBBRA> karaktereket nyomtat.

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

Az utasítás csak a PRINT utasításban szerepelhet. A TAB és (-jel közt nem lehet szóköz.

Példák:
```
(i)  10 PRINT TAB(40/2-LEN(N$)/2);N$
(ii) 20 PRINT"*";TAB(255);"*"
```

Az első példa a képernyő közepére tabulálja az N$-ban tárolt üzenetet. A következő példa azt illusztrálja, hogy a TAB hatása több soron keresztül is érvényesülhet.

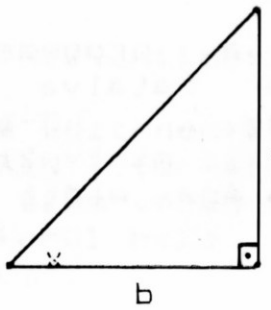
Hibalehetőségek: Ha az aritmetikai kifejezés értéke nem esik a 0–255 intervallumba, akkor ?ILLEGAL QUANTITY ERROR hibajelzést kapunk.

TAN

Rövidítés: nincs *Token*: $CO(192)

Mód: mind parancs –, mind program módban használható.

A radiánban megadott argumentum tangensét számítja ki. Az alábbi ábra mutatja, hogyan lehet egy háromszög oldalaiból a tg(x) függvényt kiszámítani:



$$tg(x)=\frac{a}{b}$$

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

Példák:

```
(i)   PRINT TAN(<pi>/2): REM ?OVERFLOW ERROR
(ii)  PRINT TAN(45/57.29578):REM = TG 45 fok=1
(iii) 30 X=(TAN(A)+TAN(B))/(1-TAN(A)*TAN(B))
```

Az első két példa a TAN hatását mutatja be, a másodikban a fokokban adott szöget először átszámítjuk radiánba. A harmadik példa a tangens függvényre vonatkozó addíciós tételt használja; X valójában TAN(A+B)-vel egyenlő.

Hibalehetőségek: Amikor cos(x) közelítőleg nulla, akkor ?OVERFLOW ERROR hibaüzenetet kaphatunk. (tg(x)=sin(x)/cos(x) miatt.)

TI és TI$

Fenntartott BASIC változók.

TI és TI$ a C-64 belső órájának segítségével a bekapcsolás óta eltelt időt mérik. TI valós változó, ami azonban csak olvasható. A 0. lapon található 'szabadon futó óra' másodpercenként körülbelül ötvenszer aktualizálódik, és TI/60 a gép bekapcsolása óta eltelt másodperceket mutatja. TI$ egy hat karakteres sztring, amelynek két-két karaktere rendre az órát, a percet és a másodpercet adja. TI$ kezdőértéke értékadó utasítással megadható.

```
(i)    TI$="081500" : REM (ha 1/4 9-kor kezdtünk el dolgozni.)
       ?TI$: REM mielőtt kikapcsoljuk a gépet.
(ii)   100 T=TI
       110 IF TI-T<120 THEN GOTO 110
(iii)  1000 PRINT MID$(TI$,1,2);":";MID$(TI$,3,2);":";
       1020 PRINT MID$(TI$,5,2)
(iv)   5000 TR=TI
       5010 <PROGRAM>
              ....
       8020 PRINT (TI-TR)/60 : END
```

Az első példa azt mutatja, hogy TI$ értékét magunk is megadhatjuk. A sztringnek, ami TI$ új értékét definiálja pontosan hat karakter hosszúnak kell lennie. A (ii) példa 2 másodpercnyi időre felfüggeszti a program futását (ciklusban várakozik).

A (iii) példa az időt a digitális órákon szokásos hh:mm:ss alakban jelzi ki. Az utolsó példa egy tetszőleges program futási idejének kiszámítására szolgál. A program futása előtt a TI-ben tárolt értéket a TR változóban elmenti, majd a program végét jelentő END végrehajtása előtt az azóta eltelt időt másodpercekben kiírja.

Hibalehetőségek: Ha olyan gépi-kódú rutint, vagy BASIC parancsot használunk, amelyik letiltja a megszakításokat, az 'óra' pontatlanná válik, késik.

USR

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>)

A USR függvény az aritmetikai kifejezés értékét az első lebegőpontos akkumulátorba helyezi el. Ezenkívül a 785-786 címen a USR függvény meghívása előtt egy gépikódú rutin címének kell lennie (a JMP utasítás a bekapcsoláskor a 784 címre kerül). A USR függvény a program vezérlését erre a címre adja át. Visszatéréskor a USR függvény értéke a lebegőpontos akkumulátor tartalma lesz.

Hibalehetőségek: A BASIC interpreter nem ellenőrzi a gépi-kódú program futását. A USR használatára ugyanaz vonatkozik mint, a SYS-re.

VAL

Rövidítés: vA *Token*: $C5(197)

Mód: mind parancs -, mind program módban használható.

Az argumentumként szereplő sztring kifejezés numerikus értékét számítja ki. A sztringet első - szám részeként már nem értelmezhető - karakteréig dolgozza fel.

Szintaxis: VAL(<sztring kifejezés>)

A <sztring kifejezés> értékeként egy legfeljebb 255 hosszú sztringet kell kapnunk.

Példák:

```
10 PRINT VAL("123.321")      : REM = 123.321
20 PRINT VAL("-123")         : REM = -123
30 PRINT VAL("1.2 E2")       : REM = 120
40 PRINT VAL("E")            : REM = 0
50 PRINT VAL("10000000000"): REM = 1E+10
60 PRINT VAL(LEFT$(TI$,2)):   REM = 0-24 KOZT
70 PRINT VAL("123+321")      : REM = 123
80 PRINT VAL("1.2.3")        : REM = 1.2
90 PRINT VAL("")             : REM = 0
```

```
100 PRINT VAL("1")+VAL("2"): REM = 3
```

Mint a fenti példák is mutatják, *bármi* is az argumentum értéke, a VAL függvény mindig előállít valamilyen számot. Ez gyakran lehet szemantikus hiba forrása. Utolsó példánk mutatja, hogy VAL aritmetikai függvény, és ezért tetszőleges aritmetikai kifejezésben használható.

A VAL felismeri a +, -, E jeleket, a tizedespontot (.), és ezenkívül természetesen a 0-9 számjegyeket. Az első - szám részeként - nem értelmezhető karakter befejezi a sztring kiértékelését.

Hibalehetőségek: Nincs.

VERIFY

Rövidítés: vE *Token*: $95(149)

Mód: mind parancs -, mind program módban használható.

Ellenőrzi, hogy valamely periférián elmentett program megegyezik-e a memóriában tárolt programmal.

Szintaxis:
VERIFY {<sztring kif.> {,<arit. kif.> {,<arit. kif.>}}}

Példák:

```
(i)   SAVE "PROG",8
      VERIFY "PROG",8
(ii)  SAVE "PROG" : REM CSEVELD VISSZA
      VERIFY "PROG"
```

Az (i) példában a PROG nevű programot kiírjuk a lemezre, majd ellenőrizzük, nem történt-e hiba. A VERIFY parancs kiadása után a következő üzeneteket kapjuk:

SEARCHING FOR "PROG"
VERIFYING

Ezeket egy OK vagy ?VERIFY ERROR üzenet követheti, attól függően, hogy a lemezes file tartalma megegyezik-e a memóriában tárolt programmal vagy sem.

A (ii) példa ugyanezt a programot kazettás file-ba menti el, s ezt ellenőrzi. A VERIFY parancs kiadása előtt a szalagot a program elé kell visszatekerni. A VERIFY "PROG" parancs kiadása

után a képernyőn a

PRESS PLAY ON TAPE

üzenet jelenik meg. A kazettás egység <PLAY> billentyűjének a lenyomása után megkezdődik a program megkeresése, majd az ellenőrzése.

Hibalehetőségek: Ha programból hajtjuk végre a VERIFY parancsot, és az OK-val végződik, a program fut tovább. Ha hibát észlel az interpreter, ?VERIFY ERROR hibajelzést kapunk, és a program futása megszakad. Ha a program nincs a lemezen, vagy a kazettán az interpreter egy EOT fejlécet olvas, akkor ?FILE NOT FOUND ERROR hibajelzést kapunk.

WAIT

Rövidítés: wA *Token*: $92(146)

Mód: mind parancs -, mind program módban használható.

Az utasítás felfüggeszti a program végrehajtását, míg a paramétereiben specifikált esemény be nem következik.

Szintaxis:
WAIT <arit. kif.>, <arit. kif.> [,<arit. kif.>]

Az első aritmetikai kifejezés a memória valamely címét jelenti, és így értékének a 0-65535 intervallumba kell esnie. A másik két aritmetikai kifejezés értékének a 0-255 intervallumba kell esnie.

A WAIT L,M1,M2 utasítás először kiszámítja a

(PEEK(L) EOR M2) AND M1

értékét. (EOR a **kizáró vagy.**) Ha az eredmény nullától különböző, a BASIC program folytatódik; ha 0, akkor az interpreter a fenti műveletet - akár a végtelenségig - ismétli. L-nek tehát olyan címnek kell lennie, amit a hardver megszakító rutin, vagy a perifériák használnak, és a benne tárolt értéket megváltoztathatják a kívánt módon. (Ha M2-t nem adtuk meg, akkor csak PEEK(L) AND M1 hajtódik végre.)

Az az esemény, amelyik a BASIC program futásának folytatását eredményezi, az L cím valamely bitjének (bitjeinek) magasra és/vagy alacsonyra állítódása lehet. Az M1-ben magasra kell állítani azokat a biteket, amelyek értékére kíváncsiak vagyunk,

a többit pedig alacsonyra. M2-ben azokat a biteket kell magasra állítani, amelyek értékének alacsonyra kell válnia a futás folytatásához. Ha például M1=00011000=24 és M2=00010000=16, akkor a program addig áll, mig az L cim 4. bitje alacsony és az 5. bitje magas nem lesz. Mihelyt a 4. bit magas vagy az 5.bit alacsony lesz, a program fut tovább.

Példák:

(i) 10 WAIT 198,255: GET A$: PRINT A$: GOTO 10
(ii) 120 WAIT 1,32,32

Az (i) példában a program addig vár, míg meg nem nyomunk egy billentyűt. Így a GET A$ utasítás sohasem olvas CHR$(0) értéket. Hasonlítsuk össze a futás eredményét a

10 GET A$: PRINT A$: GOTO 10

programmal! (Lásd a 6. fejezetben a billentyűzet pufferről szóló részt!)

A (ii) példánkban a program addig várakozik, mig a kazettás egységen valamelyik billentyűt le nem nyomjuk.

Hibalehetőségek: Egyes esetekben kellemetlen mellékhatások is jelentkezhetnek (pl. az interpreter eltéved) s a program így végtelen ciklusba kerül.

