Hornig Trapp Weltner

További Tippek és trükkök a Commodore 64-eshez

A C 64-es felhasználóinak kincsesbányája...

DATA BECKER – NOVOTRADE

Hornig Trapp Weltner

További Tippek és trükkök a Commodore 64-eshez

A C 64-es felhasználóinak kincsesbányája...

DATA BECKER – NOVOTRADE

A mű eredeti címe: 64 noch mehr Tips & Tricks. Band 2. (1984)

Fordította: APEX Szervező, Szolgáltató GMK munkacsoportja
Szaklektorok: DR. LENGYEL JÓZSEF és DR. SZIDAROVSZKY FERENC

A kiadásért felel: RÉNYI GÁBOR, a NOVOTRADE RT igazgatója
Budapest, 1986

Kiadványmenedzser: BÉKÉS TAMÁS
Felelős szerkesztő: SÍK JÚLIA
Műszaki szerkesztő: HAJDÚ ÁRPÁD

Szedte a Nyomdaipari Fényszedő Üzem (867677/9)
Készült a Somogy Megyei Nyomdában – 86-5222

ISBN 963 02 4346 6

FONTOS TUDNIVALÓ

A könyvben ismertetett kapcsolások, eljárások és programok nem tekinthetők szabadalmi oltalom alá eső ipari termékeknek. Ezek elsősorban amatőr és oktatási célokat szolgálnak. A szerzők rendkívül nagy gondot fordítottak a kapcsolások, műszaki adatok és programok helyességére, a részletek kidolgozása során többszöri ellenőrzést végeztek. Mindez azonban nem zárja ki az esetleges hibalehetőségeket.
Az előforduló hibákért és az ebből adódó következményekért a DATA BECKER cég sem szavatosságot, sem jogi felelősséget nem vállal. Az esetlegesen előforduló hibák közlését a szerzők hálásan fogadják.

TARTALOMJEGYZÉK

1. FEJEZET
BEVEZETÉS

Valószínűleg minden felhasználó tudja, hogy a C 64-es egy kitűnő számítógép. Ez azonban nem jelenti azt, hogy a számítógép építői mindenre gondoltak, és nem követhettek el kisebb hibákat. Néhány hibát szeretnénk most bemutatni. Ezek láttán sokan megijednek, és azt hiszik, hogy hibás a számítógépük. Pedig ezekkel a jelenségekkel a szaküzletben is találkozhatunk.
Most pedig lássuk a hibákat!

1. hiba
Ezt a hibát már mindnyájan tapasztalhattuk. Ha egyidejűleg nyomjuk le a bal SHIFT és a két kurzorvezérlő billentyűt, akkor a képernyőn egy pikk jel jelenik meg.

2. hiba
Nyomjuk le egyidejűleg a COMMODORE-, a pontosvessző- és az egyenlőségjel billentyűket. A kurzor eltűnik.
Mi történt?
Nyomjuk le a RUN-STOP/RESTORE-t és billentyűzzük be a

```
POKE 53281,0
```

parancsot.
Ha most ismét egyidejűleg lenyomjuk az előbbi három billentyűt, láthatjuk, hogy miért tűnt el a kurzor. A betűszín fehérről kékre változott.

3. hiba
Most pedig következzék a legérdekesebb hiba,
Menjünk a kurzorral a legalsó sorba, és addig írjuk a jeleket a képernyőre (szóközt is lehet), amíg a kurzor másodszor is sort nem vált. Nyomjuk le most az INST-DEL billentyűt. Ha most megpróbáljuk a 80. jelet letörölni, a következő hibával találkozhatunk.
A képernyőn az alábbi kép jelenik meg:

```
LOAD

? SYNTAX ERROR
READY
RUN

READY.
```

Ilyenkor nincs mit tenni, a számítógép vezérelhetetlenné vált. De ez még nem minden: elindítunk egy BASIC programot, ami addig fut, míg egy INPUT utasításhoz nem ér. Ekkor előfordulhat, hogy a billentyűzet nem működtethető. Egy nem létező program hatására ismét megjelenik az előbbi felirat. De ne gondoljuk, hogy az ily módon vezérelhetetlenné vált számítógépet csak ki- és bekapcsolással lehet ismét üzemképessé tenni. Nyomjuk le a # (SCHIFT + 3) jelet. Erre a számítógép PRESS PLAY ON TAPE felirattal jelentkezik. A kazettás egységgel rendelkezők jól ismerik ezt az üzenetet. A RUN-STOP billentyű lenyomása után megjelenik a kurzor, s gépünk ismét üzemképes.
Furcsa módon ezek a hibák csak a következő betűszínek esetén jelentkeznek: vörös, világoszöld, sötétkék, sárga (a hibák variációival), rózsaszín, sötétszürke, világoskék és világosszürke.
Ezek a színek a 3, 4, 7 és 8 billentyűkkel megjeleníthetők. Megjegyzés: akkor sem kell nyugtalankodni, ha nem találkozunk ezekkel a hibákkal. Ez nem azt jelenti, hogy rossz a gépünk, csak azt, hogy egy újabb sorozatból származik, s ezeket a hibákat már kiküszöbölték.
Arról, hogy a mi gépünk melyik sorozathoz tartozik, egy más, egyszerűbb módon is meggyőződhetünk. Töröljük a képernyőt, és írjuk be a POKE 1024,1 utasítást. Ha a bal felső sarokban megjelenik egy A betű, akkor biztosak lehetünk abban, hogy gépünk egy újabb modell. A régebbi modelleknél ui. még a szín-RAM-ba is be kell írni POKE utasítással a megfelelő értéket.

A hibák keletkezésének okát nem ismerjük. Az első kettő valószínűleg a billentyűzet-lekérdezéssel van kapcsolatban, a harmadiknál erős a gyanúnk, hogy az IRQ rutin működik helytelenül (a TI$ továbbugrik, a kurzor villog stb.).

Az eddigieket tekintsük egy rövid kitérőnek. A későbbiekben majd láthatjuk, hogy ezen apró hibák ellenére a C 64-es mi mindenre alkalmas.

2. FEJEZET
TIPPEK ÉS TRÜKKÖK HÁZI HASZNÁLATRA

A következő oldalakon számos hasznos BASIC és gépi kódú programmal találkozhatunk.
Az érdekesebb tárolócímeket és a hozzájuk tartozó lehetőségeket ismerhetjük meg...
Röviden: tippeket és trükköket házi használatra!

2.1 A kazettás egység vezérlése BASIC-ből

A kazettás egység motorjának egyik jó tulajdonsága, hogy a LOAD vagy SAVE utasítás végrehajtása után önmagától leáll. A kazettás egységek újraindítani csak a STOP, majd a PLAY billentyűk lenyomásával lehet.
Tehát a kazettás egység motorját a számítógép vezérelni tudja. Ezt a tulajdonságot természetesen felhasználhatjuk saját céljainkra is!

A kazettás egység motorjának programmal való vezérlésében nagy szerepe van az 1-es címen levő processzor portnak.
Ezenkívül szükségünk van a motorkapcsolóra, ami a 192-es ($C0) címen található.
Az eddigiek bemutatására hajtsuk végre a következőket. A PLAY billentyűvel kapcsoljuk be a kazettás egységet: a szalag csévélődik, a motor jár.
Gépeljük be a következő utasításokat:

```
POKE 192,1: POKE 1,PEEK(1) OR 32
```

A kazettás egység motorja leáll, pedig nem nyomtuk le a STOP billentyűt. Az 1-es cím 5. bitje és a motorkapcsoló beállítódott. A motor programmal való újraindításához az alábbi sorra van szükség:

```
POKE 1,PEEK(1) AND 39: POKE 192,0
```

A kazettás egység ugyanúgy működik tovább, mint azt a beavatkozásunk előtt tette! Ezekkel azonban még koránt sem merítettük ki a motor vezérlésének lehetőségeit. Pl. lekérdezhetjük a kazettás egység billentyűinek állapotát is, ismét az 1-es cím segítségével:

```
IF PEEK(1)=55 THEN PRINT" NINCS LENYOMOTT BILLENTYÜ !"
IF PEEK(1)=7 THEN PRINT" BILLENTYÜ LENYOMVA !!"
```

E helyett a WAIT utasítást is használhatjuk:

WAIT 1,16

Ennek hatására a gép addig vár, míg a kazettás egység STOP billentyűjét le nem nyomtuk.

A vezérlőutasítások összefoglalva:
Motor ki: POKE 192,1: POKE 1,PEEK(1) OR 32
Motor be: POKE 1,PEEK(1) AND 39: POKE 192,0
Várakozás a STOP billentyűre: WAIT 1,16
Várakozás a PLAY billentyűre: X IF PEEK(1)=55 THEN GOTO X

2.2 A kazetta másolás elleni védelmének egyik módszere

Most pedig bemutatjuk, hogyan lehet megakadályozni egy másolás ellen védett kazetta problémamentes használatát. Ehhez a következőket kell tudni: ha egy programot a kazettára felveszünk, akkor a program azonosítására 172 karakter áll rendelkezésünkre. A program betöltésekor azonban csak 16 karakter jelenik meg.
A többi 160 karakter a kazettapufferben marad.
Az alábbi kis BASIC program csak a programnév 17. karakterét kérdezi le. Ez töltéskor nem jelenik meg a képernyőn, ismeretlen marad.
Ha tehát valaki a programot jogtalanul akarja lemásolni, akkor a programnév 17. karakterét nem veszi át, mert azt nem ismeri.

```
0 REM  **** P1. ****
1 :
2 :
10 IF PEEK(849)=ASC("X") THEN GOTO 30
20 PRINT"? LOAD ERROR":NEW
30 PRINT"EREDETI PROGRAM !"

READY.
```

Ezt a három sort tegyük a saját programunk elé. A program felvételekor a programnév 17. karakterébe tegyünk egy X-et.

SAVE "1234567890123456X" (az X pontosan a 17. helyen álljon!)

Ha most valaki le akarja másolni a programot, akkor a program felismeri, hogy a 17. karakter helyén nincs X, kiírja a ? LOAD ERROR-t, és törli a programot. Ez a másolás elleni védelem nem ismerhető fel!
Természetesen az X helyére tetszőleges karaktert írhatunk, de ekkor a 10-es

sort értelemszerűen módosítani kell. Vigyázzunk arra, hogy az ilyen védelemmel ellátott programunkat név nélkül, egyszerűen LOAD utasítással hívjuk! Tökéletesebb lesz a védelem akkor, ha a három ellenőrző sort a program elején listázás elleni védelemmel is ellátjuk (l. a Szoftvervédelem c. fejezet 3.3 pontjában.)

2.3 A kazettapuffer eltolása

A kazettapuffer alapállapotban a $033C–$03FB (828–1019) címeken található. A pufferbe kerülnek a kazettáról jövő, ill. az oda kimentett adatok, hogy blokkonként rendeződjenek.
A kazettapuffert csak a kazettával kapcsolatos adatforgalomra (SAVE, LOAD) használjuk. Egyébként ez a tárterület használaton kívüli.
Ezt a területet felhasználhatjuk kisebb gépi kódú programok, vagy a 13–15-ös sorszámú sprite blokkok tárolására.
A puffert addig használhatjuk más célra, amíg nincs szükségünk a kazettaegységre, vagyis nem akarunk valamilyen file-t tárolni vagy betölteni. Ha ez bekövetkezik, a puffer tartalma felülíródik.
Ezt egy nagyon egyszerű, de kevésbé ismert módon a kazettapuffer-vektorok ($B2–$B3, 178–179) segítségével gátolhatjuk meg. Írjuk át a vektorok tartalmát egy másik, mondjuk a $C000 címre. Ekkor az eredeti puffertartalom változatlan marad:

```
0 REM  **** M2. ****
1 :
2 :
10 A=49152: REM AZ UJ KAZETTAPUFFER KEZDOCIME
20 HB=INT(A/256)
30 LB=A-(HI*256)
40 POKE 178,LB: POKE 179,HB

READY.
```

Ha a $C000-val kezdődő terület is foglalt, próbálkozhatunk a képernyőtárral is. Ezt a területet biztosan nem használjuk a töltés és tárolás ideje alatt. Ehhez programunkban csak a 10-es sort kell megváltoztatni.

```
10 A = 1024
```

2.4 Betöltés csak kódokkal

A kazettás egység után most foglalkozzunk a 1541-es lemezegységgel is. A LOAD és SAVE parancsok végrehajtásakor a programnevet itt is a C 64-es tárolja. A kazettapuffer azonban ilyenkor érintetlen marad.

A gép ebben az esetben a $BB–$BC (187–188) mutatót használja, amely megadja, hogy hol kezdődik a programnév tárolása. A következő kis BASIC program futtatása után a képernyőn megjelenik az utolsó lemezművelethez használt programnév:

```
0 REM  **** P3. ****
1 :
2 :
10 AD= PEEK(187)+256*PEEK(188)
20 FOR A=AD TO 40959: B=PEEK(A)
30 NA$=CHR$(B)
40 NEXT:PRINT NA$

READY.
```

Ezt a lehetőséget a programnév tárolására ki tudjuk használni.
Írjuk a következő sorokat valamelyik programunk elé!

```
0 REM  **** P4. ****
1 :
2 :
10 FOR A=40955 TO 40959
11 B=PEEK(A)
12 READ C$: C=ASC(C$)
13 IF B=C THEN 15
14 PRINT"? LOAD  ERROR";:NEW
15 NEXT
16 DATA C,O,D,E,1
```

Tároljuk az így kiegészített programunkat egy pontosan 16 karakteres névvel. Most hívjuk be a programot a szokásos módon, majd indítsuk el. A képernyőn LOAD ERROR jelenik meg, és a program törlődik.
Töltsük be ismét a programot, de ebben az esetben úgy, hogy a programnév után írjuk oda a CODE1 jelszót is. Az így betöltött program már minden nehézség nélkül elindul. Hogyan lehetséges ez? Úgy, hogy az előbbi programrészlet lekérdezi a tárolt programnév utolsó 5 karakterét, és összehasonlítja a DATA sorban megadott jelszóval. Ha a kettő egyezik, akkor a program folytatódik.
Nagyon fontos, hogy a programnév pontosan 16 karakteres legyen. Ellenkező esetben további kódok nem írhatók hozzá.

2.5 A hibacsatorna kiolvasása

Az 1541-es típusú lemezegység használatakor gyakran előfordul, hogy valamilyen hiba hatására a piros LED villogni kezd. Ilyenkor gyakran sokáig keressük a hiba okát.
Az alábbi kis rutin a lemezegység hibacsatornájának kiolvasásával gyorsabbá

és egyszerűbbé teszi a hibakeresést. Felfedi a hibát, kiírja a képernyőre, megkönnyítve ezzel okának pontos kiderítését. Gépi kódú megfelelőjével a *Lemezegység trükkök* c. fejezetben még találkozunk.

```
0 REM  **** P5. ****
1 :
2 :
10 REM A LEMEZEGYSEG HIBACSATORNAJANAK KIOLVASASA
20 OPEN 1,8,15:REM A CSATORNA MEGNYITASA
30 INPUT#1,HSZ,HN$,S,SZ
40 PRINT HSZ","HN$","S","SZ
50 CLOSE 1

READY.
```

A változók:
HSZ A hiba sorszáma
HN$ A hiba megnevezése
S Sáv
SZ Szektor

2.6 Csináljunk a SAVE-ből LOAD-ot!

A következő vektorok és a velük való mesterkedések mind a kazettás egység, mind a 1541-es típusú lemezegység használata során érdekesek lehetnek. Most a SAVE ($0332–$0333; 818–819), valamint a LOAD ($0330–$0331; 816–817) vektorokkal fogunk foglalkozni. Ezek a mutatók a ROM-ban található SAVE és LOAD rutinok kezdőcímeit tartalmazzák.
A vektorok átírásával a SAVE-ből LOAD-ot tudunk csinálni (bár ennek semmi gyakorlati jelentősége nincs, de legalább megismerjük a működésüket!):

```
POKE 818,PEEK(816): POKE 819,PEEK(817)
```

Mostantól kezdve a SAVE már nem SAVE, hanem LOAD (VERIFY). Nem lehet egyszerű módon programot tárolni! Az alapállapot a következőképpen állítható vissza:

```
POKE 818,237: POKE 819,245   ($F5ED mutató)
```

Ugyanígy ki lehet kapcsolni a LOAD-ot is:

```
POKE 816,PEEK(818): POKE 817,PEEK (819)
```

Az alapállapot visszaállítása:

```
POKE 816,165: POKE 817,244   ($F4A5 mutató)
```

Ezek után a LOAD és SAVE felcserélése sem okoz gondot:

POKE 816,237: POKE 817,245: POKE 818,165: POKE 819,244

A LOAD és SAVE kikapcsolása:

POKE 818(816),26: POKE 819(817),167

Ezek után a LOAD és SAVE parancsnokra egyszerűen READY-vel jelentkezik a számítógépünk.