4.2 SIMONS' BASIC utasítások

Ebben a paragrafusban röviden összefoglaljuk a C-64 egyik legnépszerűbb BASIC kiterjesztésének, a SIMONS' BASIC-nek az utasításait. Az I/O egységekkel, a grafikus lehetőségekkel és a hanggenerálással kapcsolatos utasítások a 6., a 7., illetve a 8. fejezetekben szerepelnek. A SIMONS' BASIC többi alapszavát a következő részekre osztva ismertetjük:

- programszerkesztő utasítások,
- sztring függvények,
- input utasítások (a billentyűzetről),
- aritmetikai függvények,
- lemezegységre vonatkozó utasítások,
- vezérlésátadó utasítások,
- hibakezelő utasítások.

Programszerkesztő utasítások

KEY <sorszám>,<sztring>

Az utasítás az első paraméterként szereplő <sorszám>-u <f> billentyűhöz rendeli a <sztring> karaktersorozatot. A megfelelő <f> billentyű megnyomása ekvivalens lesz a <sztring> begépelésével. összesen 16 <f> billentyű létezik, ezek közül 8 a billentyűzeten látható. A többiek beírása a következő:

<C= - fi>

az <f i+8> billentyűnek számit. Az <f12> billentyűzése tehát <C= f4>.

A KEY 1,"RUN"+CHR$(13) parancs kiadása után az <f1> lenyomására a BASIC program azonnal elindul.

DISPLAY

Kilistázza, hogy az egyes <f> billentyűkhöz milyen sztringek tartoznak.

AUTO <kezdőszám>,<növekmény>

Automatikus sorszámozást biztosít a programsorok beírásához. Ha egy programsor beírása után megnyomjuk a <RETURN> billentyűt, akkor nemcsak a programsort tárolja az interpreter, hanem a képernyő következő sorának az elejére kiírja a következő sorszámot. Az automatikus sorszámozást egy üres sor bevitelével leállíthatjuk.

RENUMBER <kezdőszám>,<növekmény>

Újraszámozza a programot. A GOTO, GOSUB hivatkozásokat **nem** módosítja, így azok az átszámozás után rossz helyre mutatnak!

PAUSE {<sztring>,} <aritmetikai kifejezés>

Az utasítás hatására a <sztring> kiíródik a képernyőre, és a program futása az <aritmetikai kifejezés> értékének megfelelő ideig fefüggesztődik. Az időt másodpercekben kell megadni. A <RETURN> lenyomásával a program futása az idő letelte előtt folytatható.

RESET <sorszám>

Az utasítás a DATA mutatót a paraméterben specifikált sorra állítja. A következő READ ebből a sorból kezdi el az adatok olvasását.

MERGE <név>,<egységszám>

A MERGE paramétereinek jelentése azonos a LOAD utasítás paramétereivel. Az utasítás a <név> nevű file-ban levő programsorokat beszúrja a C-64 memóriájában tárolt program sorai közé.

PAGE n

Az utasítás megadja, hogy a LIST hányasával listázza ki a programsorokat. n értékét 1-nél nagyobbra állítva a LIST hatására csak az első n programsor jelenik meg a képernyőn. A <RETURN> lenyomása n-esével folytatja a listázást. A <STOP> szokás szerint megállítja a listázást. PAGE 0 a normális listázást állítja vissza.

OPTION n

Ha n értéke 10, az ezt követően kiadott LIST utasítások a SIMONS' BASIC alapszavait **inverz formában** írják ki a képernyőre vagy a nyomtatóra. Ha a paraméter értéke ettől eltérő (0<n<255) akkor a listázás normális. (A C-64 eredeti BASIC alapszavai *nem* inverz alakban íródnak ki!)

DELAY n

A listázás sebességét adja meg. 0<n<255 lehet. A <SHIFT> billentyű lenyomásával hatása megszűnik.

FIND<sztring>

Kilistázza a program azon sorainak számát, amiben a <sztring> szerepel.

TRACE n

Ha n értéke 10, az interpreter nyomkövetési üzemmódban kezd el dolgozni, és a programok futása közben a képernyőn egy 'ablak' jelenik meg, amiben az éppen végrehajtott utasítások sorszáma látszik, egyszerre maximum 6. A nyomkövetési üzemmódot n bármely más értéke megszünteti.

RETRACE

A program szerkesztése után újból kijelzi az utoljára végrehajtott utasítások sorszámát.

DUMP

Kiírja a memóriában tárolt változók értékét

<változónév>=<érték>

alakban.

COLD

SIMONS' BASIC hideg inditása. Hatására a C-64 újra végrehajtja a teljes reset ciklust.

DISAPA:

Az ezzel az utasítással kezdődő sorok a SECURE 0 végrehajtása után nem lesznek listázhatók.

SECURE 0

Az utasítás végrehajtása után a DISAPA:-val kezdődő sorok nem látszanak a listázáskor. Nem hatástalanítható. Például

```
10 DISAPA:PRINT"EZ ITT VAN"
SECURE 0:LIST
```

hatására csak

10

jelenik meg a képernyőn.

OLD

Hatástalanítja a NEW utasítást.

Sztring függvények

INSERT (<sztring>,<sztring>,<aritmetikai kifejezés>)

A függvény értéke a harmadik paraméterként megadott sorszámú karaktertől kezdődően a második <sztring>-be beszúrt első <sztring>.

INST (<sztring>,<sztring>,<aritmetikai kifejezés>)

A függvény értéke a második paraméterként szereplő <sztring> karakterei, az <aritmetikai kifejezés>-ben megadott helytől kezdődően az első <sztring>-re cserélve.

PLACE (<sztring>,<sztring>)

Az interpreter ellenőrzi, hogy az első paraméterként szereplő <sztring> előfordul-e a második <sztring>-ben. Ha igen, a függvény értéke az első előfordulás első karakterhelyének száma lesz. Ha 0, akkor az első <sztring> nem fordul elő a második <sztring>-ben.

DUP (<sztring>,<aritmetikai kifejezés>)

A függvény értéke a <sztring> karakterei, a második paraméter értékének megfelelő sokszor ismételve.

Input/Output utasítások

CENTRE <sztring>

Az utasítás a következő sor közepére kiírja a <sztring>-nek megfelelő szöveget.

AT (<aritmetikai kifejezés>,<aritmetikai kifejezés>)

A PRINT utasításban **elválasztó jel**ként a TAB, SPC stb. mellett az AT utasítás is szerepelhet. Hatására a nyomtatás az AT-ben specifikált karakterhelytől folytatódik.

USE <nyomtatási kép>,<változó>:PRINT

Az utasítás hatására a <változó> sztring kifejezés a <nyomtatási kép>-nek megfelelő formában kerül kiírásra. Ennek megfelelően a kiíratni kívánt számot előbb a STR$ függvény segítségével karakteres alakra kell konvertálnunk. Ha a PRINT után szerepel a pontosvessző vagy a vessző, akkor nem kerül spr automatikus kocsi-vissza-soremelés kiírására.

A <nyomtatási kép> valójában egy sztring, amelyben szereplő # és . karaktereknek speciális jelentése van. A # jel a kiírandó sztring tizedespont előtti, illetve utáni számjegyeit jelenti. Például a

```
X$=STR$(X)
USE "###.#####",X$: PRINT
```

az X számot ugy irja ki, hogy az egész rész 3, a törtrész 5 karakterhelyet foglal el. A sztringben további karakterek is lehetnek, ezek a megfelelő helyen kiíródnak. Például

```
X$=STR$(X)
USE "$#####",X$:PRINT
```

hatása X=2, 123, 1544 illetve 12331 esetén rendre a következő:

```
$    2
$  123
$1544
$2331
```

A USE "# # # #.#",X$:PRINT az X$="1236.58 értéket 16263666.66 alakban írja ki.

FETCH "<sztring>",<aritmetikai kifejezés>,<változó>

Az utasítás hatása végső soron megegyezik az INPUT <változó> hatásával; a <sztring> és <aritmetikai kifejezés> értékének megadásával azonban mód van az INPUT ellenőrzésére. A FETCH nem ad ? jelet és a villogó kurzor sem jelenik meg. A <sztring> a beírható karakterek típusát jelenti:

<sztring>	**beírható karakterek**
"<Home>"	nagybetűk, CHR$(65)-CHR$(90)
"<CRSR LE>"	CHR$(32)-CHR$(64)
"<CRSR JOBBRA>	kis- és nagybetűk
"ABCDE"	csak az A,B,C,D,E karakterek

Az <aritmetikai kifejezés> az inputsor maximális hosszát adja meg. Ennél több karaktert az interpreter nem fogad el. Az inputsor bevitelét a <RETURN> megnyomásával kell befejezni.

<változó> = **INKEY**

Az értékadás eredménye az éppen benyomott <f> billentyű indexe lesz (például 3, ha az <f3>-at nyomtuk meg).

ON KEY <sztring>,: GOTO<sorszám>

Az utasítás ellenőrzi, hogy valamely, a <sztring>-ben szereplő karakternek megfelelő billentyűt megnyomtuk-e. Ha igen, a vezérlésátadás végrehajtódik. Például a

```
300 ON KEY "1234567890",:GOTO 400
310 GOTO 300
```

programrész ciklusban várakozik addig, míg egy számjegy billentyűt le nem nyomunk. (Lásd még az alábbi két utasítást is!)

DISABLE

Az ON KEY utasítás az interpreter megszakító rendszerébe avatkozik bele, ezért mihelyt a megfelelő billentyűt megnyomtuk, célszerű ezt a hatást megszüntetni. Erre szolgál a DISABLE utasítás. A fenti példa esetében a 400. sor általában egy

DISABLE utasítással kell, hogy kezdődjön.

RESUME

Feltétel nélküli vezérlésátadás az utoljára végrehajtott ON KEY utasításra.

Aritmetikai függvények

MOD (X,Y)

A függvény elosztja X-et Y-nal és a maradékkal tér vissza.

DIV (X,Y)

A legnagyobb olyan egész számmal tér vissza, amelyet Y-nal megszorozva még nem kapunk X-nél nagyobb számot. (=INT(X/Y))

FRAC (X)

X törtrésze (=X-INT(X)).

A SIMONS' BASIC lehetővé teszi bináris és hexadecimális konstansok használatát. Ezek pontosan 8 és 4 bináris, illetve hexadecimális jegyből kell, hogy álljanak. Például

```
%00001011  (=11)
$00AF      (=175)
```

EXOR (X,Y)

A két byte-os számmá konvertált X és Y bitjeire bitenként végrehajtja a 'kizáró vagy' logikai műveletet.

Lemezegységre vonatkozó utasítások

DISK <parancs>

Az utasítás hatása ekvivalens a következővel

```
OPEN 15,8,15
PRINT#15,<parancs>
CLOSE 15
```

DIR "\$ vagy **DIR "\$:<szöveg>**

Az utasítás hatására a lemez katalógusában szereplő összes, vagy annak a szöveggel egyező nevű file-jainak adatai a képernyőre kerülnek. A ? és a * jelek használata a szokásos (lásd az 5. fejezetben).