2.7 Automatikus utántöltés

Vegyük a következő esetet: írtunk egy programot, amely a futása során további programok vagy file-ok utántöltését igényli.
Ez nem jelent problémát, ha tudjuk, hogy a felhasználó kazettás egységet vagy lemezegységet fog használni. Természetesen megkérdezhetjük ezt a felhasználótól is, de ez mindkettőnknek kényelmetlen lenne. Ezt a problémát elegánsan az alábbi módon oldhatjuk meg:

450 LE=PEEK(186): OPEN 1,LE,1 vagy

670 LE=PEEK(186): LOAD"TEST",LE, (másodlagos cím)

Természetesen a számítógép sem mindentudó. Nem tudja előre megjósolni, hogy melyik készüléket fogjuk majd használni. Az előbbi példák abból indulnak ki, hogy a további tárolásra azt a készüléket fogjuk használni, amelyikről a főprogramot beolvastuk.

A 186-os cím az utoljára használt készülékszámot tartalmazza. A további készülékekre, mint pl. a nyomtatóra azonban ügyelni kell.
Ajánlatos a program elején a LE változó értékét lekérdezni. Így később más készülékek ezt az értéket nem változtatják meg.

Néhány szót a LOAD ERROR-ról

Előfordulhat, hogy egy kazettáról betöltendő program LOAD ERROR hibajelzést ad. Ez azonban nem minden esetben jelenti azt, hogy a betöltendő program a felhasználó számára elveszett.
A VC 20-as, de a Commodore 64-es is a programot kétszer egymás után tárolja a kazettán a SAVE utasítás végrehajtásakor. Betöltéskor az első változat bekerül a gépbe és összehasonlítódik a kazettán található második változattal. Ha a két változat különbözik egymástól, akkor LOAD ERROR hibajelzést kapunk.

Ha meg szeretnénk az ilyen programot menteni, meg kell vizsgálni, hogy kilistázható-e. Ha igen, akkor a program megmenthető.
A kazettapuffer 831–832-es címe a betöltött program hosszát tartalmazza. Ezt az információt csak akkor veszi át a puffer a számítógéptől, ha a betöltés hibátlan volt.
Ha LOAD ERROR hibajelzés keletkezik, akkor a számítógép nem rendelkezik ezekkel az információkkal.

A következő sorok parancs-üzemmódban való beadásával a hibásan betöltött programot megmenthetjük (amennyiben kilistázható):

POKE 46,PEEK(832): POKE 47,PEEK(831): POKE 48,PEEK(832):
POKE 49,PEEK(831): POKE 50,PEEK(832)

Ezek után a programunk RUN-nal indítható!

2.8 LOAD és SAVE a gépi kódú programokban

A gépi kódú programok szalagon való tárolásának van egy kényelmes, de kevésbé használt formája: a gépi kódú program közvetlen tárolása a szalagon. A BASIC-betöltővel szemben ennek az a nagy előnye, hogy gyorsabb, és felszabadul a BASIC betöltőprogram által használt memóriatartomány is.
Saját, általunk meghatározott karaktersorozatokat is tárolhatunk ilyen módon. Ehhez azonban tudnunk kell, hogy a programunk hol kezdődik, ill. hol végződik. Tételezzük fel, hogy a $5000–$6000 tartományt kívánjuk tárolni.
Tároláskor a szalagra a BASIC terület kezdete (43–44) és a változók kezdete (45–46) közötti területen talált információk kerülnek. Tehát a $5000-t a 43–44, a $6000-t a 45–46 vektorba kell beírnunk.

$5000 = #20480 LB = 0 HB = 80
$6000 = #24576 LB = 0 HB = 96

Így a $5000–$6000 tartomány a következő parancsokkal vihető ki:

POKE 43,0: POKE 44,80: POKE 45,0: POKE 46,96: CLR: SAVE "(név)",1,1

Az első 1-es a SAVE után a készülékszámot jelöli (lemezegység esetén 8).
A második 1-es azt jelenti, hogy a tartományt közvetlenül tároltuk, vagyis egy újabb betöltéskor automatikusan ugyanarra a helyre íródik vissza. Ezenkívül a számítógép ezt a területet nem BASIC programként kezeli, és ezért nem számítja a BASIC LINK-eket.

Tételezzük fel, hogy programunkat a $5000–$6000 tartományban tároltuk, és a $8000–$9000 tartományba kívánjuk betölteni.
Ez minden gond nélkül lehetséges.
Adjuk meg a SYS 63276 ($F72C) parancsot.
A képernyőn megjelenik a PRESS PLAY ON TAPE felirat. Kövessük a felszólítást. A SYS hatására csak a szalagfej (Tape-Header) töltődik be. Itt a fejben van megadva, hogy a program mettől meddig tart. Ezt a következőképpen változtathatjuk meg:

POKE 829,LB (kezdőcím)
POKE 830,HB (kezdőcím)
POKE 831,LB (végcím) + 2
POKE 832,HB (végcím)

A $8000–$9000 tartományt így kell megadni:

POKE 829,0: POKE 830,128: POKE 831,0: POKE 832,144 SYS 62849 ($F581)

Az utolsó parancs hatására folytatódik a betöltés. A betöltés végeztével adjunk ki egy NEW parancsot, és indíthatjuk a programot. Ez a trükk természetesen csak a kazettás egység esetén alkalmazható, hiszen a lemezegység nem használja a kazettapuffert. A lemezegységre vonatkozó címeket nem itt közöljük.

2.9 Az ASCII- és a képernyőkódok egymásba transzformálása

Mindnyájan találkoztunk már a C 64-es kézikönyvének függelékében a képernyőkódok és ASCII-kódok táblázatával. A képernyőkódokat a különböző jelek képernyőtárolóba ($0400; 1024) való bevitelére használtuk. Ehhez POKE utasításokat alkalmaztunk. Az ASCII-kódokat a PRINT utasítással kiíratható füzérek megváltoztatásához használtuk (ASC–CHR$).
Mindkét táblázatban ugyanazok a jelek vannak, de sajnos a kódjaik különböznek egymástól.
Néhány programban az ASCII-kódokat képernyőkódokká kell alakítani (lásd pl. a monitorprogramokat). Ezt a következő kis BASIC rutinnal valósíthatjuk meg:

```
0 REM  **** P6. ****
1 :
2 :
10 REM ASCII-KODBOL KEPERNYOKOD
20 T$="A": T=ASC(T$): PRINT"ASCII-KOD =";T
30 PRINT"KARAKTER = ";T$
40 BS=256*PEEK(648): REM A KEPERNYOTAROLO KEZDETE
50 KK=PEEK(BS):REM KK=KEPERNYOKOD
60 PRINT"KEPERNYOKOD=";KK
70 END
```

A T$ füzér tartalmazza a transzformálandó kódot, példánkban az A betűt. Az ASCII-kódot ASC-vel határoztuk meg és írtuk ki. A képernyőkód meghatározásához először a képernyőtároló kezdőcímét kérdeztük le (a 648-as cím a video-RAM felső byte-ja).
A T$-ben levő betűt a képernyőtároló első pozíciójába PRINT utasítással beírtuk (30-as sor). PEEK utasítással kiolvastuk ezt a címet, és így megkaptuk a képernyőkódot.
A rutin hátránya a képernyőtároló nem praktikus használata. A transzformálandó jel a képernyő bal felső sarkában jelenik meg.
Most lássunk két másik módszert, amely a transzformálást számítással oldja meg:

```
0 REM  **** P7. ****
1 :
2 :
10 REM KEPERNYOKODBOL ASCII-KOD
20 INPUT"KEPERNYOKOD";KK
30 KK=KK AND 127
40 IF KK AND 32 THEN 70
50 IF KK AND 64 THEN AK=KK OR 32: GOTO 90
60 AK=KK OR 64: GOTO 90
70 IF KK AND 64 THEN AK=KK AND 63 OR 12: GOTO 90
80 AK=KK
90 PRINT "ASCII-KOD=";AK
100 END

READY.

0 REM  **** P8. ****
1 :
2 :
10 REM ASCII-KODBOL KEPERNYOKOD
20 INPUT " ASCII-KOD";AK
30 IF AK AND 128 THEN KK=AK AND 127 OR 64: GOTO 70
40 IF NOT AK AND 64 THEN KK=AK: GOTO 70
50 IF AK AND 32 THEN KK=AK AND 95: GOTO 70
60 KK=AK AND 63
70 PRINT "KEPERNYOKOD =";KK

READY.
```

A változók:

AK ASCII-kód
KK Képernyőkód

Kényelmes kódtörlés

Gyakran szükségünk van arra, hogy a képernyő egy bizonyos részét kitöröljük. Ezt természetesen egy PRINT utasítással és a kurzorvezérlő billentyűkkel bármikor megtehetjük. A következő megoldás azonban sokkal egyszerűbb és

áttekinthetőbb, ui. nem igényli az aktuális kurzorpozíció ismeretét. A vezérlőbillentyűkre tehát nincs szükség. Egy kész, a ROM-ban meglevő rutint fogunk használni, a $E9FF (59903) rutint, amely egy képernyősort töröl.
A rutin BASIC-ből a következőképpen hívható:

```
POKE 781,X: SYS 59903
```

X = 0–24

Az X regiszterbe (a címe 781) a törölni kívánt sor számát kell beírni, vagyis egy 0 és 24 közé eső számot. A többit a rutin elvégzi.

Egy kis ügyeskedéssel ROM rutinunkat még sokoldalúbbá tehetjük. Ugorjuk át az első két byte-ot (az új ugrási cím $EA01; 59905). Adjuk meg a Z regiszterben (a címe 782) a törölni kívánt jelek számát, amelyeket az X regiszterben megadott sorban ki akarunk törölni.

```
POKE 781,X: POKE 782,Z: SYS 59905
```

X = 0–24, Z = 0–39 (0–255)

2.10 Hexadecimális bevitel

Gépi kódú programok bevitelére több lehetőségünk van:

a) monitor;
b) BASIC betöltőprogram;
c) hexadecimális jelek.

Ismerkedjünk meg egy programmal, ami a hexadecimális jeleket használja. Valószínűleg mindnyájunkkal előfordult már, hogy nagyon megtetszett egy számítástechnikai szakfolyóiratban talált program, de BASIC betöltőprogram helyett csak néhány oldal hexadecimális számot tartalmazott. Mit tehetünk ilyenkor?
A következő programmal megoldhatjuk ezt a problémát. Gépeljük be, és próbáljuk ki!
A gép először a jelek számát kéri soronként, vagyis azt, hogy egy sorba hány hexadecimális számot kívánunk írni. Ez a 11-et nem lépheti túl. Előfordulhat, hogy a dokumentáció hosszabb sorokat használ. Írjuk át ekkor a program megfelelő adatát a szükséges értékekre. Ügyeljünk azonban az áttekinthetőségre!
Ezután a gép megkérdezi a kezdőcímet. Ezt hexadecimálisan kell megadni, 4 karakter hosszúságban. A program kiírja a mindenkori első elem címét a tárolóban, nekünk csak a számokat kell bevinni.

A programot RETURN-nel szakíthatjuk meg.

```
0 REM  **** P9. ****
1 :
2 :
5 PRINT"■■          HEXADECIMALIS BEVITEL"
10 INPUT"■■■SZAM/SOR:";SW:IFSW<10RSW>11THEN10
20 INPUT"KEZDOCIM:";A$
30 IFLEN(A$)<>4THENPRINT"HIBAS BEVITEL":GOTO20
40 POKE19,0:PRINT:Y=3:FORZ=1TO4:B$=MID$(A$,Z,1):B=ASC(B$)
50 IFB>47ANDB<58THENSA=SA+VAL(B$)*16↑Y:GOTO80
60 IFB>64ANDB<71THENSA=SA+(ASC(B$)-55)*16↑Y:GOTO80
70 PRINT"HIBAS BEVITEL":GOTO20
80 Y=Y-1:NEXTZ
100 PRINT"■"SA;:FORZ1=0TOSW-1:FORZ2=1TO0STEP-1
110 POKE204,0:GETA$:IFA$=""THEN110
120 POKE207,0:B=ASC(A$):IFB>47ANDB<58THENPE=PE+VAL(A$)*16↑Z2:
    GOTO160
130 IFB>64ANDB<71THENPE=PE+(ASC(A$)-55)*16↑Z2:GOTO160
140 IFB=13THENEND
150 GOTO110
160 PRINTA$;:NEXTZ2:PRINT" ";:POKESA+Z2,PE:PE=0:NEXTZ1
170 SA=SA+SW:PRINT:GOTO100
```

2.11 DATA generátor

Ez a program kiírja a tárolóban található gép kódú programot DATA sorokban. Nekünk meg kell adni a gépi kódú program kezdőcímét, a hosszát, és azt a sorszámot, amelytől kezdve a DATA sorokat kérjük. Ez a sorszám ne legyen 221 alatt, mert akkor a DATA sor felülírja a programot.
A DATA sorok lépésenként íródnak ki. A program minden DATA sor után félbeszakad, és beolvassa a változókat a speciális memóriacímekről:

828–829 A gépi kódú program kezdete
830–831 A program hossza
832–833 A pillanatnyi sorszám

A CLR-HOME és a RETURN leütésével a sor bekerül a programba. Ha ezután beadjuk a RUN 100 parancsot és a RETURN-t, a program a 100-as sorral indul újra.
A 15-ös sorban levő értékek a billentyűzetpufferbe kerülnek. A 20-as sor a LOW és HIGH byte-ot számolja.
Lehetséges a változókat magukba a DATA sorokba beírni, és minden futáskor onnan kiolvasni. Gyakorlásképpen írjuk át a programot, hogy így működjön!

```
0 REM **** P10. ****
1 :
2 :
10 GOTO25
15 DATA19,13,82,213,49,48,48,13
20 HB=INT(X1/256):LB=X1-256*HB:RETURN
```

```
25 PRINT"■■■■■■■■■■■■ DATA   GENERATOR ■"
26 PRINT"■■HOJJH MEG A KOVETKEZO PARAMETEREKET:■■"
30 INPUT"■PROGRAMKEZDET";PK
40 INPUT"■PROGRAMHOSSZ";PH
50 INPUT"■KEZDOSOR";HZ
60 X1=PK-1:GOSUB20:POKE828,LB:POKE829,HB
70 X1=PH:GOSUB20:POKE830,LB:POKE831,HB
80 X1=HZ:GOSUB20:POKE832,LB:POKE833,HB
100 X=PEEK(832)+256*PEEK(833)
110 Y=PEEK(828)+256*PEEK(829)
120 Z=PEEK(830)+256*PEEK(831)
130 PRINT"■"X" DATA";
140 FORZ1=1TO10:PRINTPEEK(Y+Z1)"■,";
150 Z=Z-1:IFZ=0THENPRINT"■ ":GOTO220
160 NEXTZ1:PRINT"■ "
170 X=X+1:Y=Y+10
180 X1=Y:GOSUB20:POKE828,LB:POKE829,HB
190 X1=Z:GOSUB20:POKE830,LB:POKE831,HB
200 X1=X:GOSUB20:POKE832,LB:POKE833,HB
210 FORZ1=631TO638:READA:POKEZ1,A:NEXTZ1:POKE198,8:END
220 POKE631,19:POKE632,13:POKE198,2:END

READY.
```

2.12 Képernyőmásolás (SCREEN-COPY)

Saját készítésű programjainkban gyakran okoznak nehézséget a PRINT utasítások, mivel a velük való munka

a) kevésbé áttekinthető és
b) lassú.

A SCREEN-COPY program ezen a gondon kíván segíteni. Gépeljük be a programot, és indítsuk el

RUN 5-tel (!)

Indulás után megjelenik a kurzor, és elkészíthetjük az általunk tervezett képernyőmaszkot. Ezután menjünk a kurzorral az utolsó sorba és nyomjuk le a RETURN-t.
Kis idő múlva a program megkérdezi a PRINT utasítások kezdő sorszámát. Adjuk be a megfelelő értéket.
Ezután kényelmesen dőljünk hátra, lazítsunk, és a program máris adja az eredményt. Sorról sorra kiírja a képernyőtartalmakat és elhelyezi őket a programsorokban. A program befejeztével az általunk tervezett képernyő soronként, PRINT utasítások formájában jelenik meg.
Megvan az a lehetőségünk is, hogy ezután egy újabb képernyőmaszkot tervez-

zünk, és egy másik kezdősorszámmal tároljuk. Ha készen vagyunk, kitörölhetjük a SCREEN-COPY programot, és csak a PRINT utasításokat hagyjuk meg.