Vezérlésátadó utasítások

GOTO <aritmetikai kifejezés>

Kiszámított GOTO utasítás. Az <aritmetikai kifejezés> értékének megfelelő sorszámú sorral folytatódik a program végrehajtása.

IF <logikai kifejezés> THEN <utasítás> **:ELSE:** <utasítás>

Az ELSE alapszóval bővített feltételes vezérlésátadó utasítás lehetővé teszi, hogy a <logikai kifejezés>=hamis esetben a vezérlés ne a következő programsorra, hanem az ELSE-t követő utasításra kerüljön.

REPEAT...UNTIL

Az utasításpár egy, a FOR...NEXT-től eltérő ciklusszervezési lehetőséget biztosít. A REPEAT utasítás alakja:

REPEAT <utasítás>

Az UNTIL alakja:

UNTIL <logikai kifejezés>

A REPEAT utasítás végrehajtásakor a verembe kerül a visszatérést biztosító CHRGET mutató és a REPEAT tokenje; majd a REPEAT-et követő <utasítás> hajtódik végre. Az UNTIL végrehajatása a <logikai kifejezés> kiértékelésével kezdődik. Ha igaz, a program végrehajtása folytatódik; ha nem, akkor a vezérlés az utoljára végrehajtott REPEAT-ben szereplő <utasítás> végrehajtásával folytatódik. Ezt az interpreter úgy éri el, hogy a vermet végignézi, megkeresi a REPEAT tokent és visszaállítja a CHRGET mutatóját.

RCOMP: <utasítás> {:**ELSE:** <utasítás>}

A legutóbbi IF...THEN...ELSE utasítás feltételét értékeli ki újra. Ha igaz, az első utasítás hajtódik végre, ha nem, a második. Ha az ELSE hiányzik, akkor hamis feltétel esetén a vezérlés a következő programsorra kerül.

LOOP...EXIT IF...END LOOP

A három egymással szorosan összefüggő utasítás lehetőséget biztosít a ciklusból bármelyik pontján történő kilépésre. Az egyes utasítások alakja a következő:

```
LOOP
    <utasítások>
EXIT IF <logikai kifejezés>
    <utasítások>
END LOOP
```

A LOOP utasítás hatására a verembe kerül a visszatérést biztosító CHRGET mutató, a programsor száma és a LOOP tokenje, majd a LOOP utáni utasításra kerül a vezérlés.

Az EXIT IF utasítás végrehajtása a <logikai kifejezés> kiértékelésével kezdődik. Ha hamis, akkor a következő utasításra kerül a vezérlés. Ha igaz, akkor az interpreter az utasítás sorszámánál nagyobb számú programsorokban elkezdi az END LOOP utasítás keresését. Az első END LOOP utáni utasításra kerül a vezérlés.

Az END LOOP utasítás az utoljára végrehajtott LOOP utáni első utasításra adja a vezérlést. Ehhez az interpreter végignézi a vermet, és megkeresi az első LOOP tokent. Ezután a CHRGET mutatóját a LOOP utánra állítja.

PROC <név>

Az utasítás az utána következő első utasítást a <név> **címkével** látja el. A SIMONS' BASIC két alábbi utasításával lehet az így kapott címkékre hivatkozni:

CALL <név>

Feltétel nélküli vezérlésátadás a <név> címkével megjelölt programsorra.

EXEC <név>

A <név> címkéjű alprogram végrehajtása. A verembe kerül a programsor száma, a CHRGET mutató és a EXEC tokenje. Az alprogramot **nem** a RETURN, hanem az alábbi utasítással kell befejezni:

END PROC

Az utasítás megkeresi a veremben található első EXEC tokent és az ott talált adatoknak megfelelő helyről folytatja a program végrehajtását. (Nem helyettesíthető a RETURN utasítással!)

LOCAL <változólista>

Az interpreter elmenti a <változólista>-ban szereplő változók értékét a legközelebbi GLOBAL utasitás végrehajtásáig. Addig a változólistában szereplő változókat - más célokra - tovább használhatjuk.

GLOBAL

A legutolsó LOCAL utasitásban használt változók elmentett értékeit visszatölti. A LOCAL-GLOBAL utasitáspárt általában az alábbi környezetben használjuk:

```
PROC <név>
LOCAL <változólista>

<az eljárás törzse>

GLOBAL
END PROC
```

Hibakezelő utasitások

A C-64 interpreter hiba esetén megszakitja a program futását, és kiírja a megfelelő hibaüzenetet. Néha azonban hasznos, ha a programon belül is kezelni tudjuk a felmerült hibákat. Erre szolgálnak az alábbi utasítások.

ON ERROR: <utasítás>

Az utasítás végrehajtása annyit jelent, hogy az interpreter tudomásul veszi, hogy hiba esetén az <utasítás>-t kell a

programnak végrehajtania.

ERRN

Fenntartott változó, ami hiba esetén a hiba számát (kódját) tartalmazza.

ERRL

Fenntartott változó, ami hiba esetén a hibát okozó programsor számát tartalmazza.

NO ERROR

Az utasítás megszünteti a hibakezelő rutin hatását. Ezután hiba esetén a program futása megszakad, és a C-64 eredeti hibakezelő rutinja hajtódik végre.

OUT

Ha az utasítás kiadása után hiba történik, az interpreter kiírja a C-64 hibaüzenetét, és megszakítja a program futását, az ON ERROR hatását azonban **nem szünteti** meg. Például CONT-tal folytatva a programot a hibakezelés az ON ERROR-nak megfelelően történik.

Végül egy összetett példát mutatunk be, amelyik elsősorban a SIMONS' BASIC új vezérlésátadó utasításainak a használatát mutatja be. Az eljáráshívások használata nagymértékben lassítja a program futását. Ugyanez a program a GOSUB, GOTO utasítások használatával 4-5-ször gyorsabb.

A program a **Hanoi tornyai** néven ismert játék megoldását irja ki a képernyőre. A feladat a következő. Egy pálcikán N darab egyre kisebb és kisebb átmérőjű korong helyezkedik el, legalul a legnagyobb. Ezeket a korongokat kell egy másik pálcára a következő feltételek mellett - ugyanebben a sorrendben- áthelyezni:

a/ egyszerre csak egy korongot rakhatunk át;
b/ kisebb korongra nem rakhatunk nagyobbat;
c/ az átrakáshoz egy harmadik pálcikát is használhatunk.

Egy **lehetséges** megoldás:

```
100 REM ******************
110 REM * HANOI TORNYAI *
120 REM ******************
130 :
140 REM KEZDOERTEKEK
150 K$(1)="1":K$(2)="2":K$(3)="3"
160 MAX=100: VM=0
170 DIM I(MAX), J(MAX), D(MAX)
180 :
190 REM KORONGOK SZAMA
200 INPUT "<CLR>KORONGOK SZAMA";N: PRINT"<CLR>";
210 :
220 REM 1-->2,N VEREMBE TOLTESE
230 I1=1:J1=2:D1=N: EXEC PUSH
240 :
250 REM KOVETKEZO LEPES KIVETELE
260 REM HA URES, AKKOR VEGE
270 LOOP:  EXEC POP
280 EXIT IF URES
290 REM HELYETTESITO LEPESEK VISSZATOLTESE
300 IF D=1 THEN CALL KIIRAS
310 K=6-I-J
320 I1=K: J1=J: D1=D-1: EXEC PUSH
330 EXIT IF TELE
340 I1=I: J1=J: D1=1: EXEC PUSH
350 EXIT IF TELE
360 I1=I: J1=K: D1=D-1: EXEC PUSH
370 EXIT IF TELE
380 END LOOP
390 IF TELE THEN CALL HIBA
400 WAIT 198,255: GET A$: END
410 :
420 REM VEREMBE RAKAS
430 PROC PUSH
440 VM=VM+1:IF VM=MAX+1 THEN TELE=-1: END PROC
450 TELE=0: URES=0
460 I(VM)=I1: J(VM)=J1: D(VM)=D1
470 END PROC
480 :
490 REM VEREMBOL VALO KIVETEL
500 PROC POP
510 IF VM=0 THEN URES=-1: END PROC
520 URES=0: TELE=0
530 I=I(VM): J=J(VM): D=D(VM)
540 VM=VM-1: END PROC
550 :
```

```
560 PROC KIIRAS
570 S=S+1:IF S<=25 THEN CALL OSZLOP
580 S=1: X=X+5: IF X<40 THEN PRINT "<HOME>";:CALL OSZLOP
590 WAIT 198,255: GET A$
600 PRINT "<CLR>";: X=0
610 PROC OSZLOP
620 PRINT TAB(X);K$(I);"->";K$(J);
630 IF S<>25 THEN PRINT
640 END LOOP
650 :
660 PROC HIBA
670 PRINT"TELE A VEREM": END
```

5. fejezet

A lemezegység használata

5.1 A Commodore lemezegységek felépítése

Ebben a fejezetben a C-64 számítógéphez csatlakoztatható Commodore gyártmányú lemezegységek használatát ismertetjük.

A legtöbb hajlékonylemezes egység egy vagy több író/olvasó, illetve törlő fejjel rendelkezik, amelyeket egy léptető motor segítségével lehet a lemez megfelelő sávja fölé pozícionálni. A motor mindig sugárirányban mozgatja a fejeket, egy-egy lépés nagysága megközelítőleg 1/200 inch. A lemezen annak a sávnak a szélessége, amelyen az írás végülis történik, körülbelül 1/48 és 1/100 inch között változik. A lemezegység motorja nagy sebességgel forgatja a lemezt, amely a centrifugális erő hatására elveszti hajlékony jellegét, és így lehetővé válik az író/olvasó fej mozgatása.

A 'kemény' vagy Winchester lemezegységek használata nem tér el a floppy lemezegységek használatától. Felépítésben azonban különböznek. A Winchester típusú lemezegység nem cserélhető. Több lemez forog együtt egy légritka térben amelyben az író/olvasó fejeket is elhelyezték. Emiatt a speciális kezelés miatt sokkal sűrűbb tárolás valósítható meg, mint a hajlékony lemezek esetén. Mivel a Winchester nem cserélhető, a tárolási elvek különböznek egy kicsit. A Winchester lemezegységek például külön tárolják a hibás blokkok adatait, katalógusuk akármilyen hosszú lehet stb.

A lemezegységek számos áramkört, érzékelőt tartalmaznak, amelyek lehetővé teszik a léptető motor vezérlését, az írást engedélyező rés meglétének ellenőrzését, biztosítják a lemezegység adott sávjának olvasását stb. A Commodore lemezegységek egy önálló 16K-s operációs rendszert(DOS) tartalmaznak, amelyek a lemezegység kezelésén túlmenően a C-64-gyel való kommunikációt is biztosítják. A DOS használata nagyfokú önállóságot eredményez, ami lehetővé teszi, hogy a központi egységtől függetlenül is dolgozhasson. Elérhető például, hogy a lemezegység egy file-t a sornyomtatón listázzon, miközben a központi egység valami más feladatot végez.