A program törlésének ismert módja, hogy a törölni kívánt sorszámokat sorra beütjük és egyszerűen RETURN-t nyomunk. Ennél egy sokkal praktikusabb megoldást szeretnénk most bemutatni. Írjuk be a következő programot:

```
0 REM  **** P11. ****
1 :
2 :
3 GOTO30
29 POKE254,4
30 X=PEEK(254):PRINT"□"X
31 X=X+1:POKE254,X:IFX=30THENPRINT"SCREEN-COPY TORULVE":END
32 POKE631,19:POKE632,13:POKE633,82:POKE634,213:POKE635,13:
   POKE198,5:END
```

RUN 29

A SCREEN–COPY program villámgyorsan kitörlődik, és csak a 0,26,27,28 és 29 sorszámú sorokat kell kézzel kitörölni. Ezután már csak a PRINT soraink maradnak meg.
Ez a megoldás a képernyőtervezés kényelmes módszere, de van egy hátránya. A kurzor színváltásait, a REVERS ON és a REVERS OFF utasításokat csak utólag fűzhetjük a megfelelő utasításokhoz. Ez a hátrány kiküszöbölhető ugyan, de a többletmunka a felhasználás során nem térülne meg.

```
0 REM  **** P12. ****
4 GOTO13
5 PRINT"□":OPEN1,0
6 INPUT#1,A$:IFPEEK(214)=24THEN8
7 PRINT:GOTO6
8 FORZ=0TO999:POKE2456?+Z,PEEK(1024+Z):NEXTZ
9 INPUT"□KEZDOSOR";AZ:IFAZ<30THEN9
10 HB=INT(AZ/256):LB=AZ-256*HB
11 POKE828,LB:POKE829,HB
12 POKE830,0:POKE831,96
13 SP=PEEK(830)+256*PEEK(831)
14 FORZ=0TO39:H=PEEK(SP+Z)
15 GOSUB24
16 KS$=KS$+CHR$(B):NEXTZ
17 ZE=PEEK(828)+256*PEEK(829)
18 PRINT"□"ZE"?"CHR$(34);KS$;CHR$(34)";"
19 SP=SP+40:IFSP>25536THEN23
20 HB=INT(SP/256):LB=SP-256*HB:POKE830,LB:POKE831,HB
21 ZE=ZE+10:HB=INT(ZE/256):LB=ZE-256*HB:POKE828,LB:POKE829,HB
22 POKE631,19:POKE632,13:POKE633,82:POKE634,213:POKE635,13:
   POKE198,5:END
23 POKE631,19:POKE632,13:POKE198,2:END
24 H=AAND127:IFAAND32THEN27
25 IFAAND64THENB=AOR32:RETURN
26 B=AOR64:RETURN
27 IFAAND64THENB=AAND63OR128:RETURN
28 B=A:RETURN
```

Mivel ez a program már bonyolultabb, rövid magyarázatot fűzünk hozzá:

5–7. sor A képernyőt file-ként megnyitjuk. A 6-os sorban megvizsgáljuk, hogy a kurzor a RETURN lenyomásakor az utolsó sorban volt-e.

8. sor A képernyőtartalmat tároljuk, mivel ez a továbbiakban meg fog változni.

11. sor A kezdő sorszámot bevisszük, a helyességét megvizsgáljuk. A sorszámot szétbontjuk alsó és felső byte-ra, és a kazettapuffer 828–829 címeire beírjuk.

12. sor A tárolt képernyőtartalom kezdőcímét felbontjuk alsó és felső byte-ra.

16. sor A képernyő egy sorából egy füzért képezünk.

17. sor A sorszámot kiszámítjuk.

18. sor A kész programsort kiírjuk a képernyőre.

9–21. sor Kiszámítjuk az új sorszámot, a képernyőtartalom új címét, ezeket balsó és felső byte-ra bontjuk és tároljuk.
Ezeket az értékeket azért kell tárolni, mert a program önmagát szakítja meg, és egy új indításkor a változótartalmak elvesznének.

22. sor A programfutás megszakad, és az éppen most keletkezett sort önmagához írja. A kurzor a bal felső sarokba ugrik, és a program RETURN utasítást kap. Ezután a program ismét automatikusan indul, mert RUN parancsot kapott.
Ezek a műveletek akkor valósulnak meg, ha a megfelelő ASCII-értékeket POKE utasítással a billentyűzetpufferbe bejuttatjuk.

23–28. sor Ezzel a rutinnal a képernyőkódokat ASCII-kódokká alakítjuk. Ezt a transzformációt azért kell végrehajtani, mert a képernyőn és annak tárolt formájában képernyőkódokat használunk. A keletkezett képernyősorokat (KS$) azonban ASCII-kódban kell megkapnunk.
Ez a rutin természetesen más programokban is felhasználható.

2.13 BASIC tippek

Ez a fejezet a haladó BASIC programozók számára valószínűleg tartogat néhány meglepetést. Mindenképpen ajánljuk elolvasni, mert bizonyára talál benne az Olvasó olyan újdonságot, amit érdemes tudni.
Aki már írt BASIC nyelvű játékprogramot, sokat bosszankodhatott a BASIC nagy hátrányán, a lassúságán. Néha azonban a programozók is tehetnek erről...
Foglalkozzunk most néhány olyan trükkel, amellyel meggyorsíthatjuk BASIC programjaink futását.

2.13.1 Használjunk pontot a nulla helyett!

Talán meglepő, de igaz: ha a számítógép egyedülálló decimális pontot talál a BASIC programban, akkor ezt a lebegőpontos aritmetika automatikusan nullának veszi. Hogyan használhatjuk a decimális pontot mégis a nulla helyett? A nulla a BASIC programokban a számítógép számára nem egy szám, hanem csak egy ASCII-érték. Ezt az ASCII-értéket először egy számmá kell alakítani, amihez időre van szükség. Ha tehát a nulla helyett decimális pontot használunk, akkor a számítógépnek nem kell végrehajtani a transzformálást, így ezzel időt takarítunk meg.

Pl.:

```
0 REM  **** P13.  ****
1 :
2 :
10 TI$="000000"
20 FOR X=1TO100
30 A=A+0
40 NEXT X
50 PRINTTI

READY.

0 REM  **** P14.  ****
1 :
2 :
10 TI$="000000"
20 FOR X=1TO100
30 A=A+.
40 NEXTX
50 PRINTTI

READY.
```

Még gyakrabban használhatjuk a decimális pontot az IF utasításokban. A

```
10 IFA=OTHEN 20
```

helyett vegyük a következő sort:

```
10 IFA=.THEN 20
```

Gondoljunk csak vissza, hányszor használtuk már az első változatot, és mennyi időt fecséreltünk el vele!

2.13.2 Használjunk változókat az ASCII-kódok helyett!

Ha egy számot leírunk, akkor az a BASIC tárolóban ASCII-kódként jelenik meg. Pl.:

```
10 POKE 1024,1
```

A parancs végrehajtásához a számítógép az ASCII-füzért először integer (egész típusú), majd valós számmá alakítja. (A számítógép kizárólag ilyen típusú számokkal tud számolni.) A transzformáció meglehetősen hosszú időbe telik. Találnunk kell tehát olyan megoldást, amelyben az ASCII-füzér rögtön valós számmá alakul át. Ezt a célt szolgálják a változók.
Pl.:

```
0 REM  **** P15.  ****
1 :
2 :
10 TI$="000000"
20 FOR X=1TO100
30 A=A+100
40 NEXTX
50 PRINTTI

READY.


0 REM  **** P16.  ****
1 :
2 :
10 TI$="000000":B=100
20 FOR X=1TO100
30 A=A+B
40 NEXTX
50 PRINTTI

READY.
```

A különbség a példákból jól megfigyelhető. Különösen igaz ez a nagy számokra, amelyek ASCII-füzérje mindig hosszabb lesz, míg a valós szám állandóan ugyanolyan hosszú marad (csak az egyes byte-ok értéke változik meg).

2.13.3 Használjunk az egész számok helyett valósakat!

Az előző pontban már láthattuk, hogy a valós számmá való transzformálás időigényes.
Ha integer változót használunk (pl. A%), akkor ezt valóssá kell alakítani. Ezért célszerűbb azonnal valós változót használni (pl. A).

```
0 REM  **** P17.  ****
1 :
2 :
10 TI$="000000"
20 B%=100:C%=50:D%=25
30 FOR X=1TO100
40 A%=A%+B%-C%-D%
50 NEXTX
60 PRINTTI

READY.


0 REM  **** P18.  ****
1 :
2 :
10 TI$="000000"
20 B=100:C=50:D=25
30 FOR X=1TO100
40 A=A+B-C-D
50 NEXTX
60 PRINTTI

READY.
```

2.13.4 Használjunk FOR–NEXT ciklusokat!

Mint tudjuk, a FOR–NEXT ciklusok sokkal gyorsabbak, mint az IF utasítással szervezett ciklusok.
Pl.:

```
0 REM  **** P19.  ****
1 :
2 :
10 TI$="000000":X=1
20 A=A+X
30 X=X+1:IFX<>101THEN20
40 PRINTTI

READY.


0 REM  **** P20.  ****
1 :
2 :
10 TI$="000000"
20 FORX=1TO100
30 A=A+X
40 NEXTX
50 PRINTTI

READY.
```

2.13.5 A szubrutinokat a program elejére tegyük!

A szubrutinokat célszerű a program elejére tenni, mert a számítógép a megadott sorszámot mindig a program elejétől kezdve keresi. Minél később találja meg a gép a sorszámot, annál hosszabb ideig fut BASIC program.
Pl.:

```
0 REM  **** P21.  ****
1 :
2 :
10 TI$="000000"
20 FORX=1TO100
30 GOSUB100
40 NEXTX
50 PRINTTI
60 END
100 A=A+X
110 RETURN

READY.
```

```
0 REM  **** P22.  ****
1 :
2 :
10 A=A+Z
20 RETURN
100 TI$="000000"
110 FORX=1TO100
120 GOSUB10
130 NEXTX
140 PRINTTI

READY.
```

A második programot RUN 100 paranccsal kell indítani.
A bemutatott programok közötti különbség nem figyelhető meg eléggé, mivel a programok nagyon rövidek, és ezért az első programban sem kell nagyon sok sorszámot végignézni. Az összehasonlítást egyszer hosszabb programok esetében is érdemes kipróbálni.

2.14. Az ESCAPE funkció

Bizonyára mindnyájunk előtt ismeretes, hogy idézőjelek között a számítógép más üzemmódban (angolul Quote Mode) dolgozik. A kurzorvezérlő billentyűk lenyomásakor pl. kurzorvezérlő jelek jelennek meg a képernyőn. Ugyanez a helyzet a színváltó billentyűkkel is. A vezérlőjelek hatása csak a PRINT utasítás végrehajtásakor érvényesül.

Ez nem is okoz problémát, ha pontosan tudjuk, hogy mit akarunk. Ha viszont hibázunk, akkor elég körülményes ebből az üzemmódból kilépni, és a hibát kijavítani.
A következő kis rutin ehhez nyújt segítséget. Az F1 funkcióbillentyű lenyomásával a számítógép az idézőjel-üzemmódból áttér a normál üzemmódra.
Ezt természetesen fordítva is megtehetjük. Ha vissza akarunk térni az idézőjel-üzemmódba, akkor az F3 funkció billentyűt kell lenyomni.
A két üzemmód megkülönböztetésére idézőjel-üzemmódban a képernyő jobb felső sarkában egy idézőjel jelenik meg.

```
0 REM  **** P23.  ****
1 :
2 :
5 REM ESCAPE
10 FORI=40704TO40795
20 READA
30 S=S+A
40 POKEI,A
50 NEXT
60 IFS<>11008THENPRINT"?HIBA A DATAKBAN":END
70 PRINT"DATAK RENDBEN"
80 SYS40704
90 DATA120,169,15,141,20,3,169,159,141
100 DATA21,3,133,56,88,96,72,165,203
110 DATA201,4,240,34,201,5,240,59,165
120 DATA212,208,4,165,216,240,13,169,34
130 DATA141,39,4,169,1,141,39,216,76,52
140 DATA159,169,32,141,39,4,104,76,49
150 DATA234,165,212,240,18,169,0,133
160 DATA212,133,216,169,20,141,119,2
170 DATA169,1,133,198,76,52,159,165,216
180 DATA240,226,76,60,159,169,255,133
190 DATA216,76,52,159

READY.
```

A program gépi kódú listáját is közreadjuk:

```
0 REM  **** P24.  ****
1 :
2 :
3 SYS49152
4 .OPT OO,P1
5 *= $9F00
10 SEI
20 LDA #$0F
30 STA $0314
40 LDA #$9F
50 STA $0315
60 STA $38
70 CLI
80 RTS
90 PHA
```

```
100 LDA $CB
110 CMP #$04
120 BEQ $9F38
130 CMP #$05
140 BEQ $9F55
150 LDA $D4
160 BNE $9F22
170 LDA $D8
180 BEQ $9F2F
190 LDA #$22
200 STA $0427
210 LDA #$01
220 STA $D827
230 JMP $9F34
240 LDA #$20
250 STA $0427
260 PLA
270 JMP $EA31
280 LDA $D4
290 BEQ $9F4E
300 LDA #$00
310 STA $D4
320 STA $D8
330 LDA #$14
340 STA $0277
350 LDA #$01
360 STA $C6
370 JMP $9F34
380 LDA $D8
390 BEQ $9F34
400 JMP $9F37
410 LDA #$FF
420 STA $D8
430 JMP $9F34

READY.
```

2.15 A képernyő színeinek megváltoztatása

Előfordulhat, hogy valaki nem tudja elolvasni a képernyőn levő szöveget, mert a képernyő színei (a keretszín, a háttérszín és a betűszín) helytelenül vannak megválasztva. Különösen a fekete-fehér képernyőket nehéz olvasni. Kis gépi kódú programokkal segíthetünk ezen a problémán.
Természetesen a BASIC programokban (csaknem mindegyikben) és parancsüzemmódban is megváltoztathatjuk a színeket. A keretszínt az F1, a háttérszínt az F3 billentyű lenyomásával változtathatjuk. Rutinunkat a megszakító rutinok kezelik. Ez az oka annak, hogy a funkcióbillentyűk lenyomásakor egy villanást észlelünk. A rutin e két billentyű lenyomását figyeli.
Ha lenyomjuk valamelyik billentyűt, akkor a rutin a keret- vagy a háttérszín

kódját eggyel növeli. Az IRQ rutinok másodpercenként 50–60-szor szakítják meg a programot. Ez azt jelenti, hogy egy billentyű lenyomásakor a színkód 50–60-szor változik.
Programunk rövidítésére lemondtunk arról, hogy a villogást kiküszöböljük.
A helyes színkombináció kialakítására a megfelelő billentyűket többször le kell nyomni.
Ha rutinunkat egy másik programmal egyszerre akarjuk használni, akkor először töltsük be a rutint és indítsuk el. Ezután töltsük be a megfelelő programot. A két program egyidejű használata csak akkor okozhat gondot, ha a második program túl hosszú, vagy használja a 40704-es címtől kezdődő tárterületet, vagy éppen az F1, F3 funkcióbillentyűkkel dolgozik.
Nézzük először a BASIC betöltőprogramot:

```
0 REM  **** P25.  ****
1 :
2 :
10 FOR I=40704 TO 40745
20 READ A
30 POKE I,A
40 S=S+A
50 NEXT I
60 IF S<>4625 THEN PRINT "HIBA A DATA SORBAN..!!":END
70 PRINT"OK..!!"
80 SYS 40704
90 DATA 120,169,15,141,20,3,169,159,141
100 DATA 21,3,133,56,88,96,72,165,203
110 DATA 201,4,240,8,201,5,240,10,104,76
120 DATA 49,234,238,32,208,76,26,159,238
130 DATA 33,208,76,26,159

READY.
```

A szakembereket bizonyára a gépi kódú lista is érdekli, ami hasznos lehet a programozásban kevésbé jártasak számára is. Ezért most következzék a gépi kódú lista:

```
0 REM **** P26. ****
1 :
2 :
3 SYS49152
4 .OPT OO,P1
5 *= $9F00
10 SEI              ;MEGSZAKITAS K1
20 LDA #$0F         ;A VEKTOROK MEGVALTOZTATASA
30 STA $0314
40 LDA #$9F
50 STA $0315
60 STA $38          ;A RUTIN BLOKKOLASA
70 CLI              ;MEGSZAKITAS BE
80 RTS              ;AZ 1. RESZ VEGE
90 PHA              ;A REGISZTER KIMENTESE
```

```
100 LDA $CB          ;BILLENTYUZET-LEKERDEZES
110 CMP #$04         ;F1
120 BEQ $9F1E
130 CMP #$05         ;F2
140 BEQ $9F24
150 PLA
160 JMP $EA31        ;HZ IRQ FOLYTATASA
170 INC $D020        ;H KERETSZIN NOVELESE
180 JMP $9F1A
190 INC $D021        ;H HATTERSZIN NOVELESE
200 JMP $9F1A

READY.
```

Ha a BASIC program az F1 és F3 funkcióbillentyűket használja, akkor más billentyűket is alkalmazhatunk. A 110-es sorban az F1-nek 4, az F3-nak 5 felel meg. A függelékben található billentyűzet-dekódolási táblázatból könnyen választhatunk új értékeket.

2.16 Két képernyő

A most bemutatásra kerülő rutin két képernyő széles körű használatát teszi lehetővé számunkra. Különösen a grafikus megjelenítésnél (kis felbontásúról nagyfelbontású grafikára való váltás) vehetjük hasznát. Példánkban csak a kis felbontású, normál grafikára térünk ki, mert ez könnyebben érthető. Akinek kedve van, később átalakíthatja úgy a rutint, hogy a nagyfelbontású grafikát is kezelje.
A két képernyő használatával sok érdekes dolgot művelhetünk. Pl. a kalandjátékoknál felépíthetjük a második képernyőt, amíg a játékos az első előtt játszik. De ezenkívül még sok területen, pl. a szövegfeldolgozások során is komoly előnyei vannak. Az emberi fantázia nem ismer határokat.
Most pedig konkrétan a programról néhány szót.
Gépeljük be és tároljuk a programot. Nagyon fontos, hogy a program elindítása után adjunk ki egy NEW parancsot (l. a gépi kódú listát). Egy megjegyzés a gépi kódban nem programozók számára: ha nem adjuk ki a NEW parancsot, akkor a program bevitelekor a számítógép üzemképtelenné válik. A második képernyőt az F1 billentyűvel hívhatjuk elő. Átkapcsolás után a kurzor eltűnik, de ez nem a számítógép üzemképtelenségét jelzi. A RETURN vagy más billentyű lenyomásával a kurzor ismét megjelenik. Az első átkapcsoláskor a második képernyő tele van inverz karakterekkel, mivel ez a terület átkapcsolás előtt a BASIC területhez tartozott. Töröljük egyszerűen a képernyőt, és az máris rendelkezésünkre áll.
A két képernyő színtárolója megegyezik egymással. Tehát a színinformációk az egyik képernyőről átkerülnek a másikba. Ennek megakadályozására a rutinba egy olyan részt építettünk be, amelynek hatására az egész szín-RAM a

kurzor pillanatnyi színének megfelelő értékre állítódik. A második képernyő tárolója a 2048($0800) – 3047($0BE7) címek között van.
Tudnunk kell, hogy a sprite mutatók címeit is megváltoztattuk. Ezek jelenleg a 3064–3071-es címeken találhatók. A jobb áttekinthetőség érdekében egy táblázatot is készítettünk:

		Rutinnal	
	Alapállapot	1. képernyő	2. képernyő
A képernyőtároló kezdőcíme	1024 ($0400)	1024 ($0400)	2048 ($0800)
A képernyőtároló végcíme	2023 ($07E7)	2023 ($07E7)	3047 ($0BE7)
Sprite mutatók	2040 ($07F8)	2040 ($07F8)	3064 ($0BF8)
A BASIC tároló kezdőcíme	2048 ($0800)	3072 ($0C00)	3072 ($0C00)
Színtároló	55296 ($D800)	55296 ($D800)	55296 ($D800)
A különböző állapotok elérése	SYS 64738	F3 lenyomása	F1 lenyomása

Rutinjainkat játékprogramok készítésénél is jól lehet használni: erre két lehetőség van:

a) fel kell szólítani a felhasználót az F1, ill. az F3 billentyű lenyomására;
b) POKE 203,4 a 2. képernyő használatához
POKE 203,5 az 1. képernyő használatához

Ezek után lássuk a BASIC betöltőprogramot:

```
0 REM  **** P27. ****
1 :
2 :
5 REM 2 KEPERNYO
10 FOR I=40702 TO 40787
20 READ A
30 POKE I,A
40 S=S+A
50 NEXT I
60 IF S=11070 THEN PRINT"OK..!!":POKE3072,0:SYS40702:END
70 PRINT"HIBA A DATA SORBAN..!!"
80 END
90 DATA169,12,133,44,120,169,17,141,20
100 DATA  3,169,159,141, 21,  3,133, 56, 88, 96
110 DATA165,203,201,  4,208, 13,169, 37,141
```

```
120 DATA 24,208,169,  8,141,136,  2, 76, 50
130 DATA159,201,  5,208, 36,169, 21,141, 24
140 DATA208,169,  4,141,136,  2,169,216,133
150 DATA252,160,  0,132,251,173,134,  2,145
160 DATA251,200,240, 13,192,232,208,247
170 DATA166,252,224,219,208,241, 76, 49
180 DATA234,230,252, 76, 61,159

READY.
```

A gépi kódú programlista:

```
0 REM  **** P28. ****
1 :
2 :
3 OPEN1,8,1,"@0:P 28G"
4 OPEN2,4
5 SYS49152
6 .OPT O1,P2
7 *= $9EFE
10 LDA #$0C
20 STA $2C
30 SEI
40 LDA #$11
50 STA $0314
60 LDA #$9F
70 STA $0315
80 STA $38
90 CLI
100 RTS
110 LDA $CB
120 CMP #$04
130 BNE $9F24
140 LDA #$25
150 STA $D018
160 LDA #$08
170 STA $0288
180 JMP $9F32
190 CMP #$05
200 BNE $9F4C
210 LDA #$15
220 STA $D018
230 LDA #$04
240 STA $0288
250 LDA #$D8
260 STA $FC
270 LDY #$00
280 STY $FB
290 LDA $0286
300 STA ($FB),Y
310 INY
320 BEQ $9F4F
330 CPY #$E8
340 BNE $9F3D
```

```
350 LDX $FC
360 CPX #$DB
370 BNE $9F3D
380 JMP $EA31
390 INC $FC
400 JMP $9F3D

READY.
```

Még két tipp a képernyő használatához

A kialakított képernyőt tárolhatjuk (hardcopy-val felvehetjük lemezegységre vagy kazettás egységre). Ehhez a következőket kell tennünk:

a) jegyezzük meg a 45–46-os címek tartalmát;
b) írjuk be a

POKE 43,0: POKE 44,X

utasítást. Az X értéke a tárolandó képernyőtől függ. X=4 esetén az első, X=8 esetén a második képernyőt tároljuk;
c) írjuk be a

POKE 45,0: POKE 46,X+4

utasítást;
d) írjuk be a SAVE "(név)",8 parancsot;
e) végül állítsuk vissza a BASIC terület kezdőcímének mutatóit a

POKE 43,1: POKE 44,12: POKE 3072,0

POKE 45,...: POKE 46,... (a megjegyzett értékek)

utasításokkal.

2.17 Futó felirat gépi kódban

A játékokról szóló fejezetben találunk egy futó feliratokat előállító rövid BASIC programot. Most ugyanezt mutatjuk meg gépi kódban.
A BASIC rutinhoz képest annyi a különbség, hogy a gépi kódú program az IRQ-t használja, fut parancs-üzemmódban és BASIC programokban is.
A programban saját szöveget adhatunk meg, amely a képernyő legfelső sorában jelenik meg. A szöveghossz nem haladhatja meg a 40 karaktert, és csak betűket tartalmazhat.
A grafikus jeleket nem kezeli a program. Ha mégis ragaszkodunk a grafikus

jelekhez, ezeket POKE utasítással kell a megfelelő címeken elhelyezni. A szöveg a 40448-as címen kezdődik. A szöveg színét szabadon választhatjuk a következő módon:

POKE 40760,(szín)

A futó felirat sebességét az alábbiak szerint kell megadni:

POKE 40783,(sebesség)

Alapállapotban ezen a címen 5 található. A sebességet és a színt a programban is megváltoztathatjuk. A szín a 260-as DATA sor második pozícióján, míg a sebesség a 280-as sor hetedik pozícióján található. Javításkor ne feledjük az ellenőrző összeget is átírni a 60-as sorban!

A BASIC betöltőprogram:

```
0 REM  **** P29. ****
1 :
2 :
5 REM  FUTO FELIRAT
10 FOR I=40704 TO 40789
20 READ A
30 POKE I,A
40 S=S+A
50 NEXT I
60 IF S<>10107 THEN PRINT"HIBA A DATA SORBAN..!!":END
70 PRINT"OK..!!"
80 PRINT:POKE 19,1
90 INPUT"SZOVEG : ";A$
100 POKE 19,0:PRINT
110 FOR I=1 TO LEN(A$)
120 POKE 40448+I-1,ASC(MID$(A$,I,1))AND NOT 64
130 NEXT I
140 FOR I=LEN(A$) TO 40
150 POKE 40448+I,32
160 NEXT I
170 SYS 40704
180 INPUT"UJ SZOVEG (I/N) ";A$
190 IF A$="I" THEN 80
200 DATA120,169, 28,141, 20,  3,169,159,141
210 DATA 21,  3, 88,133, 56,162,  0,189,  0,158
220 DATA157, 40,158,232,224, 40,144,245
230 DATA 96,206,255,159,208, 50,162,  0,173
240 DATA 50,159,105,  1,201, 40,144,  2,169   0
250 DATA141, 50,159,189, 26,158,157,  0,  4
260 DATA169,  1,157,  0,216,232,224, 40,144
270 DATA240,174, 50,159,232,224, 40,144,  2
280 DATA162,  0,142, 50,159,162,  5,142,255
290 DATA159, 76, 49,234

READY.
```

Ha rutinunkat egy programmal együtt akarjuk futtatni, akkor először a rutint töltsük be és futtassuk, majd ezután töltsük be a programot. Ekkor a rutin BASIC betöltője felülíródik, de a rutin már össze van kötve az IRQ-val.

Az érdeklődőknek a gépi kódú program listája:

```
0 REM  **** P30. ****
1 :
2 :
3 SY549152
4 .OPT OO,P1
5 *= $9F00
10 SEI
20 LDA #$1C
30 STA $0314
40 LDA #$9F
50 STA $0315
60 CLI
70 STA $38
80 LDX #$00
90 LDA $9E00,X
100 STA $9F28,X
110 INX
120 CPX #$28
130 BCC $9F10
140 RTS
150 DEC $9FFF
160 BNE $9F53
170 LDX #$00
180 LDA $9F32
190 ADC #$01
200 CMP #$28
210 BCC $9F2E
220 LDA #$00
230 STA $9F32
240 LDA $9E18
250 STA $0400,X
260 LDA #$01
270 STA $D800,X
280 INX
290 CPX #$28
300 BCC $9F31
310 LDX $9F32
320 INX
330 CPX #$28
340 BCC $9F4B
350 LDX #$00
360 STX $9F32
370 LDX #$08
380 STX $9FFF
390 JMP $EA31

READY.
```

Rutinunk az *Így kell az IRQ-t programozni* c. fejezetben leírtak szerint van felépítve. További részleteket ebben a fejezetben találunk.

2.18 A STOP funkció

Ez a rutin is az ún. megszakítási rendszerrel (IRQ) dolgozik. Az F1 billentyű lenyomásával megállíthatjuk, majd az F3 billentyűvel tovább futtathatjuk a programot. A rutint parancs-üzemmódban használhatjuk a listázás megszakítására, de BASIC programok, sőt néhány gépi kódú játék futtatása során is segítségünkre lehet. Ha valami váratlan esemény történik, pl. cseng a telefon, akkor a rutinnal megszakíthatjuk a munkát. A rutin legelőnyösebben a listázás megállítására és folytatására használható. A rutinnal kényelmesebben és egyszerűbben tudjuk átnézni a hosszabb programokat. Megtakaríthatjuk vele a nehezen kezelhető BREAK használatát és a LIST parancs többszöri begépelését.
Végül lássuk a BASIC betöltőprogramot:

```
0 REM  **** P31. ****
1 :
2 :
5 REM STOP FUNKCIO
10 FOR I=40704 TO 40752
20 READ H
30 POKE I,H
40 S=S+H
50 NEXT I
60 IF S<>5727 THEN PRINT"HIBA H DATA SORBAN..!!":END
70 PRINT"OK..!!"
80 SYS 40704
90 DATA 120,169, 15,141, 20,  3,169,159,141
100 DATA 21,  3,133, 56, 88, 96, 72,165,203
110 DATA201,  4,240,  4,104, 76, 49,234,169
120 DATA159, 72,169, 41, 72,  8, 72,138, 72,152
130 DATA 72, 76, 49,234,165,203,201,  5,208
140 DATA235,240,229
```

A gépi kódú program listája:

```
0 REM  **** P 32. ****
1 :
2 :
3 SYS49152
4 .OPT OO,P1
5 *= $9F00
10 SEI
20 LDA #$0F
30 STA $0314
40 LDA #$9F
50 STA $0315
60 STA $38
```

```
70 CLI
80 RTS
90 PHA
100 LDA $CB
110 CMP #04
120 BEQ $9F1A
130 PLA
140 JMP $EA31
150 LDA #9F
160 PHA
170 LDA #29
180 PHA
190 PHP
200 PHA
210 TXA
220 PHA
230 TYA
240 PHA
250 JMP $EA31
260 LDA $CB
270 CMP #05
280 BNE $9F1A
290 BEQ $9F16

READY.
```

Ha az F1 és F3 funkcióbillentyűk más célra foglaltak, akkor a rutint nagyon egyszerűen átírhatjuk másik két billentyűre. Csak a $9F12 és a $9F2B címekre kell a megfelelő értékeket beírni.
A gépi kódú monitorral nem rendelkezőknek ez azt jelenti, hogy a BASIC betöltőprogram 110-es sorában a 4-es, a 130-as sorában az 5-ös értéket kell megváltoztatni. A megfelelő kódok a függelékben levő kódtáblázatból kikereshetők.

2.19 Véletlen? Az RND közelebbről

Ebben a fejezetben a véletlenszám generálásról lesz szó. Lehet, hogy meglepően hangzik (vagy talán mégsem?), de a számítógép által generált szám egyáltalán nem véletlenszám, hanem egy bonyolult számítási sorozat végeredménye. A matematikai logika alapján dolgozó számítógép csak ilyet tud előállítani.
A gép a véletlenszámot egy komplikált algoritmus többszöri végrehajtásával állítja elő. Ez az algoritmus a következő

1) az utolsó RND értéket (ami a 139–143 címeken található) betölti a lebegőpontos akkumulátorba (FAC);
2) a FAC ezt a számot 11879546-tal (a $E08D–$E091 tartományban tárolódik) megszorozza;

3) ehhez a számhoz hozzáad 3.92767774E–08-at (a $E092–$E096 címeken van);
4) ezután következik a tártartalmak cseréje

   $65 cím tartalmának cseréje $62 tartalmával,
   $63 cím tartalmának cseréje $64 tartalmával,
   $66 címre 0-t ír,
   $61 címre az $80 értéket írja;
5) végezetül a FAC balra tömöríti, majd kerekíti a kapott eredményt, és ismét tárolja.

Rövid magyarázat a konstansokhoz

Ha a konstansok tárterületét egyszer megnézzük, a következő hexadecimális számokat találjuk ott:

$E08D 98 35 44 7A 00
$E092 68 28 B1 46 00

Aki nem ismeri közelebbről a FAC működését, nem valószínű, hogy ezekből a hexadecimális számokból elő tudja állítani a konstansokat.

A problémát a következő rövid program megoldja:

```
0 REM  **** P33.  ****
1 :
2 :
10 :      LDA #$LB
20 :      LDY #$HB
30 :      JSR $BBA2     ;SZAM A FAC-BA
40 :      JSR $BDDD     ;FAC ASCII-BEN ES $0100-TOL TAROLVA
50 :      LDA #$00      ;A FUZER  ALSO    BYTE-JA
60 :      LDY #$01      ;A FUZER FELSO BYTE-JA
70 :      JSR $AB1E     ;A FUZER KIIRASA
80 :      RTS

READY.
```

LB és HB tartalmazza a FAC-ba kerülő 5 byte kezdőcímét. Ha a FAC-t kiolvassuk, a $0061-es címet találjuk benne.

A BASIC betöltőprogram:

```
0 REM  **** P34.  ****
1 :
2 :
10 FOR X=0 TO 17: READ A:POKE 828+X,A: NEXT X
20 DATA169,141,160,224,32,162,187
30 DATA32,221,189,169,0,160,1
40 DATA32,30,171,96
```

A rutin SYS 828-cal indítható (a kazettapufferben van). LB és HB helyére mindig a megfelelő értéket kell írni. Ezeket az értékeket külön POKE utasításokkal is megadhatjuk:

```
POKE 829,LB: POKE 831,HB
```

Lássunk néhány példát az LB–HB értékpárokra:

LB	HB	Érték
141	224	1. konstans
146	224	2. konstans
139	0	az utolsó véletlenszám
97	0	FAC

A rutin gépi kódban is használható. Az RND(1) gépi kódú megfelelője a $E0BE (57534) címen kezdődik. Az RND(–1) és RND(0) rutinok kezdőcímeit nem ismerjük, de mindenki nyugodtan próbálkozhat velük a $E097 és $E0EF címek között. Gépi kódban is előállíthatunk véletlenszámokat a számlálók segítségével. Pl. az óra számláló a 160–162 címen, vagy sorszámláló az 53266-os címen tartalmazza, annak a sornak a sorszámát, amelyikre az adott pillanatban a képernyőre ír az elektronsugár.