A tárgyalt lemezegységek közül egyedül a VIC 1541-es csatlakoztatható közvetlenül a C-64 soros buszára. A többi lemezegység az IEEE 488-as szabvány szerint kommunikál. Ebben az esetben a megfelelő illesztő kártyáról külön kell gondoskodnunk. Mind a soros busz, mind az IEEE 488 ugyanazon logika szerint dolgozik, a különbség csak annyi, hogy a soros busz egyetlen vonalat használ adatátvitelre, míg az IEEE 488 nyolcat. (A kommunikáció részleteiről a 9. fejezetben, a KERNAL rutinok tárgyalásánál szólunk. A soros busz időzítéseit megtaláljuk a Függelékben.) Ezért a soros busz lényegesen lassabb, mint az IEEE 488-as párhuzamos busz.

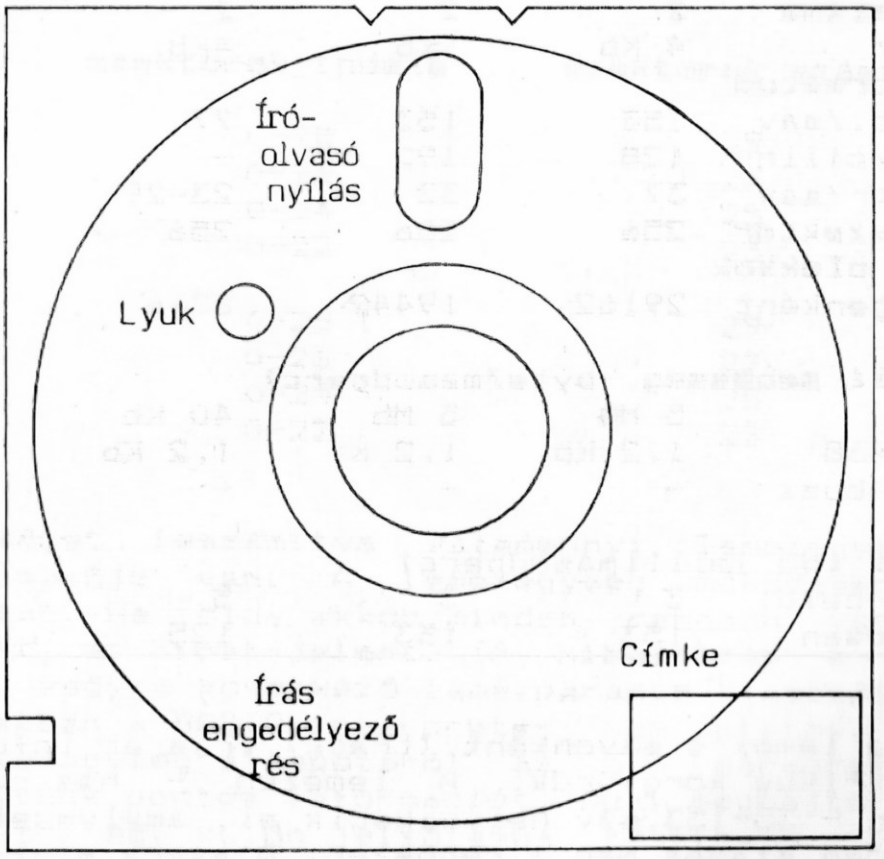
A VIC 1541-es lemezegységgel együtt szállítják a C-64 és a lemezegység közti adatkábelt is. Ugyanezzel a kábellel VIC 1541 típusu lemezegységeket, illetve soros kimenettel rendelkező nyomtatókat és egyéb perifériákat is összeköthetjük. A soros buszon maximum öt eszköz lehet.

Ha IEEE 488-as szabványú lemezegységet vásároltunk, akkor a megfelelő adatkábeleket külön kell megvásárolnunk. A lemezegység és a számítógép, illetve két lemezegység összekötéséhez más és más kábel kell. Ha C-64-hez kötjük az IEEE 488-as lemezegységeket, akkor az adatkábelt közvetlenül az interface kártyával, nem pedig a géppel kell összekötni.

A Commodore cég lemezegységeinek a hardver száma 8. Ha több lemezegységet akarunk egyszerre használni, akkor ezeket vagy szoftver, vagy hardver úton át kell számoznunk.

Az alábbi ábra egy hajlékonylemez (floppy) vázlata. A négyzet a lemezt magában foglaló borítékot szemlélteti, a kör alakú rész a mágneses adathordozó valódi (de nem látható) szélét mutatja. Az írásvédő rés a lemezek fizikai védelmét 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 lemezegység típusától függően vagy a felső, vagy mindkét oldalon történik a rögzítés.

A behelyezés iránya



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.

A Commodore lemezegységek adatai

	D9090	D9060	8250	1001	1541
Típus	Winch.	Winch.	floppi	floppi	floppi
Meghajtó	1	1	2	1	1
író/Olvasó fejek/megh.	6	4	2	2	1
Tároló kapacitás					
formázva	7.47 Mb	4.98 Mb	2.12 Mb	1.05 Mb	170 Kb
Legnagyobb					
SEQ file	7.41 Mb	4.94 Mb	1.05 Mb	1.05 Mb	168 Kb
REL file	7.35 Mb	4.90 Mb	1.04 Mb	1.04 Mb	167 Kb
DOS rendszer					
proc.száma	2	2	2	2	1
puffer	4 Kb	4Kb	4Kb	4Kb	2Kb
Lemezformátum					
cilind./sáv	153	153	77	77	35
szek./cilind.	128	192	-	-	-
szektor/sáv	32	32	23-29	23-29	17-21
byte/szektor	256	256	256	256	256
Szabad blokkok					
egységenként	29162	19442	8266	4133	664
Átviteli sebesség (byte/másodperc)					
belső	5 Mb	5 Mb	40 Kb	40 Kb	40 Kb
IEEE 488	1.2 Kb	1.2 Kb	1.2 Kb	1.2 Kb	-
soros busz	-	-	-	-	0.15 Kb
Elérési idő (millimásodperc)					
sávon belül	3	3	5	5	30
átlagosan	153	153	125	125	360

A DOS a lemezre **sávonként** (track) írja az információt. A sávok koncentrikus körgyűrűk. A lemezen - használt lemezegységtől függően - 35-153 sáv helyezkedik el, amelyeket kívülről 1-gyel kezdődően számoz meg a rendszer. A sávra szeletenként, szektoronként (sector) egy blokknyi (256 byte) információ kerül felírásra úgy, ahogy ez az alábbi ábrán látható:

Egy blokkba (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 hajlékonylemezes egységei - eltérően a legtöbb gyártótól - a külső (alacsonyabb sorszámú) sávokban sűrűbben írnak, és így ott több szektor fér el:

VIC 1541

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

CBM 8250/SFD 1001

sáv(track) sorszáma	szektorok indexe	szektorok száma
1- 39	0-28	29
40- 53	0-26	27
54- 64	0-24	25
65- 77	0-22	23
78-116	0-28	29
117-130	0-26	27
131-141	0-24	25
142-154	0-22	23

A VIC 1541 lemezegységet leszámítva valamennyi lemezegységnek egy kétszínű LED kijelzője van. A lemezegység működéséről a lámpa színe tájékoztat. Ha zöld, akkor minden rendben van. Ha tartósan pirosra vált, az hibát jelent. (A hibajelzés a hibacsatorna olvasásáig, vagy a következő lemezparancs kiadásáig nem törlődik.) Ilyen esetben a BASIC interpreter nem biztos, hogy jelzi a hibát! A lemezegység állapotáról az 5.3 paragrafusban leírt módon szerezhetünk pontos információt. Duál meghajtó esetén mindegyik meghajtón egy külön jelzőlámpa található. Amikor ez kigyullad, akkor a meghajtóban levő lemez és a lemezegység között adatátvitel zajlik. A VIC 1541-es lemezegység előlapján két lámpa található. A zöld színű a tápfeszültség meglétét

jelzi. A piros színű pedig a lemez és a lemezegység közti adatátvitelt. A hiba jelzésére a piros lámpa folyamatos villogása szolgál.

5.2 Tárolási alapelvek

A DOS a lemezen tárolt adatokról nyilvántartást (katalógus) vezet, ami rögzíti, 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 katalógusában (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 az ún. katalógusban tárolja. Ezek az információk a következők:

a/ a lemez neve és azonosító száma;
b/ a lemez formája;
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 blokkok foglaltságának jelzésére a katalógus elején fentartott byte-ok szolgálnak. A katalógusnak ezt a részét BAM-nak hívjuk az angol elnevezés (block availability map) alapján. A BAM-ok konkrét struktúrája lemezegységenként eltér.

File-ok azonosítására a nevük szolgál. Normál körülmények közt egy lemezen két azonos nevű file akkor sem lehet, ha különböző a típusa. A file-ok nevét a BASIC utasításokban rövidíthetjük. Ha egy file név *-ra végződik, akkor azt a lemezegység az azzal a névvel kezdődő összes file-lal azonosítja. Egy törlő utasítás például az összeset letörli. Ha egy ilyen nevű file-t betöltünk, akkor az ilyen névvel kezdődők közül a katalógusban először szereplőt tölti be.

A névben szereplő kérdőjelek helyén tetszőleges karakterek állhatnak. A ????.LST például egyaránt vonatkozik a

KOMP.LST
PROB.LST
PRG1.LST

file-okra, de nem vonatkozik a KOM.LST vagy az LST.KUKA fileokra.

A katalógusból a "DEMO"-val kezdődő összes file neve beolvastatható a memóriába a

LOAD "$:DEMO*",8

paranccsal.

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-, illetve szektor számát adja meg. A file-hoz tartozó első blokk adatait a DOS a katalógusban találja 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.

Két meghajtóval rendelkező lemezegységek esetén minden egyes, a lemezegységgel kapcsolatos művelet előtt meg kell adni, hogy melyikre vonatkozik. Ennek hiányában a művelet mindig az utoljára kiválasztott meghajtóra vonatkozik. A LOAD utasítás - ha nem adtuk meg a meghajtó számát - először a 0-ás meghajtóban levő lemezen keresi a file-t, s ha ott nem találja, csak akkor fordul az 1-es meghajtóhoz. A LOAD "$",8 utasítás **mindkét** katalógust betölti.

5.3 A 15. csatorna használata

A lemezzel történő adatcseréhez a file nevének 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 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., az 1. és a 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 csatornának** hívjuk. Ezen keresztül adhatunk parancsokat a DOS-nak, illetve kaphatunk információt a lemezegység állapotáról. Ha a lemezegység hibát észlel, azt az előlapon látható **piros lámpa kigyulladása** jelzi. (A VIC 1541-es lemezegység esetén a piros lámpa ilyenkor **villog.**) Előfordulhat, hogy a BASIC **nem** ad hibajelzést. Ilyenkor csak a hibacsatorna olvasásával kaphatunk információt arról, milyen hiba is történt. A hibacsatornát - az INPUT# használata miatt - csak program módban tudjuk olvasni.

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 nem 8 a lemezegység hardver száma, akkor értelemszerűen a 9-11 számok valamelyikét kell használnunk.

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(<d>):<név>,<id>"   vagy röviditve:
PRINT#15,"N(<d>):<név>,<id>"
```

<d> a meghajtó sorszáma. Hiányában a művelet az utoljára kiválasztott meghajtóra vonatkozik. A <név> karaktersorozat a lemez neve lesz. A <név> csak a katalógusba kerül beírásra, míg az <id>, pontosan **két karakteres azonosító** valamennyi blokkba. Ha két kiíró utasítás között kicseréljük a lemezt, a DOS érzékeli, hogy új <id> azonosítójú lemezzel dolgozik és hibát jelez. A CBM 8250, illetve SFD 1001 lemezegység ilyenkor automatikusan újraolvassa a BAM-ot.

A fenti utasítás PRINT#15,"NO:<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 lemezen levő adatokra már tényleg nincs szükségünk.

A meghajtó 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 (illetve annak valamelyik meghajtóját) inicializálni, melynek hatására a DOS lezárja a megnyitott file-okat és újra olvassa a BAM-ot.

PRINT#15,"INITIALIZE[<d>]" vagy rövidítve:
PRINT#15,"I[<d>]"

A <d> hiányában mind a két meghajtót inicializálja a lemezegység.

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[<d>]:<név>" vagy
PRINT#15,"S[<d>]:<név>"

A <d> a meghajtó sorszámát jelenti. Hiányában az utoljára kijelölt meghajtóra vonatkozik a törlő utasítás. A <név>-ben 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, hogy lényegtelen, 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 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[<d>]" vagy rövidítve a
PRINT#15,"V[<d>]"

utasítást használhatjuk. Az utasítás hatására a DOS újraszervezi

a <d> meghajtóban levő lemezen levő file-okat, anélkül, hogy azok tartalmát megváltoztatná. A VALIDATE DOS-parancs **törli a véletlen-file blokkjait**, ezért olyan lemez esetén, amelyik tartalmaz véletlen-file-t is, semmiképp se használjuk.

File-ok átnevezése

Néha szükség lehet arra, hogy egy file-nak megváltoztassuk a nevét. Ezt a következő utasítással érhetjük el:

```
PRINT#15,"RENAME{<d>}:<új név>={<d>}:<régi név>"
PRINT#15,"R{<d>}:<új név>={<d>}<régi név>"
```

File-ok másolása és összefűzése

Lehetőségünk van egy szekvenciális file másolatát ugyanazon a lemezen, de más név alatt létrehozni:

```
PRINT#15,"COPY{<d>}:<új név>={<d>}:<régi név>" vagy
PRINT#15,"C{<d>}:<új név>={<d>}:<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{<d>}:<új név>={<d>}:<r.név1>,{<d>}:<r.név2>,
                                {<d>}:<r.név3>,{<d>}:<r.név4>"
```

A hibacsatorna olvasása

A 15.csatorna az

```
INPUT#15,A$,B$,C$,D$ E,E$
```

utasítással olvasható. Az A$, B$, C$, D$, E$ 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;
E$=a meghajtó száma.

Az E$ változót **csak** duál lemezegységek esetén érdemes használni.

Az A$, C$, D$, E$ csak számjegyeket tartalmaz, ezért az INPUT utasításban aritmetikai változókkal helyettesíthetjük őket:

```
INPUT#15, E1,E2$,E3,E4,E5
```

Lemez másolása
Duál lemezegység esetén lehetőség van az egyik meghajtóban levő lemez **teljes** másolására. Ezt a következő parancs kiadásával végezhetjük el:

PRINT#15,"DUPLICATE<d1>=<d2>" vagy röviden
PRINT#15,"D<d1>=<d2>"

Az utasítás hatására a <d2> meghajtóban levő lemez tartalma fizikailag teljes egészében átmásolódik a <d1> meghajtóban levő lemezre. A <d1> lemez eredeti tartalma teljes egészében elvész. Például PRINT#15,"D0=1" hatására az 1-es meghajtóban levő lemezt másoljuk át a 0-ás meghajtóban levő lemezre.

5.4 Programok tárolása

Programok lemezen való tárolására, illetve visszatöltésére 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 száma,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 pozíciójától kezdődően töltődik be.

A file keresése duál meghajtók esetén a 0. meghajtóban levő lemezen kezdődik, s ha azon nincs, akkor az 1. meghajtóban levővel folytatódik. Abban az esetben, ha csak az egyik meghajtóban levő lemezen akarjuk keresni, a <név> sztringnek a "0:", illetve "1:" sztringgel kell kezdődnie.

A katalógus a LOAD "$",8 utasítással tölthető be, majd a LIST utasítással írható ki a képernyőre.

A LOAD részletes ismertetése a 4.1 paragrafusban található meg.

Programok elmentése
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. Nincs mód a memória valamely más részének az elmentésére. Ha a programot valamelyik meghatározott meghajtón levő lemezre akarjuk kimenteni, akkor a <név> sztringnek a "0:", illetve "1:" sztringgel kell kezdődnie. Ennek hiányában a program mindig a 0-s meghajtóban levő lemezre íródik ki.

Az utasítás - feltéve, hogy <név> nevű file még nincs a lemezen - létrehoz egy új, <név> nevű file-t, és a memóriában tárolt program tartalmát elmenti a file-ba. Ha a file már létezik, a mentés nem történik meg. Hibajelzést nem kapunk csak a lemezegység piros lámpája villog. Ezt elkerülendő a SAVE első paraméterét

"@0:<név>"

alakban kell megadnunk. Ennek hatására a DOS először felírja az új file-t, majd ennek végrehajtása után törli a régi verziót és a katalógusmutatót az új file-ra állítja.

Programok ellenőrzése
A

VERIFY <név>,<egységszám>

utasítás ellenőrzi, hogy a <név> nevű file tartalma megegyezik-e a C-64 memóriájában tárolt programmal. Ha igen, akkor OK választ kapunk, ha nem, hibajelzést. A DOS először a 0., majd az 1. meghajtóban kezdi el keresni a file-t. Abban az esetben, ha csak az egyik meghajtóban levő lemezen akarjuk keresni, a <név> sztringnek a "0:", illetve "1:" sztringgel kell kezdődnie.

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 LOAD utasítás végrehajtása után a memóriában elvár.

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.

```
100 REM ***************************
110 REM *                         *
120 REM *                         *
130 REM * DISK.APPEND             *
140 REM *                         *
150 REM *  KET BASIC PROGRAMOT     *
160 REM *  TARTALMAZO FILE-T      *
170 REM *  FUZ EGYMASHOZ.         *
180 REM *                         *
190 REM ***************************
200 :
210 REM A FILE-OK NEVEI
220 INPUT "ELSO FILE NEVE=";E$
230 INPUT "MASODIK FILE NEVE=";M$
240 INPUT "EREDO FILE NEVE=";F$
250 :
260 REM *****************************
270 REM * AZ ELSO FILE ATMASOLASA *
280 REM *****************************
290 OPEN 2,8,2,E$+",P,R":OPEN 3,8,3,F$+",P,W"
300 :
310 REM EL=A MUTATOK ELTOLASI ERTEKE
320 GOSUB 430,ME=NE:GOSUB 500:EL=0:MUT=NE
330 PRINT"KEZDOCIM=";NE:GOSUB 520
340 CLOSE 2:OPEN 2,8,2,M$+",P,R"
350 :
360 REM *********************************
370 REM * A MASODIK FILE HOZZAFUZESE *
380 REM *********************************
390 GOSUB 430:EL=MUT-NE:MUT=NE:GOSUB 520
```

```
400 PRINT#3,CHR$(0);CHR$(0);:CLOSE2 : CLOSE 3:END
410 :
411 REM ***************
412 REM * ALPROGRAMOK *
413 REM ***************
420 REM EGY KET BYTE-OS CIM ELOALLITASA
430 GOSUB 460:Y$=X$:GOSUB 460:NE=256*ASC(X$)+ASC(Y$):RETURN
440 :
450 REM EGY KARAKTER BEOLVASASA
460 GET#2,X$:IF X$="" THEN X$=CHR$(0)
470 RETURN
480 :
490 REM EGY CIM KET BYTE-JANAK ELOALLITASA
500 Y$=CHR$(ME/256):X$=CHR$(ME AND 255):PRINT#3,Y$;X$;:RETURN
505 :
506 REM EGY PROGRAMSOR ATMASOLASA MUTATO NELKUL
510 FOR I=1 TO DB:GOSUB460:PRINT#3,X$;:NEXT I:RETURN
514 :
515 REM ******************************
516 REM * A TELJES PROGRAM MASOLASA *
517 REM ******************************
518 REM A MUTATO ELOALLITASA
520 GOSUB 430:IF NE=0 THEN RETURN
523 :
525 REM A PROGRAMSOR HOSSZANAK(DB)
526 REM ES MUTATOJANAK(NE) ELOALLITASA
527 REM ES A MUTATO MODOSITASA
530 DB=NE-MUT-2:MUT=NE:ME=MUT+EL
534 :
535 REM A MUTATO KIIRASA:A PROGRAMSOR KIIRASA
540 GOSUB 500:GOSUB 510:GOTO 520
```

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. . . . . .
                  ↑

                  író/olvasó fej
```

A sorozat valamelyik helyére pozícionálva áll az író/olvasó fej. Az írás és olvasás hatására a fej előre mozog. A file előző pontjára csak úgy tudunk visszatérni, ha lezárjuk, és újra megnyitva elölről 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:

```
OPEN lf,<egység>,<csatorna>,"{<d>:}<név>,SEQ,<mod>"
```

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);
iv/ **le nem zárt file olvasására** (MODIFY)

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" alakot kell használnunk. Ez törli a régi file-t és egy újat hoz létre ugyancsak <név> névvel.

A 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.

A iv/ esetben egy, az íráskor - feltehetően programhiba vagy áramkimaradás miatt - le nem zárt file tartalma nyerhető vissza és pl. átmásolva egy új file-ba, helyreállítható.

Az alábbi program demonstrálja az egyes megnyitási módok hatását:

```
100 REM ************************
105 REM * DSK.SEQV2             *
110 REM ************************
115 REM * SZEKVENCIALIS FILE   *
120 REM * MEGNYITASI MODOK     *
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
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
```

Szekvenciális file-ok lezárása

Szekvenciális file-okat a CLOSE lf utasítás segítségével kell lezárni, ahol lf a szóbanforgó 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 formába kiírja az lf logikai file számú file-ba.

GET#lf,<változólista>

Az utasítás az lf logikai file számú file következő karaktereit beolvassa, és azokat egyenként a változólista elemeihez rendeli. Ismeretlen strukturájú file-okat a legegyszerűbb a GET# utasítás segítségével olvasni.