Az RND nemcsak játékprogramokhoz használható. Jól alkalmazható olyan file-ok esetén, melyeket nem konkrét adatokkal kell megtölteni. Aki az előbbieket gondosan elolvasta, tudja, hogy a véletlenszámot a számítógép a 139–143-as címeken tárolja. Mivel a gép a következő véletlenszámot az előző alapján számítja ki, a rutinnak egy állandó kezdőértéket adhatunk. Íly módon elértük, hogy a rutin minden egyes futásakor ugyanazt a számsorozatot állítja elő. Ezeket az értékeket újrafuttatáskor ismét felhasználhatjuk.
Ezt mutatja be a következő program:

```
0 REM  **** P35.  ****
1 :
2 :
10 GOSUB 1000
20 PRINT CHR$(147): X=20480
30 POKE 204,0
40 GETA$: IFA$=""THEN 40
50 IFA$= CHR$(133) THEN 100
60 IF ASC(A$)<32 OR ASC(A$) >95 THEN 40
70 IF PEEK(207) THEN 70
80 POKE 204,1: PRINTA$;
90 FOR YY=0 TO 3: Y=Y+INT(RND(1)*40): NEXT :Y=Y+ASC(A$)
95 POKE X,Y:X=X+1: Y=0: GOTO 30
100 POKE X,0: GOSUB 1000
110 PRINT CHR$(147): X=20480
120 Z=PEEK(X): IF Z=0 THEN END
130 FOR YY= 0 TO 3: Z=Z-INT(RND(1)*40): NEXTYY
```

```
140 PRINT CHR$(Z);: X=X+1: Z=0: GOTO 120
1000 PRINT CHR$(147): FORX=0 TO 4
1010 INPUT A: IF A<0 OR A>255 THEN 1010
1020 POKE 139+X,A
1030 NEXT X
1040 RETURN

READY.
```

A program indítása után meg kell adni öt értéket. Megadhatunk egy tetszőleges szöveget is. Ha készen vagyunk, akkor nyomjuk le az F1 billentyűt.
Ezután következik a második futtatás. Adjuk meg még egyszer az előbbi öt számot, és a számítógép ugyanazt a szöveget fogja kiírni.
Ha a számok nem egyeznek meg, akkor a szöveg megcsonkul. Ez a program még nem ad abszolút biztos eredményt, mert néhány apró eltérést nem vettünk figyelembe (a 90-es és 130-as sorokban levő INT függvény miatt). Így érthetőbbé válik, hogyan titkosíthatunk adathalmazokat.

a) Állítsuk be az RND értéket egy meghatározott értékre (a felhasználó kódja) POKE utasítással a 139–143 címekre.
b) Titkosítsuk az adatokat az RND függvénnyel. A titkosítás módja a titkunk marad.
c) Az adatok beolvasásakor ismét meg kell adni a kódot, amely alapján az RND ismét a meghatározott értékeket állítja elő.
d) Az RND az adatokat dekódolja. Hibás kód esetén hibás eredményeket kapunk.

Ezen egyszerű trükkel lehetővé válik, hogy számokat és szövegeket titkosítsunk, és így csak a beavatott felhasználó tudjon az adatokhoz hozzáférni. Elméletileg persze a felhasználói kódot nem ismerők is megfejthetik a titkosítást úgy, hogy minden lehetséges felhasználói kódot kipróbálnak, de ez a gyakorlatban nagyon hosszú időt venne igénybe.

Még egy tipp

Ha az illegális felhasználók dolgát még jobban meg akarjuk nehezíteni, akkor az adatok titkosítására szánt Y = Y + RND(1) parancsot a felhasználó vigye be a gépbe. Mivel így a lehetőségek száma a kóddal ellentétben határtalan, lehetetlen az adatokhoz való illegális hozzáférés.

Összefoglalva

A pszeudo-véletlenszámok előállítására a BASIC-ben az RND utasítást használjuk. Ennek az utasításnak argumentuma van, ami jelentőséggel bír. A pozitív, a negatív és a nulla argumentum sokban különbözik egymástól. Pozitív argumentum esetén a pszeudo-véletlenszám az előző véletlenszámból képző-

dik. A negatív argumentum megváltoztatja a véletlenszám bázis értékét (139–143-as címek) a negatív argumentum függvényében.
Az RND(X) függvénnyel úgy tudunk valódi véletlenszámot generálni, hogy a bázist mindig átírjuk. Ez a következő programsorral valósítható meg:

```
10 X=RND(-TI): X=INT(RND(1) * .,.
```

2.20 Módosított INPUT

A C 64-es beviteli utasításai nagyon szegényesek: alapállapotban csak a GET és INPUT utasítások állnak rendelkezésünkre. Ez a két utasítás sok esetben nem elégíti ki igényeinket: a GET utasításnál hiányzik a kurzor, az input utasításnál pedig zavarólag hat az állandóan megjelenő kérdőjel. Néhány írásjelet nem is fogad el az INPUT. (Ezt már bizonyára mindnyájan tapasztaltuk.)
A következőkben az adatbevitelt megkönnyítő olyan tippekkel foglalkozunk, amelyek az előbb említett hiányosságokat kívánják kiküszöbölni.
Nézzük először a kérdőjelet, amely akarva akaratlanul minden INPUT utasításnál megjelenik, figyelmeztetve bennünket arra, hogy adatbevitel következik.
Az INPUT kérdőjelének megszüntetésére több lehetőség van:

```
0 REM  **** P36.  ****
1 :
2 :
10 POKE 19,1: INPUT"NYOMJA LE A RETURN-T !";A$:PRINT:POKE 19,0

READY.
```

vagy

```
0 REM  **** P37.  ****
1 :
2 :
10 OPEN 1,0
20 INPUT#1,A$
30 CLOSE 1

READY.
```

Ez utóbbi a billentyűzetet perifériaként nyitja meg.
Nehezebb dolgunk van akkor, ha a normál szöveg mellett pl. írásjeleket stb. akarunk bevinni, melyekre egy szövegfeldolgozó programban van szükségünk. E feladat megoldására készítettük a következő, nem túl hosszú BASIC rutint. A rutin elég gyors ahhoz, hogy egy szövegfeldolgozó programban nyugodtan használhassuk.
Az INPUT ismert lehetőségein kívül a rutin használatával a következőket is megtehetjük:

– egy füzérhez tetszőleges számú elemet rendelhetünk, csak a megfelelő DIM utasítást kell megváltoztatni;
– vezérlőjeleket is megadhatunk, melyeket a program kiíráskor figyelembe is vesz;
– tetszőleges szöveget készíthetünk. A lekérdezés eredményét a KE$ változóban tároljuk. VAL utasítással még számokat is megadhatunk.

Végül a BASIC rutin listája:

```
0 REM  **** P38. ****
1 :
2 :
10 REM SZIMULÁLT INPUT
20 KF$="MIT KELL BEVINNI ??":GOSUB60000:REM A RUTIN HÍVÁSA
25 PRINTKE$
30 END
40 :
50 :
60 :
60000 REM SZIMULÁLT INPUT (RUTIN)
60010 DIM KE$(1000):PRINTKF$;" ";
60050 POKE204,0
60051 POKE212,1:GETKE$:IFKE$=""THEN60051
60052 POKE207,0
60055 IFKE$=CHR$(13)THENPOKE204,1:PRINT" ":KE$="":GOTO60100:REM
      RETURN-BILLENTYŰ
60056 IFKE$=CHR$(20)THENGOTO60120
60060 PRINTKE$;
60070 KE$(KE)=KE$:KE=KE+1
60080 GOTO60050
60100 FORSL=0TOKE-1
60101 KE$=KE$+KE$(SL)
60102 NEXT:RETURN
60120 IFKE=0THENGOTO60050
60125 POKE212,0:POKE207,0:PRINTCHR$(157)CHR$(32)CHR$(32)
      CHR$(157)CHR$(157);
60130 KE=KE-1:KE$(KE)="":GOTO60050

READY.
```

A felhasznált fontosabb címek:

212 Idézőjel-kapcsoló
204 A kurzor ki- és bekapcsolása

A fontosabb változók:

KE$ A lekérdezés eredménye
KF$ Kérdésfüzér (a rutin hívásakor megjelenik a képernyőn)

Ha az INPUT utasításban vesszőt is akarunk használni, akkor a következő BASIC sort írjuk be:

10 POKE 198,1: POKE 631,34: INPUT A$

Idézőjelek között a számítógép elfogadja a vesszőt, mert itt nem különleges jelként, hanem egyszerű beviteli jelként értelmezi. A billentyűzetpufferben (631–640) egy idézőjelet tárolunk (kódja: 34). A 198-as címen ezt tudatjuk a számítógéppel.
Így az INPUT utasítás első karaktere egy idézőjel lesz, amely után már használhatjuk a vesszőt is. Ezt az idézőjelet a füzér később nem veszi figyelembe, és a LEN utasítást sem befolyásolja.

2.21 Lemezegységtrükkök

Hardver szempontból a C 64-eshez 8 lemezegység csatlakoztatható. A lemezegységek a 8–15-ös csatornákat használhatják. Gyárilag azonban minden lemezegység 8-as egységszámúra van állítva, így ezt szükség esetén meg kell változtatni.
Ehhez nyújt segítséget az alábbi program:

```
0 REM  **** P39.  ****
1 :
2 :
10 INPUT"REGI EGYSEGSZAM";RE
20 INPUT"UJ EGYSEGSZAM";UE
30 OPEN1,RE,15
40 PRINT#1,"M-W"CHR$(119)CHR$(0)CHR$(2)CHR$(UE+32)CHR$(UE+64)
50 CLOSE1

READY.
```

Ha tehát több lemezegységgel akarunk dolgozni, akkor a következőket kell tennünk:
Csatlakoztassuk az első lemezegységet, és adjunk neki egy nyolctól különböző egységszámot. Ezután csatlakoztassuk a második lemezegységet.
Mivel ez 8-as egységszámú, a két lemezegység most már külön-külön is címezhető. Ha szükséges, címezzük át ezt a lemezegységet is stb.
Több lemezegység használata esetén az alábbi program célravezetőbb:

```
0 REM  **** P40.  ****
1 :
2 :
10 PRINT"HANY LEMEZEGYSEGGEL AKAR DOLGOZNI?"
20 INPUTHY
30 IFHY<1ORHY>8THEN20
40 RE=8:FORY=1TOHY
50 PRINT"ADJA MEG A "Y". LEMEZEGYSEG SZAMAT"
60 GETA$:IFA$=""THEN60
70 OPEN1,RE,15
80 PRINT#1,"M-W"CHR$(119)CHR$(0)CHR$(2)CHR$(39+Y)CHR$(71+Y)
90 CLOSE1:PRINT"A LEMEZEGYSEG SZAMA"RE
100 RE=RE+1:NEXTY

READY.
```

Ennek a módszernek azonban van egy nagy hátránya: a lemezegység kikapcsolása után az egységszáma ismét 8 lesz.
A lemezegység egységszáma hardver megoldással is megváltoztatható. Ezzel a módszerrel azonban csak 8–11 között választhatjuk meg az egységszámokat, ezért több lemezegység esetén a szoftver eljárást kell alkalmazni. A lemezegységeket először természetesen inicializálni kell. Ennek a célnak egy megfelelő programmal és autostarttal ellátott EPROM felelne meg. De az itt ismertetett második program is megfelel a célnak, csak ezt minden felhasználás előtt be kell tölteni.
Most pedig lássuk a hardver megoldást! Felhívjuk a figyelmet arra, hogy mivel ehhez a készüléket fel kell nyitni, a garanciája megszűnik. De nem kell félni, előírás szerinti munka esetén nem lesz semmi baj.
Először áramtalanítsuk a gépet! Oldjuk a készülék alján található négy csavart, majd emeljük le a fedelét. E művelet során legyünk óvatosak, nehogy az író-olvasó fej megsérüljön. A fedél eltávolítása után egy nyomtatott áramköri lapot látunk magunk előtt. Ennek a közepén több nagyobb IC-t találunk.
Keressük meg a 6502-est, amely a CPU, vagyis a központi egység. Közvetlenül e mellett található a 6522-es I/0 processzor. Ettől induljunk el a lemezegység előlapja felé. Először egy elektrolitkondenzátort láthatunk, a jele C 64.
Rögtön emellett találunk két forrasztást. Ezek félkör alakúak, és egy keskeny híd köti össze őket. Attól függően, hogy melyik félkört vágjuk át, a következő egységszámokat kapjuk:

Átvágott félkör	Egységszám
Egyik sem	8
1 (bal)	9
2 (jobb)	10
1 és 2	11

Ez után a művelet után a lemezegység már mindig pl. a 9-es egységszámon jelentkezik be, a kikapcsolás után is.

Ha már egyszer felnyitottuk a lemezegységünket, nézzünk meg egy másik dolgot is!
Normális esetben a lemezt mindkét oldalával betehetjük a lemezegységbe, ha rajta egy szimmetrikus másik ablakot vágunk. Ez a valóságban másképpen néz ki. A nyomtatott áramköri lap (nyáklap) hosszanti oldala mentén több csatlakozóaljzat helyezkedik el, amelyek közül a legnagyobb a P6 jelű. A csatlakozón egy narancs és egy ibolyakék színű vezeték található. E két vezetéket kell egy kapcsolóval összekötnünk, és a lemezegység hátán kívülre vezetnünk. Így a lemezeket kivágás nélkül is lehet két oldalasan használni. A készülék ui. a kivágásról átvilágítással győződik meg, amelyet mi a kapcsolóval egyszerűen áthidalunk. Vigyázzunk, mert így leragasztott lemezre is írhatunk véletlenül!
Most pedig bemutatunk egy rövid rutint, melyet minden lemezegységgel dolgozó programunkba beépíthetünk. Ha egy programban file-okat akarunk lemezen

tárolni, és a lemezegységet véletlenül nem kapcsoltuk be, akkor a program DEVISE NOT PRESENT ERROR hibajelzéssel leáll. Ennél sokkal jobb lenne, ha ilyen esetben a számítógép a felhasználót a lemezegység bekapcsolására figyelmeztetné.
A következő rutin megvizsgálja, hogy a lemezegység be van-e kapcsolva. Ha igen, akkor a $FF(255) regiszter 0-ra állítódik, ellenkező esetben az értéke 5 lesz. Programunkban ezt a címet kell lekérdeznünk, és a válasznak megfelelően eljárnunk.
A rutint a kazettapufferben helyeztük el, mivel ez a lemezegység üzemét nem zavarja.

```
0 REM **** P41. ****
1 :
2 :
3 OPEN1,8,1,"@0:P 41"
5 SYS36864
6 .OPT O1,P1
7 *=$033C
10 LDA #$01      ;A FILE-NEV HOSSZA
20 LDX #$D0      ;CIM ALSO  BYTE
30 LDY #$FF      ;CIM FELSO BYTE
40 JSR $FFBD     ;A FILE-NEV KIIRASA
50 LDA #$01      ;LOGIKAI FILE-SZAM
60 LDX #$08      ;EGYSEGSZAM
70 LDY #$00      ;MASODLAGOS CIM
80 JSR $FFBA     ;A FILE-PARAMETER KIIRASA
90 JSR $FFC0     ;A FILE MEGNYITASA
100 BCS $0355     ;A KESZULEK NEM VALASZOL
110 LDA #$00      ;A KESZULEK MEGVAN
120 STA $FF       ;A KAPCSOLO TAROLASA
130 LDA #$01      ;LOGIKAI FILE-SZAM
140 JSR $FFC3     ;A FILE LEZARASA (CLOSE)
150 RTS
```

A file-névnek a $FFD0 címet választottuk, mivel ezen a címen a $-jel ASCII-kódja, a 36 áll. Választhattuk volna a $FFE5 címet is. Ekkor a file-név * lett volna. Ez azonban nem célszerű, mert így nem kereshetünk bármilyen nevet, és FILE NOT FOUND hibajelzést kapunk.

A BASIC betöltőprogram:

```
0 REM  **** P42. ****
1 :
2 :
10 FORY=0TO32:READA:POKE828+Y,A:NEXTY
20 DATA169,1,162,208,160,255,32,189,255
30 DATA169,1,162,8,160,0,32,186,255
40 DATA32,192,255,176,2,169,0,133,255,169,1,32,195,255,96

READY.
```

A rutin SYS 828-cal hívható.

Lássunk egy rövid rutint, amely a hibacsatornát olvassa ki.
A hibacsatorna kiolvasása c. fejezetben láthattuk hogy ez BASIC-ből hogyan zajlik le. Most a gépi kódú lekérdezést mutatjuk be.

Aki a KERNAL rutint ismeri (l. a *KERNAL* c. fejezetben), tudja, hogy egy speciális OPEN rutinnal rendelkezik. Az INPUT #1 utasítással még nehezebb a helyzet. A KERNAL-ban van egy rutin, amellyel az IEC-buszról lehívhatunk egy jelet. Ha ez utóbbi rutint jól megnézzük, valószínűleg feltűnik, hogy nem használja az OPEN rutint.
Ezt a lehetőséget fogjuk mi is kihasználni kis rutinunkban, amely a kazettapufferben helyezkedik el.

```
0 REM  **** P43.  ****
1 :
2 :
3 OPEN1,8,1,"@0:P 43"
5 SYS36864
6 .OPT O1,P1
7 *= $033C
10 LDA #$08          ;A LEMEZEGYSEG SZAMA
20 STA $BA           ;TAROLAS
30 JSR $FFB4         ;TALK JEL KULDESE (A LEMEZEGYSEG KULDI)
40 LDA #$6F          ;MASODLAGOS CIM
50 STA $B9           ;TAROLAS
60 JSR $FF96         ;A MASODLAGOS CIM KIKULDESE A TALK UTAN
70 JSR $FFA5         ;EGY KARAKTER LEHIVASA A LEMEZEGYSEGRUL
80 JSR $FFD2         ;A KARAKTER KIIRASA
90 CMP #$0D          ;RETURN
100 BNE $034A        ;NEM,TEHAT TOVABB
110 JSR $FFAB        ;UNTALK JEL KULDESE (HIBAS VEGE)
120 RTS               ;VISSZA

READY.
```

A kedves Olvasó most valószínűleg azt kérdezi, hogy a másodlagos cím értékét miért nem 15-re állitottuk be, mint ahogy ezt a BASIC-ben megszoktuk. Normális esetben az OPEN rutin a másodlagos cím értékéhez éppen $60-at ad. Mivel mi az OPEN rutint nem használjuk, ezt az összeadást előzetesen elvégeztük (l. a 343 LDA #$6F utasítást).
A BASIC betöltőprogram:

```
0 REM  **** P44.  ****
1 :
2 :
10 FORX=0TO27:READA:POKE828+X,A:NEXTX
20 DATA169,8,133,186,32,180,255,169,111,133,185,32,150,255
30 DATA32,165,255,32,210,255,201,13,208,246,32,171,255,96

READY.
```

A rutin SYS 828-cal hívható.