INPUT#lf,<változólista>

Az utasítás beolvassa az lf logikai file számú 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 elválasztójelekkel (vessző, pontosvessző, kettőspont, kocsivissza-soremelés) kell elválasztani. Ha tehát különböző mennyiségeket INPUT#-kal akarunk visszaolvasni, akkor a kiíráskor nekünk kell az elválasztó jelekről gondoskodnunk. Ha tehát egy file első három elemeként az A$,B,C% változók értékét szeretnénk kiírni úgy, hogy az INPUT#-al 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):

I	L	D	I	K	O	,		2	5	0	0	,	3	CHR(13)	EOF

Ugyanez a ii/ esetben:

I	L	D	I	K	O	CHR(13)		2	5	0	0	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. (Az ST változó részletes leírása a 4. fejezetben található.) 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.SEQLIST        *
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,"0:"+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
```

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:

OPEN lf,<egységszám>,<csatorna>,"{<d>:}<név>,L,"+CHR$(<hossz>)

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. <egységszám> 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 "@:<név>" alakot, a file - ha esetleg létezett - nem fog törlődni. A relatív file-ok csak a SCRATCH DOS paranccsal törölhetők. Már meglevő relatív file-t az

OPEN lf,<egységszám>,<csatorna>,"{<d>:}<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 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:

PRINT#hf,"P"+CHR$(<csatorna>)+CHR$(L)+CHR$(H)+CHR$(P)

hf a 15. csatorna megnyitásakor használt logikai file szám,

<csatorna> a relatív file megnyitásában használt csatornaszám (secondary adress). L és H a rekord sorszámának alsó és felső byte-ja. A valódi sorszám tehát 256*H+L. Végül P a rekordon belüli kezdőpozíció.

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-ból való olvasásra, illetve az oda való írásra. Az I/O művelet a legutolsó P DOS-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 túl ha olvasunk vagy írunk, hibajelzést kapunk (ST=64). Mivel azonban a rekord vége általában logikai határ is, célszerű elkerülni az olyan I/O műveletet, amelyik átlépi a rekord határát.

Példánkban egy számokat tartalmazó file-t nyitunk meg, és 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 ****************************
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          *
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
```

Végezetül egy példát adunk, amelyik a relatív file-okat egy személyzeti nyilvántartás megvalósítására használja. A program a valódi helyzetet természetesen a végletekig leegyszerűsíti, de jól mutatja, hogyan lehet egyszerűbb adatbázisok létrehozására használni.

A program minden egyes személyről a következő adatokat tartja nyilván:

a/ név;
b/ születési hely;
c/ születés éve;
d/ fizetés.

Mód van új emberek adatainak felvitelére, a meglevő adatok módosítására, egyes személyek adatainak törlésére. A törlés azt jelenti, hogy az utolsó rekordot átmásoljuk a törlendő rekord helyére, és az N számot, ami az adatbázisban szereplő személyek számát tárolja, eggyel csökkentjük. Ez nem a legszerencsésebb megoldás, hiszen ha eddig ABC rendben voltak az adatbázisban, akkor ez azonnal elromlik. Lehetőség van továbbá az egyes rekordok olvasására, illetve az adatbázis teljes tartalmának nyomtatón történő kilistázására.

Az első program létrehozza a relatív file-t, és az első rekordba beírja, hogy még egyetlen személy adatát sem tartalmazza. Magának a nyilvántartó programnak a használata előtt ezt a programot kell lefuttatni.

```
100 REM ****************
110 REM * SZEMELYI.GEN *
120 REM ****************
130 :
140 REM ADATBAZIS GENERALASA
150 REM ====================
160 OPEN 15,8,15
170 PRINT#15,"S:ADATOK"
180 OPEN 2,8,2,"ADATOK,L,"+CHR$(150)
190 PRINT#15,"P"+CHR$(2)+CHR$(1)+CHR$(0)+CHR$(1)
200 :
210 N=0: PRINT#2,N
220 CLOSE 2: CLOSE15
```

```
1000 REM **********************
1010 REM *SZEMELYI ADATBAZIS*
1020 REM **********************
1030 :
1040 REM N$ NEVEK
1050 REM S$ SZULETESI HELYEK
1060 REM SI SZULETESI IDOK
1070 REM M  MEGBIZHATOSAGI INDEX (1 HA IGEN; 0 HA NEM)
1080 :
1090 V$="":NMAX=254: GOSUB 1250
1100 :
1110 REM MENU KIIRASA
1120 REM ============
1130 PRINT "■■■■■■■■■■■■SZEMELYI NYILVANTARTAS"
1140 PRINT "■■■"
1150 PRINT "      UJ EMBER FOLVETELE.....1"
1160 PRINT "      LISTAZAS..............2"
1170 PRINT "      MODOSITAS.............3"
1180 :
1190 PRINT "      TORLES................4"
1200 PRINT "      VEGE..................5"
1210 INPUT "■■■■■■■■MELYIKET VALASZTJA";V
1220 ON V GOSUB 1340,1450,1630,2040,1770
1230 GOTO 1130
1240 :
1250 REM ADATBAZIS MEGNYITASA
1260 REM ====================
1270 :
1280 OPEN 15,8,15
1290 OPEN 2,8,2,"ADATOK"
1300 PRINT#15,"P"+CHR$(2)+CHR$(1)+CHR$(0)+CHR$(1)
1310 INPUT#2,N
1320 RETURN
1330 :
1340 REM UJ EMBER FOLVETELE
1350 REM ==================
1360 IF N = NMAX THEN RETURN
1370 PRINT "■■■■■■■■SZEMELYI ADATOK"
1380 PRINT "■■■"
1390 I%=N+1:GOSUB 1930
1400 PRINT "■■■ KIVANJA-E MODOSITANI(I/N)
1410 GET A$:IF A$="" THEN GOTO 1410
1420 IF A$<>"I" THEN GOTO 1440
1430 PRINT "■■■■■■■■": GOTO 1390
1440 GOSUB 2380: N=N+1:RETURN
1450 :
1460 REM LISTAZAS
1470 REM ========
1480 IF N=0 THEN RETURN
1490 INPUT"■■■■MELYIK REKORDDAL KEZDJEM";I%
1500 IF I%<1 THEN I%=1
1510 IF I%>N THEN I%=N
```

```
1520 GOSUB 2300: GOSUB 1830
1530 PRINT"■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■";
1540 PRINT" ■+■ ELORE     ■-■ HATRA "
1550 PRINT" ■H■ MASOLAT  ■RETURN■ VISSZA";
1560 GET A$:IF A$="" THEN 1560
1570 IF A$="+" THEN I%=I%+1:GOTO 1500
1580 IF A$="-" THEN I%=I%-1: GOTO 1500
1590 IF A$="H" THEN GOTO 2180
1600 IF A$<>CHR$(13) THEN GOTO 1560
1610 RETURN
1620 :
1630 REM ADATOK MODOSITASA
1640 REM ==================
1650 PRINT "■■■■ HANYADIK REKORDOT KIVANJA MODOSITANI:":INPUT " ";I%
1660 IF (I%<1) OR (I%>N) THEN RETURN
1670 GOSUB 2300: GOSUB 1830
1680 PRINT "■■■ KIVANJA-E MODOSITANI(I/N)"
1690 GET A$:IF A$="" THEN 1690
1700 IF A$<>"I" THEN GOSUB 2380: RETURN
1710 PRINT "■■■■■■■■"
1720 GOSUB 1930
1730 GOTO 1680
1740 :
1750 REM ADATBAZIS LEZARASA
1760 REM ===================
1770 PRINT#15,"P"+CHR$(2)+CHR$(1)+CHR$(0)+CHR$(1)
1780 INPUT#2,N
1790 CLOSE2: CLOSE15: END
1800 :
1810 REM KIIRO SZUBRUTIN
1820 REM ================
1830 PRINT "■■■■■■■REKORDSZAM=";I%
1840 PRINT "■■■"
1850 PRINT " NEV.......... ";N$
1860 PRINT " SZUL.HELY.... ";S$
1870 PRINT " SZUL.IDO.....";SI
1880 A$="IGEN"
1890 IF M=0 THEN A$="NEM"
1900 PRINT " MEGBIZHATO-E. ";A$
1910 RETURN
1920 :
1930 REM BEOLVASO SZUBRUTIN
1940 REM ===================
1950 INPUT " NEV.........";N$
1960 INPUT " SZUL.HELY...";S$
1970 INPUT " SZUL.IDO....";SI
1980 INPUT " MEGBIZHATO-E";X$
1990 M=1: IF X$="NEM" THEN M=0
2000 RETURN
2010 :
2020 REM TORLO RUTIN
2030 REM ============
2040 PRINT"■■■■■HANYADIK REKORDOT KIVANJA TOROLNI:"
```

```
2050 INPUT " ";J%:
2060 IF (J%<1) OR (J%>N) THEN RETURN
2070 I%=J%: GOSUB 2300: GOSUB 1830
2080 PRINT"[RVS] KIVANJA-E TOROLNI?"
2090 GET A$ : IF A$="" THEN GOTO 2090
2100 IF A$<>"I" THEN RETURN
2110 I%=N: GOSUB 2300
2120 I%=J%: GOSUB 2300
2130 N=N-1
2140 RETURN
2150 :
2160 REM HARDCOPY
2170 REM ========
2180 OPEN 4,4
2190 PRINT#4,"REKORDSZAM=";I%
2200 PRINT#4: PRINT#4
2210 PRINT#4," NEV.......... ";N$
2220 PRINT#4," SZUL.HELY.... ";S$
2230 PRINT#4," SZUL.IDO.....";SI
2240 A$="IGEN"
2250 IF M=0 THEN A$="NEM"
2260 PRINT#4," MEGBIZHATO-E. ";A$
2270 PRINT#4:CLOSE4
2280 GOTO 1560
2290 :
2300 REM EGY REKORD BEOLVASASA
2310 REM ======================
2320 PRINT#15,"P"+CHR$(2)+CHR$(I%+1)+CHR$(0)+CHR$(1)
2330 INPUT#2,N$
2340 PRINT#15,"P"+CHR$(2)+CHR$(I%+1)+CHR$(0)+CHR$(40)
2350 INPUT#2,SI,M,S$
2360 RETURN
2370 :
2380 REM EGY REKORD KIIRASA
2390 REM ===================
2400 PRINT#15,"P"+CHR$(2)+CHR$(I%+1)+CHR$(0)+CHR$(1)
2410 PRINT#2,N$
2420 PRINT#15,"P"+CHR$(2)+CHR$(I%+1)+CHR$(0)+CHR$(40)
2430 PRINT#2,SI,V$,M,V$,S$
2440 RETURN
```

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 hf a 15. csatornának megfelelő logikai file-szám (ami általában 15). A <csatorna> a # pszeudo-file megnyitásában szereplő csatornaszám. A <meghajtó> a

meghajtó száma, általában 0. (Olyan lemezegység esetén, amelyik két lemezmeghajtót is tartalmaz, a <meghajtó> értéke 0 vagy 1 lehet.) A <sáv> és a <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:" vagy az "U1:" szöveggel helyettesíthető. (Lásd a USER utasítást!)

Következő példánk a lemez megadott blokkját olvassa be a C-64 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:"+<sztring>
    vagy
PRINT#hf,"BLOCK-WRITE:";<csatorna>;<meghajtó>;<sáv>;<szektor>
```

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> a

meghajtó száma, általában 0. (Olyan lemezegység esetében, amelyik két lemezmeghajtót is tartalmaz, a <meghajtó> értéke 0 vagy 1 lehet.) A <sáv> és a <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-W:" vagy az "U2:" szöveggel is helyettesithető. (Lásd a USER utasítást.)

Első példánk az 1. sáv 0.-20. sorszámú szektoraiba 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, M

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 ellentetje. A B-F parancsban specifikált blokknak a BAM-ban megfelelő bit 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
```

a 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 (0-255-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; a <mutató> a # pszeudo-file pufferének pointerét állítja át. A "B-P:" rövidítés itt is megengedett. A B-P parancsot elsősorban a B-W és a B-R utasításokkal együtt használjuk. (A 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 szóbanforgó 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

A VIC 1541-es egység esetén két további 'user' utasítás van:

UI+	C-64 sebességének kiválasztása
UI-	VIC-20 sebességének kiválasztása

Külön szólunk az U1 és az 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.

5.7 Gépi kódú programok

A bemutatott, lemezegységet használó gépi kódú programok, programrészek elsősorban a KERNAL rutinokat használják, így megértésükhöz a 6. és a 9. fejezet mélyebb ismeretére van szükség. A gépi kódú programrészek megírásához a KERNAL rutinokon kívül nagy segítséget nyújtanak a következő, 0. lapon levő címek:

Cím	Leírás
183($00B7)	Aktuális file-név (parancs) hossza
184($00B8)	Aktuális logikai file szám
185($00B9)	Aktuális csatornaszám
186($00BA)	Aktuális egységszám
187-188($00BB-00BC)	Az aktuális file-név (parancs) elejére mutat.

Példák

i/ Programunk egy már BASIC vagy gépi kódú rutinnal megnyitott szekvenciális file adatait olvassa és írja ki a képernyő tetejére. A képernyő- és ASCII kódok eltérnek, ezért a 'rekord vége' jel (CHR$(13)), mint egy M betű fog megjelenni. A programot az

```
OPEN 2,8,2,"FILE,SEQ,READ":SYS 828
```

paranccsal hajtathatjuk végre.

```
033C                    10          *=$033C
033C A2 02              20          LDX #$02
033E 20 C6 FF           30          JSR $FFC6
0341 A0 00              40          LDY #$00
0343 20 CF FF           50 L1       JSR $FFCF
0346 99 00 04           60          STA $0400,Y
0349 A9 0E              70          LDA #$0E
034B 99 00 D8           80          STA $D800,Y
034E C8                 90          INY
034F F0 04             100          BEQ OUT
0351 A5 90             110          LDA $90
0353 F0 EE             120          BEQ L1
0355 4C CC FF          130 OUT      JMP $FFCC
```

ii/ **Egy parancs sztring elküldése**
A legtöbb esetben a parancs az input pufferben található, ez azonban nem törvényszerű. A parancs elküldéséhez a következőket kell végrehajtanunk:

(i) $08 betöltése $BA-ba (egységszám=8)
(ii) $6F betöltése $B9-be (csatornaszám=15+TALK)
(iii) 'LISTEN' üzenet elküldése
(iv) Az IEEE parancs ($6F) elküldése
(v) A parancs sztring byte-onként való elküldése
(vi) 'UNLISTEN' üzenet elküldése.

A gépi kódú rutint a SYS 828,"<parancs> típusú aktivizálásra terveztük, első használata előtt az OPEN 15,8,15 utasítást nem kell kiadni. A program elején a használt KERNAL rutinok kezdőcímeit felsoroltuk.

```
1000                    10          ; PARANCS ELKULDESE
1000                    20          ;
1000                    30          ; HIVASA:
1000                    40          ;          SYS 828,"<PARANCS>
1000                    50          ;
FFA8                    60 CIOUT    = $FFA8
FF93                    70 SECOND   = $FF93
0073                    80 CHRGET   = $73
FFB1                    90 LISTEN   = $FFB1
00BA                   100 FA       = $BA
00B9                   110 SA       = $B9
FFAE                   120 UNLSN    = $FFAE
033C                   130          *= $033C
033C 20 73 00          140          JSR CHRGET   ; A (,) ATLEPESE
033F A9 08             150          LDA #$08
0341 85 BA             160          STA FA       ; EGYSEGSZAM = 8
0343 A9 6F             170          LDA #$6F
0345 85 B9             180          STA SA       ; CSATORNASZAM = 15
0347 A5 BA             190          LDA FA
0349 20 B1 FF          200          JSR LISTEN
034C A5 B9             210          LDA SA
034E 20 93 FF          220          JSR SECOND
0351 20 73 00          230 LOOP     JSR CHRGET
0354 C9 00             240          CMP #$00
0356 F0 06             250          BEQ OUT
0358 20 A8 FF          260          JSR CIOUT
035B 4C 51 03          270          JMP LOOP
035E 20 AE FF          280 OUT      JSR UNLSN
0361 60                290          RTS
```

Például:
SYS 828,"I inicializálja a lemezegységet,
SYS 828, "S:PROG* törli az összes PROG-gal kezdődő file-t stb.

iii/ Utolsó példánk a hiba csatornát olvassa és tartalmát kiírja a képernyő tetejére. A program ugyancsak a SYS 828 paranccsal hajtatható végre.

```
033C                  10          *=$033C
033C A9 08            20          LDA #$08     ; EGYSEGSZAM = 8
033E 85 BA            30          STA $BA
0340 20 B4 FF         40          JSR $FFB4
0343 A9 6F            50          LDA #$6F     ; CSATORNA = 15
0345 20 96 FF         60          JSR $FF96    ; HIBA CSATORNA
0348                  65          ;              KOVETKEZO BYTE
0348 20 A5 FF         70 LOOP     JSR $FFA5
034B C9 0D            80          CMP #$0D
034D F0 05            90          BEQ OUT
034F 20 D2 FF        100          JSR $FFD2
0352 D0 F4           110          BNE LOOP
0354 20 D2 FF        120 OUT      JSR $FFD2
0357 20 AB FF        130          JSR $FFAB
035A 60              140          RTS
```

5.8 BASIC 4.0 utasítások

A Commodore nagyobb gépein a BASIC 3.0-ás, illetve 4.0-ás változatai futnak. A C-64-hez is vásárolhatók olyan bővítések, melyek lehetővé teszik ezeknek az utasításoknak a használatát.

Elsősorban az IEEE 488-as illesztők szoktak magán az illesztőn túl olyan bővítéseket tartalmazni, amelyek - többek közt - ezeket az utasításokat is implementálják.

Ebben a paragrafusban a Commodore BASIC V4.0 speciális, a lemezegységek kezelésére szolgáló utasításait ismertetjük. Az előbb már említett interface-ek további, de egymástól meglehetősen eltérő utasításokat is tartalmaznak, ezekre külön nem térünk ki.

Az utasítások felsorolása előtt egy általános megjegyzés. A BASIC 4.0-ás utasítások, a meghajtó megnevezésének hiányában minden parancsot a 0-ás meghajtóra vonatkoztatnak. Ezzel ellentétben, a BASIC 2.0-ás változata ilyenkor mindig az utoljára kijelölt meghajtót használja. A BASIC 2.0 LOAD parancs a programot először mindig a kijelölt meghajtóban keresi, s ha ott nem találja, akkor a másik meghajtóban levő lemezt is végigolvassa. Egy meghajtós lemezegységek esetén ilyen probléma természetesen nem merül fel.

Az alábbi leírásban a "<név>" sztringek mindenütt helyettesíthetők egy **tetszőleges sztring kifejezéssel** is. Ebben az esetben azonban ezt mindig **kerek zárójelek közé** kell tenni:

```
DLOAD ("TEXT"+D$),D1
```

A file-ok írásra való megnyitásánál, illetve programok elmentésénél a @ szerepe ugyanaz, mint a BASIC 2.0-ban, azzal az egy különbséggel, hogy a @ jel és a név közé nem kell kettőspontot tenni. Például

```
DSAVE "@PROGRAM"
```

minden esetben felülírja - az esetleg létező - "PROGRAM" nevű programot.

APPEND

Rövidítés: aP# *Token:* $D4 (212)

Szintaxis: APPEND# <lf>, "<név>" {,D<d>} {ON U<1>}

Az utasítás az <1> lemezegység <d> meghajtójában levő <név> nevű szekvenciális file-t nyitja meg továbbírásra. Ekvivalens az OPEN <lf>,<1>,<csat.>,"<d>:<név>,S,A" utasítás kiadásával.

BACKUP

Rövidítés: bA *Token:* $D2 (210)

Szintaxis: BACKUP D<d1> TO D<d2> {ON U<1>}

Az utasítás az <1> lemezegység <d1> meghajtójában levő lemezt fizikailag átmásolja a <d2> meghajtóban levő lemezre. Csak duál lemezegység esetén van hatása. A <d2> meghajtóban levő lemez tartalma teljes egészében elvész. Ekvivalens a

PRINT#15,"D<d2>=<d1>"

utasítással. (Feltéve, hogy a 15-ös logikai file számhoz az <1> lemezegység 15. csatornája tartozik.)

CATALOG
DIRECTORY

Rövidítés: cA *Token:* $D7 (215)
dI *Token:* $DA (218)

Szintaxis:

{ CATALOG } {D<d>} {ON U<1>}
{ DIRECTORY }

Az utasítás az <1> lemezegység <d> meghajtójában levő lemez katalógusát a képernyőre listázza. A memóriában levő program sértetlenül megmarad. BASIC 2.0 ekvivalense nincs.

COLLECT

Rövidítés: coL *Token:* $D1(209)

Szintaxis: COLLECT {D<d>} {ON U<1>}

Az utasítás az <1> hardver számú lemez <d>-ik meghajtóban levő lemezt szervezi újjá. Hatása ekvivalens a

PRINT#15,"<d>:VALIDATE"

utasítás kiadásával. (Feltéve, hogy a 15-ös logikai file számhoz az <1> lemezegység 15. csatornája tartozik.)

CONCAT

Rövidítés: conC *Token:* $CC(204)

Szintaxis:
CONCAT {D<d1>} {,"<név1>"} TO {D<d2>} {,"<név2>"} {ON U<1>}

Az utasítás az <1> lemezegység <d2> meghajtójában levő <név2> nevű file-hoz hozzáfűzi a <d1> meghajtóban levő <név1> nevű file-t. Az utasítást csak SEQ vagy PRG típusú file-okra lehet használni. Az utasítás hatása ekvivalens a

PRINT#15,"C<d2>:<név2>=<d2>:<név2>,<d1>:<név1>",<1>

utasítás kiadásával. (Feltéve, hogy a 15-ös logikai file számhoz az <1> lemezegység 15. csatornája tartozik.)