Könyvünk nem programgyűjtemény, ezért nem tartalmaz sok olyan rutint, amelyek a lemezegység-üzemeltetők számára érdekes lehet. Aki az eddig ismertetett rutinok alapján önálló rutin megírását is célul tűzte maga elé, próbálja meg először pl. az előbbi rutint az OPEN utasítás használatával átírni. Érdekes lehet egy olyan rutin megírása is, amely a lemez tartalomjegyzékét úgy olvassa be, hogy az éppen futó programot nem zavarja meg.

Végül néhány tipp

Ha egy meghatározott programot keresünk, de nem tudjuk, hogy melyik lemezen van, akkor a tartalomjegyzék egy részének betöltéséhez a következő utasítást használhatjuk:

```
LOAD "$0: (név)",8
```

Ha a keresett program nincs a lemezen, akkor csak a lemez neve fog a képernyőn megjelenni.
Az utasításban a * jokert is használhatjuk:

```
LOAD "$0: PAS*",8
```

Hatására a lemezen található összes PAS-sal kezdődő programnév kilistázódik.
Bizonyára mindenki tudja, hogy a SHIFT/RUN-STOP billentyűkkel a kiválasztott program szalagról betölthető, majd automatikusan indítható. Azt viszont már kevesebben ismerik, hogy ez a lemezekre is igaz:

```
LOAD "GRAFIK",8: (SHIFT/RUN-STOP)
```

A normál LOAD megadása után a RETURN helyett a SHIFT/RUN-STOP billentyűket kell lenyomni. Megjelenik a LOAD felirat, és a megnevezett program betöltődik, majd automatikusan indul. A * jelet itt is használhatjuk:

```
LOAD "*",8: (SHIFT/RUN-STOP)
```

Hatására a lemez első prgramja betöltődik, és automatikusan indul.

A tartalomjegyzék betöltésénél a megjelentetni kívánt file-típus is megadható a LOAD utasításban:

```
LOAD "$*=T",8
```

A T helyére a következő paraméterek írhatók:

T	File-típus
P	program
S	soros
R	relatív
H	használói

Ez a lemezegységet kezelő összes parancsra igaz. Ha tehát pl. az összes program file-t le akarjuk törölni a lemezről úgy, hogy a többi file-típus sértetlen maradjon, akkor a következőket kell megadni:

```
OPEN 15,8,15
PRINT # 15, "S:*=P"
```

Hatására az összes program-file törlődik.

3. FEJEZET
SZOFTVERVÉDELEM

3.1 A LIST parancs kezelése

A LIST parancsa a BASIC programozók egyik leggyakrabban használt parancsa, amelynek segítségével szinte „átvilágíthatjuk" programunkat.
A legtöbb programszerző azonban nem nagyon örül ennek, mert utasításai egyszerűen leolvashatók, így a program könnyen megváltoztatható.

Ahhoz, hogy egy program kilistázását hatékonyan meg tudjuk akadályozni, előbb ismerkedjünk meg a LIST parancs működésével.

3.1.1 Listázás sorszámok nélkül

A 22-es cím megváltoztatásával (az ideiglenes füzértábla mutatója) a következőket tapasztaljuk: a lista sorszámok nélkül jelenik meg. Az utasítás a következő:

```
POKE 22,35
```

Az alapállapotot a

```
POKE 22,25
```

utasítással állíthatjuk vissza.
De legyünk óvatosak: az ilyen közbeavatkozásoknak rendszerint mellékhatásai is vannak, amelyekkel tisztában kell lennünk.
Amíg a 22-es címen 35-ös érték van, addig a PRINT utasítások nem jelennek meg. A hibajelzések az alapállapotban térnek vissza.

A programot sorszám nélkül a nyomtatón is megjeleníthetjük. Ügyeljünk azonban arra, hogy a listázáskor az utolsó sor nem jelenik meg. Ez a sor csak a

```
PRINT #1 (ill. a megfelelő logikai file-szám)
```

utasítás hatására íródik ki.

Természetesen a 22-es címre más értékeket is írhatunk. Hatásaikat a következőkben foglaltuk össze:

POKE 22,25 alapállapot
POKE 22,32 a sorszámok nem olvashatók
POKE 22,33 a sorszámok helyén "!" áll.
POKE 22,34 ? FORMULA TOO COMPLEX ERROR, majd újból alapállapot
POKE 22,35 a sorszámok teljesen eltűnnek

3.1.2 Listázás elleni védelem – a LIST parancs kikapcsolása

Néha az is szükséges, hogy a teljes listát letiltsuk. Erre több lehetőség kínálkozik.

1) A következő REM sor arról gondoskodik, hogy az őt követő egyetlen programsort se lehessen kilistázni. Helyette a SYNTAX ERROR hibajelzés jelenik meg.

```
0 REM  **** P45.  ****
1 :
2 :
10 PRINT "1. VEDELMI VARIACIO"
20 REM (SHIFT & L)
30 PRINT "TOBBE NEM LISTAZHATO !"
```

A védekezésnek ez a módja azonban könnyen átlátható, és eltávolítható. (Ha viszont kombináljuk más védelemmel, pl. a sorszámozás megváltoztatásával, akkor a REM utasítást nem lehet kitörölni.)
Ha a program minden 5. sora ilyen REM utasítást tartalmaz, akkor ezen sorok eltávolítása elég körülményes lesz.

2) Ez a módszer első pillantásra nem is tűnik védelemnek.
Vigyük be a következő programot:

```
0 REM  **** P46.  ****
1 :
2 :
10 REM 2. VEDELMI VARIACIO
20 PRINT"LEFEDVE": REM""

READY.
```

Menjünk a kurzorral a 20-as sorban a REM mögé, majd annyiszor nyomjuk le az INST billentyűt, ahány betű van a sorban (most 21-szer!).
Végül ugyanennyiszer nyomjuk le a DEL billentyút.
A 20-as sorban a REM után 21 inverz T fog megjelenni.
Minden egyes T a sor egy jelét törli. Listázáskor a védett sor kiíródik, de rögtön le is törlődik az inverz T hatására.
Ha a sornak csak egy részét akarjuk kitörölni, akkor kevesebb T-t használunk a REM után.

Ezt a megoldást azonban csak rövid, max. 10 jelet tartalmazó sorokhoz ajánljuk. Több karakter esetén ui. észrevehetővé válik a képernyőn a felvillanó sor. (A HELP ± segédprogram # L parancsával viszont minden gond nélkül listázható.)

Bizonyára most azt gondolja a kedves Olvasó, hogy ezek a védekezési módok olyanok, mint a tavalyi hó, közismertek és unalmasak.
Úgy véljük, hogy a bemutatott két megoldás alapelvének ismerete nagyon hasznos.

3) Ezután a magunk módján teljesebb, más elven alapuló védekezési módokat mutatunk be.

A következő megoldás bonyolultnak tűnik, de ha figyelmesebben megvizsgáljuk, akkor belátható, hogy elég hatékony védelmet nyújt a „programkalózok" ellen.
A programot fűzzük a saját programunk után:

```
0 REM  **** P47.  ****
1 :
2 :
62000 FOR A=PEEK(43)+256*PEEK(44)TOPEEK(45)+256*PEEK(46)-3
62010 IFPEEK(A)=58ANDPEEK(A+1)=58ANDPEEK(A+2)=58THENGOSUB62030
62020 NEXT A:END
62030 IFPEEK(A+3)=58ANDPEEK(A+4)=58THENPOKEA,0:A=A+4:RETURN

READY.
```

A védeni kívánt sorokat kezdjük öt kettősponttal. Pl.

```
0 REM  **** P48.  ****
1 :
2 :
45 PRINT "PELDA"

READY.
```

A programsor védelemmel ellátva:

```
0 REM  **** P49.  ****
1 :
2 :
45 :::::PRINT "PELDA"

READY.
```

Jelöljük meg az összes védeni kívánt sort. Ezután indítsuk a programot RUN 62000 paranccsal. A programfutás ideje a védendő sorok számától függ, de általában sokáig tart. Ezért érdemes ezt a védelmi módszert előbb rövid programokon kipróbálni, hogy láthassuk a sebességét. Listázáskor a védett soroknak csak a sorszáma jelenik meg, a tartalma nem. Végül a 62000–62030-as sorokat kitörölhetjük, és csak a védett programot tároljuk a lemezen.

A védekezés elve

Először képezünk egy vektort, amely a teljes BASIC programot tartalmazza (43–44 a BASIC kezdete, 45–46 a program vége).
Ezután keressük az öt kettőspontot (58-as kód). Ha megtaláltuk, akkor az első kettőspont kódját 0-ra változtatjuk, a többi marad változatlanul.
Ha most kiadjuk a LIST parancsot, akkor a rutin egy 0-t talál, ami neki a sor végét jelzi, tehát a listázást abbahagyja.
Mindebből következik, hogy a maradék négy kettőspontot csak félrevezetésnek szántuk.
Ezt a megoldást a sebessége teszi barátságtalanná, ezért a programot gépi kódban is megírtuk. A sebességéről csak annyit, hogy egy 15 kbyte hosszú programot kb. 1 s alatt véd le listázás ellen.

A gépi kódú program BASIC betöltőprogramja:

```
0 REM  **** P50. ****
1 :
2 :
10 REM GEPI KODU PROGRAM TETSZOLEGES BASIC LISTA VEDELMERE
20 REM KEZDOCIM: $8000 (DEC.32768), HOSSZA: 80 BYTE
30 REM A VEDENDO SOROKAT >-JELLEL ES 4 KETTOSPONTTAL KELL KEZDENI
40 REM PL: '10 REM' HELYETT '10 >::::REM'
50 REM A PROGRAM INDITASA: SYS 8*4096
60 :
70 :
80 :
90 READA:REM A PROGRAM BEOLVASASA
100 IFA=-1THENGOTO150:REM A VEGE JEL ELLENORZESE
110 CH=CH+A:REM AZ ELLENORZO OSSZEG SZAMITASA
120 POKE8*4096+I,A:REM A DATAK TAROLASA
130 I=I+1:GOTO70:REM A KOVETKEZO ERTEK BEOLVASASA
150 IFCH<>98391THENPRINT"HIBA A DATA SORBAN..!!":LIST200-
160 PRINT"DATAK OK.  A PROGRAM INDITASA■■■■SYS 8*4096 !":
    END
200 DATA162,120,169,8,160,0,132,34,133,35,177,34,201,177,240,
    9,200,208
210 DATA247,230,35,202,208,242,96,142,80,128,141,81,128,140,
    82,128,165,35
220 DATA133,37,165,34,133,36,162,4,200,240,28,177,36,201,58,
    208,10,202
230 DATA208,244,172,82,128,169,0,145,34,174,80,128,173,81,128,
    172,82,128,76
240 DATA16,128,230,37,76,47,128,-1

READY.
```

A rutin SYS 8*4096-tal hívható.
A védendő sorokat a programban egy > jellel és négy kettősponttal kell megjelölni.

3.2 A BASIC LINK megváltoztatása

A sorok láthatatlanná tételére alkalmas módszer a BASIC LINK megváltoztatása.
De előbb lássuk, mi is az a BASIC LINK?
A kérdés megválaszolásához ismerjük meg néhány BASIC tároló tartalmát és jelentését:

```
2048   0
2049  16 LINK alsó byte
2050   8 LINK felső byte
2051  10 sorszám alsó byte
2052   0 sorszám felső byte
2053 153 a sor tartalmának kezdete
.
.
.
2063   0 sorvég
2064  31 LINK alsó byte (az előző LINK ide mutat)
2065   8 LINK felső byte
2066  20 sorszám alsó byte
2067   0 sorszám felső byte
2068 153 a sor tartalmának kezdete (153 a PRINT kódja)
```

A 2048-as címen 0 áll, ami a BASIC terület kezdetét jelenti. Az e fölötti két címen található a LINK. A LINK alsó és felső byte formájában tartalmazza a következő BASIC sor kezdőcímét.
A következő két byte a pillanatnyi sorszám. Ezt követi a sor tartalma, amely 0-val zárul. Ismét következik egy LINK, amely a következő sorra mutat. Majd jön a sorszám, amelyre az előző LINK mutatott. A program végén az utolsó LINK 0,0.

A LINK tehát lényegében a LIST rutin számára készült. Ha nem akarunk egy sort kilistázni, akkor meg kell változtatni a LINK-et úgy, hogy a következő sorra mutasson.

Ezt a feladatot oldja meg a következő BASIC program:

```
0 REM  **** P51. ****
1 :
2 :
60000 POKE53280,1:POKE53281,1:POKE646,6:PRINT"■";
60001 PRINTTAB(10)"■ LINK VALTOZTATO "
60002 PRINTTAB(5)"■■■■EZ A PROGRAM A BASIC LINKET ":PRINT
60003 PRINTTAB(5)"VALTOZTATJA MEG. A PROGRAM BEKERI":PRINT
60004 PRINTTAB(5)"AZ ELSO ES MASODIK SORSZAMOT.":PRINT
60006 PRINTTAB(5)"AZ OSSZES EBBE AZ INTERVALLUMBA":PRINT
```

```
60007 PRINTTAB(5)"ESU SOR TOBBE NEM LISTAZHATO! ":PRINT
60008 PRINTTAB(5)"MINEL HOSSZABB AZ ■UN■ PROG-":PRINT
60009 PRINTTAB(5)"RAMJA, ANNAL TOVABB TART A FUTAS!":PRINT:PRINT
60010 PRINTTAB(5)"NYOMJON LE EGY BILLENTYUT!"
60011 GETA$: IFA$=""THEN 60011
60012 PRINT"J";
60015 INPUT"AZ ELSO SORSZAM";A:INPUT"A MASODIK SORSZAM";E
60019 PRINT"TALALT SORSZAM :"
60020 Q=PEEK(43)+256*PEEK(44)
60030 ZL=PEEK(Q+2)+256*PEEK(Q+3):IFZL=ATHENH1=Q:H2=Q+1:GOTO60060
60031 IF ZL>ATHENPRINT"■NINCS ILYEN SOR !":END
60035 PRINTTAB(23)"J";ZL
60040 Q=PEEK(Q)+256*PEEK(Q+1):IFQ>=PEEK(45)+256*PEEK(46)-3THENEND
60050 GOTO60030
60060 Q=PEEK(43)+256*PEEK(44)
60070 IFPEEK(Q+2)+256*PEEK(Q+3)=ETHENPOKEH1,PEEK(Q):
      POKEH2,PEEK(Q+1):END
60080 Q=PEEK(Q)+256*PEEK(Q+1):IFQ>=PEEK(45)+256*PEEK(46)-3
      THENEND
60090 GOTO60070

READY.
```

3.3 Sortörlés? SYNTAX ERROR!

A védekezés másik módszere – a BASIC LINK megváltoztatása mellett – a sorszám megváltoztatása. A sorszámok a LINK-hez hasonlóan alsó és felső byte formájában vannak a tárolóban.
Azt tudjuk, hogy alapesetben 63999-nél nagyobb sorszámot adhatunk, különben SYNTAX ERROR hibajelzést kapunk. A következőkben megmutatjuk, hogyan lehet a sorokat 65535-ig sorszámozni. Majd azt is megmutatjuk, hogyan lehet több sort ugyanazzal a sorszámmal ellátni, továbbá hogyan követhet egy alacsonyabb sorszámú sor egy magasabb sorszámút.

Amint már mondtuk, a sorszám alsó és felső byte formájában van a tárolóban. Ebből következik, hogy a legkisebb lehetséges sorszám a 0 (0+256*0), a legnagyobb pedig a 65535 (255+256*255). Egy program futtatásakor a számítógépnek közömbös, hogy a BASIC programban a sorok hogyan vannak számozva. A gép a sorokat a tárban levő sorrendjük alapján dolgozza fel. Éppen ezért minden további nélkül lehetséges, hogy a sorszámokat a programlefutás veszélyeztetése nélkül megváltoztassuk.
Figyelembe kell azonban vennünk, hogy a GOTO, GOSUB, THEN utasításokat hozzá kell igazítani a megváltozott körülményekhez.

Eddig minden szép és jó, de hogyan lehet a kijelölt sorszámok alsó és felső byte-ját megváltoztatni a tárolóban?
Ez a végtelenségig is eltarthat!