COPY

Rövidítés: coP *Token:* $D3 (211)

Szintaxis:
COPY {D<d1>} "<név1>" TO {D<d2>} "<név2>" {ON U<1>}

Az utasítás az <1> lemezegység <d1> meghajtójában levő <név1> nevű SEQ vagy PRG típusú file-t átmásolja a <d2> meghajtóban levő lemezre <név2> név alatt. Ha <d1>=<d2>, akkor ugyanaz a file kétszer fog a lemezen szerepelni, csak más név alatt. Ha **mindkét név hiányzik**, akkor az utasítás a <d1> meghajtóban levő **összes** file-t átmásolja a <d2> meghajtóban levő lemezre. A

<d2>-n levő file-ok megmaradnak.

Az utasítás ekvivalens a

PRINT#15,"C<d2>:<név2>=<d1>:<név1>"

utasítás kiadásával. (Feltéve, hogy a 15-ös logikai file számhoz az <1> lemezegység 15. csatornája tartozik.)

DCLOSE

Rövidítés: dc1O *Token:* $CE (206)

Szintaxis:
DCLOSE {#<lf>} {ON U<1>}

Az utasítás az <1> egységen lezárja a DOPEN-nel megnyitott <lf> logikai file számú file-t. Ha a <lf>-et nem adjuk meg, akkor valamennyit lezárja. BASIC 2.0 ekvivalens nincs.

DLOAD

Rövidítés: dL *Token:* $D6 (214)

Szintaxis: DLOAD "<név>"{,D<d>} {ON U<1>}

Az utasítás betölti az <1> lemezegység <d> meghajtójában levő <név> nevű programot a BASIC munkaterületre. Ekvivalens a

LOAD "<d>:<név>",<1>

utasítással.

DOPEN

Rövidítés: doP *Token:* $CD (205)

Szintaxis:
DOPEN #<log. file szám>, "<név>" {,D<d>} {ON U<1>} { ,L<rh> / ,W / {,R} }

Az utasítás REL vagy SEQ típusú file-okat nyit meg az <1> hardver számú lemezegység <d> meghajtójában levő lemezen. Az L

és W paraméter egyszerre nem használható. Az L<rh> paramétert relatív file-ok kreálásánál kell használni, ekkor L a rekordhossz. A W paramétert SEQ típusú file írásra való megnyitásakor kell megadni. Az R paraméter használata csak szekvenciális file-ok esetén megengedett. Ekkor olvasásra nyitja meg.

Példák:

```
(i)    DOPEN# 2,"ADATOK",D1,W
(ii)   DOPEN# 7, "PROBA",L120
(iii)  DOPEN# 2, "PROBA" ON U9
```

Az (i) példa a 8-as hardver számú lemezegység első meghajtójában levő lemezen nyit meg írásra egy SEQ típusú file-t. Ha ilyen file már létezett, akkor hibajelzést kapunk. Ha a létező file felülírását akarjuk, akkor a "@ADATOK" file nevet kell használnunk.

A (ii) példában egy "PROBA" nevű relatív (REL) file-t nyitunk meg a 8-as lemezegység 0-ás meghajtójában levő lemezen. A rekordhossz 120 lesz.

A (iii) példa a "PROBA" nevű file-t nyitja meg a 9-es lemezegység 0-ás meghajtójában levő lemezen. Ha ez a file szekvenciális (SEQ) volt, akkor írásra nyitja meg a rendszer. Ha relatív file volt, akkor írásra és olvasásra egyaránt megnyílik.

DS és DS$

Fenntartott BASIC változók

A BASIC 4.0-nak további két fenntartott változója is van. Ezek a **DS**, illetve a **DS$** változók. DS a lemezegység hibakódjával tér vissza, DS$ pedig a teljes hibaüzenettel, beleértve a hibakódot, a sáv- és szektorszámot is. A DS-re vagy a DS$-ra történő hivatkozás felismerésekor az interpreter elolvassa a 15. csatornát, majd ennek megfelelően állítja be DS, illetve DS$ értékét. Az ST változó természetesen továbbra is ugyanúgy használható.

DSAVE

Rövidítés: dsA *Token:* $D5 (213)

Szintaxis: DSAVE "<név>" {,D<d>} {ON U<1>}

Az utasítás az <l> egységszámú lemezegység <d> meghajtójában levő lemezre másolja ki a memóriában levő BASIC programot. Az utasítás ekvivalens a

SAVE "<d>:<név>",<l>

utasítással.

HEADER

Rövidítés: heA *Token:* $D0 (208)

Szintaxis: HEADER "<név>" {,D<d>} {,I<id>} {ON U<l>}

Az utasítás megformázza az <l> lemezegység <d> meghajtójában levő lemezt. <név> a lemez neve lesz, <id> két karakteres azonosító. Ha az I paramétert nem adjuk meg, akkor csak törli az utasítás a lemezt, de nem formázza újra. Az utasítás hatása ekvivalens a

PRINT#15,"N<d>:<név>,<id>"

kiadásával. A 15-ös logikai file-t az OPEN 15,<l>,15 utasítással kell megnyitni.

RECORD#

Rövidítés: reC# *Token:* $CF (207)

Szintaxis: RECORD# <lf>, <rek.szám> {,<pozíció>}

Az utasítás a <log. file szám>-ú relatív file író/olvasó fejét a <rek.szam>-ú rekord <pozíció> helyére állítja be.Ha a <pozíció>-t nem adjuk meg, akkor az 0-nak felel meg. Az utasítás ekvivalens a megfelelő P DOS parancs kiadásával. Annak szintaxisa azonban lényegesen bonyolultabb.

RENAME

Rövidítés: reN *Token:* $DB (216)

Szintaxis: RENAME "<név2>" TO "<név1>" {,D<d>} {ON U<l>}

Az utasítás az <l> lemezegység <d> meghajtójában levő <név2> nevű file-t átnevezi <név1>-re. Hatása ekvivalens a

PRINT#15,"R<d>:<név1>=<név2>"

utasítás kiadásával. (Feltéve, hogy a 15-ös logikai file számhoz az <l> lemezegység 15. csatornája tartozik.)

SCRATCH

Rövidítés: scR *Token:* $D9 (217)

Szintaxis: SCRATCH "<név>" {,D<d>} {ON U<l>}

Az utasítás hatására az <l> lemezegység <d> meghajtójában levő lemezről a DOS törli a <név> névnek megfelelő file-okat. A * és ? karakterek hatása a szokásos. Az utasítás ekvivalens a

PRINT#15,"S<d>:<név>"

utasítás kiadásával. (Feltéve, hogy a 15-ös logikai file számhoz az <l> lemezegység 15. csatornája tartozik.)

Végül az 5.5 paragrafusban már szerepelt, a relatív file-ok használatát bemutató program BASIC 4.0-ás változatát mutatjuk be. A program lényegesen egyszerűbb és áttekinthetőbb lett:

```
100 REM ****************************
110 REM *                          *
120 REM *                          *
130 REM * PELDA RELATIVE FILE-RA   *
140 REM *                          *
150 REM * A PROGRAM 30 SZAMOT IR   *
160 REM * EGY RELATIV FILE-BA.     *
170 REM * A SZAMOK AZUTAN TETSZES  *
180 REM * LEKERDEZHETOK ES HA      *
190 REM * KELL, MODOSITHATOK.      *
200 REM *                          *
210 REM ****************************
213 REM *                          *
214 REM * BASIC 4.0-AS VALTOZAT    *
215 REM *                          *
218 REM ****************************
220 :
230 REM A RELATIVE FILE LETREHOZASA
240 SCRATCH "PROBA"
250 DOPEN#2, "PROBA",L14
260 RECORD#2,30: PRINT#2,CHR$(255);
```

```
270 :
280 REM A 30 SZAM KIIRASA
290 REM ==================
300 REM A SZAMOK MEGEGYEZNEK A REKORD SORSZAMAVAL
310 REM HA MAGUNK AKARJUK MEGADNI OKET
320 REM A SORBAN ALLO MEGJEGYZESBEN
330 REM LEVO UTASITAST KELL HASZNALNI
340 :
350 FOR I=1 TO 30
360 SZAM=I: REM INPUT "KOV.SZAM";SZAM
370 REM A REKORDSZAM BEALLITASA
380 RECORD#2,I
390 :
400 REM A HIBA ELLENORZESE
410 IF (DS<20) OR (DS=50) THEN GOTO 450
420 PRINT DS$: DCLOSE#2: END
430 :
440 REM A SZAM KIIRASA
450 PRINT#1,SZAM;
460 IF (DS<20) OR (DS=50) THEN GOTO 500
470 GOTO 420
480 :
490 REM A CSATORNAK LEZARASA
500 NEXT: DCLOSE#2
510 :
520 REM MODOSITO RESZ. HA A PROBA NEVU
530 REM FILE A 30 SZAMMAL MAR LETEZIK
540 REM INNEN KELL A PROGRAMOT INDITANI
550 DOPEN#1,"PROBA"
560 :
570 REM MODOSITO RESZ
580 REM ==============
590 INPUT"MELYIK SZAM KELL";I
600 I=INT(I):IF (I<1) OR (I>30) THEN GOTO 590
610 RECORD#1,I
620 REM A SZAM BEOLVASAS
630 INPUT#1,SZAM
640 PRINT SZAM
650 PRINT "AKARJA-E MODOSITANI(I/N)"
660 GET A$: IFA$="" THEN GOTO 660
670 IF A$="N" THEN GOTO 710
680 INPUT "UJ SZAM ERTEKE";SZAM
690 RECORD#1,I
700 PRINT#1,SZAM
710 PRINT"ELEG VOLT-E (I/N)"
720 GET A$: IF A$="" THEN GOTO 720
730 IF A$="N" THEN GOTO 590
740 :
750 REM A CSATORNAK LEZARASA
760 DCLOSE#1: END
```

Készült: VASKUT MÜFIL Nyomda – KARTON GM.
87.–173. I. kötet
Kiadó: LSI ATSz
Felelős Kiadó: dr. Kovács Magda

I-II. kötet ára: 370,- Ft

NOVOTRADE - 2C ÁRUHÁZ

1136 Bp., Balzac u. 35. Tel.: 402-954

ÁLLAMI KÖNYVTERJESZTŐ VÁLLALAT - NOVOTRADE 2C

Táncsics Könyvesbolt
1073. Bp. Lenin krt. 17.
Tel.: 422-178

Műszaki Könyváruház
1061. Bp. Liszt F. tér 9.
Tel.: 420-353

Tankönyvcentrum
1050. Bp. Október 6.u. 8-9.

AKADÉMIAI KÖNYVESBOLT
MAGISZTER KÖNYVESBOLTJA
1052. Bp. Városház u. 1.
Tel.: 382-402 , 382-440

NOVOTRADE