A következő segédprogrammal nem fáradságos a sorszámok megváltoztatása, még hosszabb programok esetében sem:

A program listája:

```
0 REM  **** P52. ****
1 :
2 :
60000 Q=PEEK(43)+256*PEEK(44)
60001 PRINT"J":GOSUB60100
60002 Q=PEEK(Q)+256*PEEK(Q+1):IFQ>=PEEK(45)+256*PEEK(46)-3THENEND
60003 GOTO60001
60010 IFPEEK(Q)=0THENGOSUB60100
60020 NEXTQ:END
60100 LB=PEEK(Q+2):HB=PEEK(Q+3)
60105 POKE199,1:PRINT"SORSZAM-VALTOZTATO":PRINT
60110 PRINT"JELELT SORSZAM :";LB+256*HB
60120 PRINT:PRINT"[1] VALTOZTATAS"
60130 PRINT:PRINT"[2] TOVABB"
60140 PRINT:PRINT"[3] VEGE"
60150 GETA$:IFA$=""THEN60150
60160 IFA$="2"THENRETURN
60170 IFA$="3"THENEND
60180 A$="0":PRINT:INPUT"UJ SORSZAM (0-65535)";A$:A=VAL(A$)
60190 HB=INT(A/256):LB=A-(256*HB)
60200 POKEQ+2,LB:POKEQ+3,HB:RETURN

READY.
```

Programmagyarázat

A 60000-es sorban határozzuk meg az első LINK címet. A 60020-as sortól ágazik el a program a rutinba. Itt kérdezzük le az első sorszám alsó és felső byte-ját (LINK cím +2 és +3).
Az így előállított sorszám jelenik meg a képernyőn (60110-es sor).
Választhatunk, hogy 1) megváltoztatjuk vagy 2) tovább megyünk.
A 2) pont a sorszámot változatlanul hagyja, és keresi a következőt. Az 1) pontban lehet a sorszámot megváltoztatni. Megadhatunk egy tetszőleges számot 0 és 65535 között. Az általunk decimálisan megadott sorszámot alsó és felső byte-ra bontjuk (60190-es sor), majd POKE utasítással bevisszük a tárba a megfelelő címre. Végül a 60010-es sorban a régi LINK helyére az új kerül. Ismét jön az elágazás a 60020-as sorra...
Ha a programot nem szakítjuk meg, akkor automatikusan a tárban levő program végéig fut (60010-es sor).

Az általunk létrehozott sornak az eredetivel szemben több előnye van.

Ha a megváltoztatott sorszám 63999 fölött van, akkor felülírással többé nem törölhető.
Ha a megváltoztatott sorszám kisebb, mint az eredeti, akkor az új sorszám

szintén nem törölhető. Ez csak akkor lehetséges, ha a megváltoztatott sor a program első sora.

Végül kitörölhetjük a segédprogramot, és tárolhatjuk tetszés szerint megváltoztatott programunkat. A védelem a programmal tárolódik.

3.4 Mesterséges vezérlőjelek

Most egy nem közismert védekezési eljárást mutatunk be. Mindnyájan ismerjük a természetes vezérlőjeleket. Ezek a kurzormozgató jelek, amelyek idézőjelek között a PRINT utasításban fordulnak elő, vagy a szín-vezérlőjelek, amelyek a CTRL és egy színváltó billentyű egyidejű lenyomásával keletkeznek idézőjel-üzemmódban.
Vannak azonban ún. mesterséges vezérlőjelek is. Vezérlőjelek, mert billentyűlenyomással képezhetők.

Most csak egy mesterséges vezérlőjellel foglalkozunk.
Írjuk le a következő sort:

```
10 REM " "
```

Menjünk a kurzorral a második idézőjelre. Nyomjuk le egyidejűleg a CTRL és a RVS ON (9) billentyűt! Majd egyszerre a SHIFT és az M billentyűt!
A második idézőjel helyén ekkor fekete négyzetben egy világos átlós vonal jelenik meg. Ez a mesterséges vezérlőjel. Mindjárt világossá válik, hogy mi történt. Nyomjuk le egyidejűleg a SHIFT és a Q billentyűket!
Újból megjelenik egy vezérlőjel: a CURSOR UP. Kapcsoljuk ki az inverz üzemmódot a CTRL és a RVS OFF (0) billentyű egyidejű lenyomásával. Végül írjuk be, hogy TESZTSOR. Nyomjuk le a RETURN billentyűt.
A képernyőn a következő sor jelenik meg:

```
0 REM  **** P53.  ****
1 :
2 :
10 REM"■TESZTSOR

READY.
```

Listázzuk ki ezt a sort! Csak a TESZTSOR felirat látható!
A mesterséges vezérlőjel egy RETURN (CR) műveletet hajt végre, de úgy, hogy közben gondoskodik a REM után álló természetes vezérlőjelek utasításainak végrehajtásáról. Majd megjelenik a CURSOR UP vezérlőjel. Végül kiíródik a TESZTSOR szöveg, ami a sor elejét, a 10 REM"-t felülírja.

Hogyan lehet ebből listázás elleni védelmet csinálni?

Pl. így:

```
0 REM  **** P54.  ****
1 :
2 :
100 REM"■090 SYS 4096

READY.
```

Úgy tűnik, mintha a 90-es sorban egy gépi kódú programot hívnánk. Ha ezután a 100-as sortól a listát egy másik módszerrel eltüntetjük, senki nem fogja az elrejtett BASIC programra listáját keresni, mert látja a gépi kódú program való hivatkozást.

Vagy így:

```
0 REM  **** P55.  ****
1 :
2 :
100 PRINT"LEMEZVE!":REM"

READY.
```

A program a kritikus sort kilistázza, és a következő utasításnak megfelelően azonnal felülírja.

Érdemes tovább próbálkozni ezekkel a mesterséges vezérlőjelekkel. Számos további felhasználási lehetőségre bukkanhatunk.

3.5 Védekezés POKE utasításokkal

Az eddigiekben a LIST rutint többé-kevésbé sikerült kikerülnünk. A következő listázás elleni védelmi módszer éppen a LIST rutin módosításán alapszik.
Ez a módszer a 774-es és a 775-ös ($0306–$0307) címeket használja.
Ezeken a címeken helyezkedik el az ún. LIST vektor, amely normális esetben a $A71A címre mutat. Ez a cím a BASIC értelmezőben van és a BASIC tokenek érthető utasításszóvá alakítását végzi el.
Ezt a rutint használja a LIST parancs is. A LIST vektor megváltoztatásával egy tetszőleges rutinra ugorhatunk, és így megváltoztathatjuk a LIST parancsot.
Nézzünk egy példát erre:

```
POKE 774,226: POKE 775,252
```

Írjuk be a megadott POKE utasításokat. Ha ezután használjuk a LIST parancsot, akkor a gép a bekapcsolási állapotba kerül vissza.
Az alapelv ebből is világosan látszik. A LIST vektort átírtuk a POKE utasításokkal a $EA31 címre, amely a RESET rutin kezdőcíme. A LIST paranccsal erre a

címre ugrottunk. Hasonló módon megadhatjuk egy saját rutin kezdőcímét is, amelyet a LIST paranccsal hívunk.

A módszernek látható hátránya, hogy a védelem csak a program futtatása során lép működésbe.
Autostarttal egybekötve viszont már érdekes eredményt mutat.

3.6 A "veszélyes" billentyűk blokkolása

Számítógépünkön van néhány, a programszerzők szempontjából "veszélyes" billentyű. A programozók szeretnék pl. meggátolni, hogy programjukat futás közben le lehessen állítani.
Lássuk először a RUN-STOP billentyűt, ami a programfutást szakítja meg. Nem jelenthet komolyabb akadályt, hogy a megszakító vektort, amely alapállapotban a $EA31 címre mutat, három byte-tal feljebb toljuk. Ezzel elértük a RUN-STOP lekérdezés átugrását.
Ez a következőképpen néz ki:

POKE 788,PEEK(788)+3 vagy egyszerűen POKE 788,52

Visszatérés az alapállapotba:

POKE 788,PEEK(788)–3 vagy POKE 788,49

A RUN-STOP billentyűvel a kikapcsolása után is dolgozhatunk: Lekérdezhetjük BASIC-ből a többi billentyűhöz hasonlóan:

```
0 REM  **** P56.  ****
1 :
2 :
100 GET A$:IF A$="" THEN 100
110 IF A$=CHR$(3) THEN PRINT"A STOP BILLENTYU LE VAN NYOMVA!"

READY.
```

A programban a STOP billentyűvel visszatérhetünk a menühöz stb.
Hasonló gondokat okozhat a RESTORE billentyű is, amivel szintén meg lehet állítani a futó programokat. De nem kell megijedni, vele is el fogunk bánni!
Most az NMI-vektort hívjuk segítségül, ami alapállapotban az $FE47 címre mutat. Ha ezt a címet $FEC1-re átírjuk, akkor ezt a problémát is megoldottuk. Ezen a címen ui. az NMI rutint záró RTS utasítás van.
Ezt a következő utasítással érhetjük el:

POKE 792,193 az NMI kikapcsolása

Visszatérés az alapállapotba:

```
POKE 792,71                    az NMI bekapcsolása
```

Ha a RUN-STOP és RESTORE billentyűket egyidejűleg akarjuk kikapcsolni, akkor használjuk az ún. STOP vektort. Ez alapállapotban a $F6ED címre mutat. Azért érdemes ezt a vektort használni, mert így egyszerűen két legyet üthetünk egy csapásra.
Az utasítás:

```
POKE 808,254
```

Ez a POKE utasítás a RUN-STOP és RESTORE billentyűk letiltásán kívül a listázást is többé-kevésbé meggátolja, ami azonban a program futását nem zavarja.
Visszatérés az alapállapotba:

```
POKE 808,237
```

Egy programot félbeszakíthatunk POKE utasítással, de kívülről egy hardver RESET-tel is. Az előbbi után a program többé nem listázható minden további nélkül. Az utóbbi esetén a hardver RESET működését szoftver úton is megakadályozhatjuk.
Erről bővebben a *Megszakítások* c. fejezetben olvashatunk.

Van azonban még néhány "veszélyes" billentyű, melyekkel ugyan nem lehet a programot megállítani, de zavaró hatásúak lehetnek. A SHIFT és C= billentyű együttes lenyomására a számítógép karakterkészletet vált. Ezt a

```
POKE 657,128
```

utasítással gátolhatjuk meg, és a

```
POKE 657,0
```

utasítással térhetünk vissza alapállapotba.

Ezzel a "kikapcsolás-veszélyes" billentyűk ismertetését befejeztük.
A következő táblázat néhány további POKE utasítás hatását mutatja be.

Sok sikert a használatukhoz!

Hatás	POKE utasítások
a STOP kikapcsolása	POKE 788,52: POKE 808,239
visszakapcsolása	POKE 788,49: POKE 808,237
a STOP, a RESTORE és a LIST kikapcsolása	POKE 808,234 vagy POKE 808,225
visszakapcsolása	POKE 808,237
a RESTORE kikapcsolása	POKE 793,203 vagy POKE 792,193
a SAVE letiltása	POKE 818,32: POKE 819,245
visszaállítása	POKE 818,237: POKE 819,245
a LIST letiltása	POKE 775,200
megengedése	POKE 775,167
A billentyűzet kikapcsolása	POKE 649,0
visszakapcsolása	POKE 649,10

3.7 Egy gépi kódú játék szimulálása

Sok kísérletet tettek már arra, hogy a BASIC programok listázás és másolás elleni védelmét valóban biztosan megoldják. Ám aki egy kis szakmai ismerettel és gépi kódú monitorral rendelkezik, meg tudja fejteni ezeket a védelmeket. Sokan nem nyúlnak a gépi kódú programokhoz, mivel nem értik a gépi kódot, s így azt hiszik, hogy azok megváltoztathatatlanok.
Egy program értékét külön növeli, ha a felhasználó megtudja, hogy az gépi kódú játék.
Rutinunk egy gépi kódú programot szimulál, amely azonban alapjaiban BASIC nyelvre íródott.
Az ilyen program arról ismerhető fel, hogy csak egy BASIC sora van, ami SYS utasítást tartalmaz, vagyis ugrást a gépi kódú rutinra.
Hogyan érhetjük el azt, hogy a listából csak ez a SYS-t tartalmazó sor jelenjen meg?
Egészen egyszerűen. Először adjuk meg a SYS utasítást. Aztán vagy hagyjuk el az egész szöveget, vagy változtassuk meg tetszés szerint. A sorszámokat szabadon választhatjuk.
A példák:

```
1984 SYS(2110) BY MEDLAY SPARROW
  10 SYS 2110
```

Miután kiválasztottuk a nekünk szimpatikus formát, írjuk le egy BASIC sorba. Majd vigyünk be néhány sort:

```
POKE 43,80: POKE 2127,0: NEW
```

```
0 REM  **** P57.  ****
1 :
2 :
10 FOR I = 2110 TO 2126
20 READ A
30 POKE I,A
40 NEXT I
50 DATA 169,80,133,43,169,52,141,20,3
60 DATA169,193,141,24,3,76,113,168

READY.
```

A DATA sorok ellenőrzését elhagytuk, mivel olyan kevés volt belőlük. Ezért most fokozottabban ügyeljünk a pontos beírásra.
A rutin rövid gépi kódú listája:

```
0 REM  **** P58.  ****
1 :
2 :
10 :      #$80    ;UJ BASIC-KEZDET
15 :
20 :      0840    STA $2B       ;TAROLAS
30 :      0842    LDA #$34      ;KIKAPCSLOJA A RUN-STOP-OT
40 :      0844    STA $0314
50 :      0847    LDA #$C1      ;KIKAPCSOLJA A RESTORE-T
60 :      0849    STA $0318
70 :      084C    JMP $A871     ;A RUN RUTIN HIVASA

READY.
```

Indítsuk a programot RUN paranccsal, és töröljük ki. Ezután töltsük be a saját, védeni kívánt programunkat. Fontos, hogy a program 0-ás sorral kezdődjön. Írjuk be a

POKE 43,1

utasítást, és tároljuk a programot a szokásos módon. Ha most programunkat visszatöltjük a gépbe, akkor a LIST utasításra csak a SYS-t tartalmazó sor jelenik meg. A program RUN-nal indítható. Hatására a gép kis rutinra ugrik, amely feltölti a tárolót és kikapcsolja a RUN-STOP és a RESTORE billentyűket. A BASIC program elindul, de elindítás után már egyszerű módon nem állítható meg.

Ez a rutin sem ad 100%-os védelmet, de a többi bemutatott módszerrel együtt alkalmazva megakadályozhatja a jogtalan hozzáférést.

4. FEJEZET
PARANCSBŐVÍTÉS SAJÁT KEZŰLEG

Saját parancsainkat többféle módon fűzhetjük az operációs rendszerhez. Ezek közül hármat mutatunk most be:
a) a BASIC-CODE-LINK-ek megváltoztatása;
b) a CHRGET-rutin megváltoztatása;
c) az IRQ rutin (megszakító rutin) megváltoztatása.

4.1 A BASIC-CODE-LINK-ek megváltoztatása

Több parancs összefűzésének ez a leghatékonyabb módszere.
Azt a vektort fogjuk felhasználni, amely hatására a program a BASIC utasításokat kiértékelő rutinra ugrik. A vektor címe $0308–$0309 (776–777). A vektort saját gépi kódú rutinnal fogjuk megváltoztatni. A programot helyezzük el a kazettapufferbe ($033C–$03FB):

```
0 REM **** P59. ****
1 :
2 :
3 OPEN1,8,1,"@0:P 59G"
4 OPEN2,4
5 SYS36864
6 .OPT O1,P2
7 *= $033C
10 LDA #$47
20 STA $0308
30 LDA #$03
40 STA $0309
50 RTS
60 JSR $0073
70 CMP #$5F
80 BEQ $0351
90 JMP $A7E7
100 LDA #$05
110 STA $0286
120 JMP $A7E4

READY.
```

A program két részre osztható:
az 1. rész ($033C–$0346) a BASIC-CODE-LINK-et a 2. részre irányítja;
a 2. rész ($0347–$0354) a tulajdonképpeni parancsbővítményeket az eredeti parancsok után írja.

A 2. rész utasításai:

JSR $0073	A gép a megadott következő karakterre áll (CHRGET rutin)
CMP #$5F	A karakter összehasonlítódik a "balra nyíl" ASCII-kódjával
JMP $A7E7	Mivel a karakterek nem egyenlők, a program a vektor által megadott címre ugrik. Itt megvizsgálja, hogy a karakter más normál BASIC utasítás-e
LDA #$05	
STA $0286	Ez a tulajdonképpeni parancsbővítmény. Esetünkben üres, a betűszín egyszerűen zöldre változik
JMP $A7E4	A gép visszatér a normál üzemmódba
Próbáljuk ki!	A BASIC betöltőprogram:

```
0 REM  **** P60. ****
1 :
2 :
10 FORX=828TO856:READA:POKEX,A:NEXTX
20 DATA169,71,141,8,3,169,3,141,9,3,96
30 DATA32,115,0,201,95,240,3,76,231,167,169,5,141,134,2,76,228,167

READY.

RUN

SYS 828
```

Ha most lenyomjuk a "balra nyíl" billentyűt (a RETURN-t is), akkor a betűszín zöldre vált.
Mint látható, ez nem is volt olyan bonyolult. Az ilyen programozási módnak azonban van egy nagy hátránya: nagyon lassú, mivel több parancsbővítmény esetén a keresési idő jelentősen megnő.
Ezt a hátrányt a következőképpen küszöbölhetjük ki: minden új parancs elé tegyünk egy ismertetőjelet, pl. "nyíl balra" vagy felkiáltójelet. Így a programban csak ezt az ismertetőjelet kell lekérdezni. Ha ilyen nincs, akkor a gép a billentyűzet-dekódolóra ugrik, egyébként pedig megvizsgálja a parancsot.

A jobb érthetőség kedvéért egy példaprogram:

```
0 REM  **** P61. ****
1 :
2 :
3 OPEN1,8,1,"@0:P 616"
4 OPEN2,4
5 SYS36864
6 .OPT O1,P2
7 *= $033C
10 LDA #$47
20 STA $0308
30 LDA #$03
40 STA $0309
```

```
50 RTS
60 JSR $0073
70 CMP #$21        ;A "!"-JEL ASCII-KODJA
80 BEQ $0351       ;UJ PARANCS KERESESE
90 JMP $A7E7       ;NINCS UJ PARANCS
100 JSR $0073      ;KIOLVASSA A ! UTANI JELET
110 CMP #$91       ;ON  ($91=AZ ON TOKENJE)
120 BEQ $035F      ;IGEN
130 CMP #$A8       ;NOT ($A8=A NOT TOKENJE)
140 BEQ $0374      ;IGEN
150 JMP $AF08      ;NINCS ERVENYES PARANCS, SYNTAX ERROR
160 SEI            ;ON PARANCS
170 LDA #$6E       ;AZ IRQ-T $036E-RE TESZI
180 STA $0314
190 LDA #$03
200 STA $0315
210 CLI
220 JMP $A7E4      ;AZ ON UTASITAS LEZARASA
230 INC $0286      ;UJ IRQ
240 JMP $EA31      ;REGI IRQ TOVABB
250 SEI            ;NOT UTASITAS
260 LDA #$31       ;AZ IRQ VISSZAHELLITASA
270 STA $0314
280 LDA #$EA
290 STA $0315
300 CLI
310 JMP $A7E4      ;A NOT UTASITASA LEZARASA

READY.
```

A BASIC betöltőprogram:

```
0 REM  **** P62. ****
1 :
2 :
10 FORX=828TO898:READA:POKEX,A:NEXTX
20 DATA169,71,141,8,3,169,3,141,9,3,96
30 DATA32,115,0,201,33,240,3,76,231,167,32,115,0,201,145,240,
   7,201,168
40 DATA240,24,76,8,175,120,169,110,141,20,3,169,3,141,21,3,88,
   76,228,167
50 DATA238,134,2,76,49,234
60 DATA120,169,49,141,20,3,169,234,141,21,3,88,76,228,167
```

A RUN és SYS 828 utasítások után két új paranccsal rendelkezünk. A !ON az IRQ vektort a $036E címre állítja. Így a betűszín kódja minden 1/60 s-ban eggyel nő.

Ezt az új hatást a második új parancsunkkal állíthatjuk le: !NOT.

Az ismertetőjelnek azonban van más jó tulajdonsága is: a régi utasításoknak új funkciót adhatunk anélkül, hogy az eredeti utasítást elveszítenénk.

Még néhány tipp

Elegánsabb megoldás volna, ha az utasításokat egy táblázatba foglalnánk, és a beadott utasítást a táblázat elemeivel sorban összehasonlítanánk (hasonló módon dolgozik a BASIC értelmező).

Nagyon érdekes az új utasítások tokenizálása is. Ezzel tárolóterületet takarít-hatunk meg.
A tokenizáló vektor címe: $0304–$0305 (772–773). A következő tokenek szabadok: 202–254.

4.2 A CHRGET rutin megváltoztatása

A parancsbővítés lehetséges módja az $0073–$0084 tartományban található ún. CHRGET rutin megváltoztatása. Ez a következőképpen néz ki:

```
0 REM **** P63. ****
1 :
2 :
10 :     0073  INC $007A
20 :     0075  BNE $0079
30 :     0077  INC $007B
40 :     0079  LDA $HHLL    :LEHIV EGY KARAKTERT A BASIC SZOVEGBOL
50 :     007C  CMP #$3A
60 :     007E  BCS $008A
70 :     0080  CMP #$20
80 :     0082  BEQ #$20
90 :     0084  SEC
100 :    0085  SBC #$30
110 :    0087  SEC
120 :    0088  SBC #$D0
130 :    008A  RTS
```

Mivel ez a rutin a RAM-ban helyezkedik el (bekapcsoláskor a ROM-ból átmásolódik a RAM-ba), tetszés szerint megváltoztathatjuk. A $0073–$007B vektor tartalma maradhat az eredeti. A $007C-n a JMP $ (saját program) utasításnak kell állni.
Programunk megvizsgálja, hogy a következő beolvasott karakter új utasítás-e. Ha nem, akkor az eredeti CHRGET rutin hajtódik végre.
Példaként adjunk a NEW parancsnak egy új jelentést: feleljen meg egy RESET-nek.
Az új rutin ismét a kazettapufferben helyezkedik el.

```
0 REM  **** P64. ****
1 :
2 :
3 OPEN1,8,1,"@0:P 64"
5 SYS36864
6 .OPT O1,P1
7 *= $033C
10 LDA #$4C
20 STA $007C
30 LDA #$49
40 STA $007D
50 LDA #$03
60 STA $007E
```

```
70 RTS
80 CMP #$82      ;NEW
90 BEQ $035F     ;IGEN
100 CMP #$83      ;NEM, DE A CHRGET FOLYTATODIK
110 BCS $035E
120 CMP #$20
130 BNE $0358
140 JMP $0072
150 SEC
160 SBC #$30
170 SEC
180 SBC #$D0
190 RTS
200 JMP $FCE2

READY.
```

Az első részben egyszerűen megváltoztattuk a CHRGET rutint. A második részt a CHRGET rutin minden programfutáskor átugorja.

Végül a BASIC betöltőprogram:

```
0 REM  **** P65.  ****
1 :
2 :
10 FOR X= 828 TO 865: READ A: POKE X,A: NEXT X
20 DATA 169,76,133,124,169,73,133,125,169,3,133,126,96
30 DATA 201,162,240,18,201,58,176,13,201,32,208,3,76,114,0
40 DATA 56,233,48,56,233,208,96,76,226,252

READY.
```

Ha most kiadunk egy NEW parancsot, akkor a számítógép egy RESET-et hajt végre. Mint láthattuk, ez sem nehéz módszere a parancsbővítésnek.

4.3 Az IRQ rutin megváltoztatása

Elgondolkodtató, hogy ezt a megoldást miért alkalmazzák olyan ritkán parancsbővítésre.
Mindnyájan tudjuk, hogy az IRQ rutin (a megszakító rutin) minden 1/60 s-ban végrehajtódik.
Ezt használjuk ki a következő gépi kódú programban:

```
0 REM  **** P66.  ****
1 :
2 :
3 OPEN1,8,1,"@0:P 66G"
4 OPEN2,4
5 SYS36864
6 .OPT O1,P2
```

```
7 *= $033C
10 SEI
20 LDA #$49
30 STA $0314
40 LDA #$03
50 STA $0315
60 CLI
70 RTS
80 LDA $CB
90 CMP #$39
100 BEQ $0352
110 JMP $EA31
120 JSR $E544
130 JMP $EA31

READY.
```

A BASIC betöltőprogram:

```
0 REM  **** P67.  ****
1 :
2 :
10 FOR X= 828 TO 855: READ A: POKE X,A: NEXTX
20 DATA 120,169,73,141,20,3,169,3,141,21,3,88,96
30 DATA 165,203,201,57,240,3,76,49,234,32,68,229,76,49,234

READY.
```

A RUN és SYS 828 megadása után látszólag nem történik semmi. Ha viszont lenyomjuk a "balra nyíl" billentyűt, akkor a képernyő törlődik.
Figyelem! Előfordulhat, hogy a művelet után a kurzor nem látszik. Nem kell megijedni, mert ha írunk a képernyőre, akkor azonnal megjelenik.
A magyarázat: az IRQ rutint egy saját rutinra irányítottuk. Itt megvizsgáljuk, hogy lenyomtunk-e egy billentyűt (esetünkben a "balra nyíl" billentyűt). Ha igen, akkor törlődik a képernyő.
Az eredeti IRQ rutinra minden esetben vissza kell ugrani.
Bizonyára mindenki észrevette, hogy a parancsbővítés előzőekben bemutatott módszerei és e módszer között van egy nagy különbség.
Az előző két esetben az utasítást RETURN-nek kellett követni, míg itt azonnal végrehajtódott.

Miután megismertük az új utasításoknak az értelmezőben való elhelyezési módjait, mindenki kipróbálhatja a számára legfontosabb saját utasítások írását. Ehhez mind a három módszer jól használható.

5. FEJEZET
GRAFIKA

5.1 Alapfogalmak

Előbb-utóbb mindenki eljut odáig, hogy a C 64-es billentyűzetén keresztül elérhető grafikus karakterek már nem felelnek meg programozói elképzeléseinek. Ekkor felmerül az igény a karaktergenerátor megismerésére. Mi is tulajdonképpen a karaktergenerátor?
A billentyűzetről a képernyőre kerülő minden jelnek jól meghatározott karakternek kell lenni.
A tárnak azt a területét, ahol ezek a jelek tárolódnak, karaktergenerátornak nevezzük. Ez a ROM-ban található, a $D000–$DFFF címeken. Felmerül a kérdés: hogyan tárolja a számítógép ezeket a karaktereket?
A kérdés jobb megértéséhez rajzoljuk fel a képernyő egy tetszőleges karakterének felnagyított mátrixát. Legyen az a karakter pl. az A betű:

```
 76543210                bit-helyek
1. . .**. . .1
2. .****. .2
3.**. .**.3
4.******.4
5.**. .**.5
6.**. .**.6
7.**. .**.7
8. . . . . .8
 76543210
```

Mint az ábrából is látható, a karakter egy 8×8-as pontmátrixban helyezkedik el. Ez összesen 64 pont.
A karakter alakját digitális jelekké kell alakítani. Az átalakítás soronként 1 byte-ot jelent, amelynek minden bitje egy pontnak felel meg a sorban.
Egy példa biztosan érthetőbbé teszi a feladatot. Tekintsük pl. az A betű 3. sorát. Ez a következőképpen néz ki:

```
.**. .**.
```

A byte értékének kiszámításához a megfelelő helyiértékű biteket össze kell adni.

```
2↑7  2↑6  2↑5  2↑4  2↑3  2↑2  2↑1  2↑0  a bitek értéke
128  64   32   16   8    4    2    1
.    *    *    .    .    *    *    .
0+   64+  32+  0+   0+   4+   2+   0 =  102 (a byte értéke)
```

Aki az eddigiekből nem értette volna meg, hogyan kell a karakterek egy sorának megfelelő byte-értéket kiszámítani, nézze meg a következő kis gyakorlóprogramot.

A változók:

B Az aktuális bit értéke (2↑A)
S$ A kiszámítandó sor
BY A byte értéke

A program listája:

```
0 REM  **** P68. ****
1 :
2 :
10 REM KARAKTERGENERATOR GYAKORLOPROGRAM
20 PRINTCHR$(147):POKE53280,1:POKE53281,1:POKE646,6:REM BETUSZIN
30 CLR:PRINT"■ BYTE - BIT - GYAKORLOPROGRAM           ■":PRINT
40 PRINT"[1] SOR BEVITELE":PRINT
50 PRINT"[2] BYTE KIIRASA":PRINT:PRINT
60 PRINT"KEREM VALASSZON !"
70 GETA$:IFA$=""THEN70
80 IFA$="1"THEN200
90 REM *** BYTE BEVITELE ***
100 PRINTCHR$(147)
110 INPUT"A BYTE ERTEKE (0-255)";BY:PRINT:IFBY<0ORBY>255THEN110
120 FOR A=7TO0STEP-1
130 B=2↑A
140 IFB<=BYTHENPRINT"●";:BY=BY-B:GOTO160
150 PRINT"■O■";
160 NEXT
170 PRINT"■■NYOMJON LE EGY BILLENTYUT!"
180 GETA$:IFA$=""THEN180
190 GOTO20
200 REM *** SOR KIIRASA ***
210 PRINTCHR$(147)
220 PRINT"X=A PONT ERTEKE 1   M=A PONT ERTEKE 0":PRINT:PRINT
230 FOR A=7TO0STEP-1
240 PRINT8-A"PONT ?";
241 GETA$:IFA$<>"M"THENIFA$<>"X"THEN241
250 PRINTA$:IFA$="X"THENBY=BY+2↑A:S$=S$+"●":GOTO270
260 S$=S$+"■O■"
270 NEXTA
280 PRINT:PRINTS$"="BY
290 PRINT:PRINT"NYOMJON LE EGY BILLENTYUT!"
300 GETA$:IFA$=""THEN300
310 GOTO20

READY.
```

A program indítása után két lehetőség közül választhatunk:

– sor bevitele vagy
– byte bevitele.

Ha a sor bevitelét választjuk, akkor a nyolc pontot egy jelsoron állíthatjuk be. Pontot X-el lehet elhelyezni, s egy másik betűvel lehet kitörölni. Miután mind a nyolc pontot meghatároztuk, a sor megjelenik a képernyőn, és kiszámítódik a byte értéke.

A byte bevitelének választásakor minden fordított sorrendben megy végbe. A byte értékét kell megadni 0 és 255 között, és a program kinyomtatja a megfelelő sort.

5.2 A karaktergenerátor a tárolóban

Az alábbi elrendezés a karaktergenerátor ROM-beli elhelyezkedését szemlélteti.

```
                $D000 (53248)
1a                    nagybetű/grafika
                      normál üzemmód
                $D400 (54272) 1. karakterkészlet
1b                    nagybetű/grafika
                      inverz üzemmód
                $D800 (55296)
2a                    kisbetű/nagybetű
                      inverz üzemmód
                $DC00 (56320) 2. karakterkészlet
2b                    kisbetű/nagybetű
                      inverz üzemmód
                $DFFF (57343)
```

A karaktergenerátor tehát két blokkra oszlik, melyek szintén két részből állnak. Az 1. blokk a C 64-es 1. karakterkészletét tartalmazza: a nagybetűs és a grafikus üzemmódot.
A 2. blokk az előzőeknek megfelelően a 2. karakterkészletet tartalmazza: a kisbetűs és a nagybetűs üzemmódot.
A blokkok két részre tagozódnak: a normál és az inverz üzemmódú részre.
Ennek alapján kiszámítható a karaktergenerátor mérete. Egy karakter tárolásához 8 byte-ra van szükség (soronként 1 byte). Egy blokkban 256 karakter van, 128 normál és 128 inverz karakter.
A következő program megjeleníti a teljes karakterkészletet a képernyőn:

```
0 REM  **** P69.  ****
1 :
2 :
10 FOR A=0 TO 255: POKE 1024+A,A: POKE 55296+A, 1: NEXT

READY.
```

Mivel a 2. karakterkészlet is 256 karakterből áll, ez összesen 512 karaktert jelent. Így a karaktergenerátornak 512 * 8 = 4 Kbyte méretűnek kell lenni.

5.3 A karakterkészlet kiolvasása

A megoldandó probléma: a karaktergenerátor a ROM-ban helyezkedik el, így a tartalmát nem tudjuk módosítani.
Ezen úgy fogunk segíteni, hogy a karaktergenerátort áttöltjük egy RAM területre. Az áttöltést az alábbi rövid programmal végezhetjük el:

```
0 REM  **** P70.  ****
1 :
2 :
10 DIM B(4095)
20 FOR A=0 TO 4095: REM $D000 - $DFFF
30 B(A) =PEEK(53248+A)
40 NEXT A: REM A KIOLVASAS BEFEJEZODOTT
50 FOR A=0 TO 4095
60 PRINT B(A);: NEXT A
70 END

READY.
```

A program a tárolótartalmakat kb. 1 perc alatt olvassa ki, majd nagyon sok számot ír a képernyőre.
Már az első pillantásra látható, hogy nem a karaktergenerátor tartalmát olvastuk ki: a kiírt számok közül a legtöbb 240 fölött van vagy 0.
Nos, a hibát hamar megtaláltuk: az $D000–$DFFF közti terület kétszeresen van lefedve. A krakter-ROM mellett egy RAM tároló is van ugyanebben a címtartományban.
Ezen a RAM területen helyezkednek el a VIC-ek, amelyek pl. a sprite-okat vezérlik, a SID (Sound Interface Device), valamint a színtároló.
Tehát a kiolvasás ezzel a módszerrel nem járt eredménnyel. A számunkra érdekes ROM helyett az érdektelen RAM-ot olvastuk ki.
Ahhoz, hogy a ROM-hoz hozzáférjünk, néhány információt meg kell tanulnunk a számítógépről. A ROM hozzáférésével egy időben azonban a számítógép nem tudja a RAM-ot is kezelni.
A ROM-ot akkor másolhatjuk, ha az 1-es cím (processzor port) második bitjét töröljük. Ezzel egy időben a megszakítást meg kell akadályozni (SEI). Ez BASIC-ben a következőképpen néz ki (a sorokat az előző programba kell beilleszteni):

5 POKE 56334,0: POKE 1,51: REM megszakítás ki, hozzáférés lehetséges

45 POKE 1,55: POKE 56334,1: REM megszakítás be, CHAR-ROM ki

A módosított program futtatása előtt két dologra hívjuk fel a figyelmet. Az 56334-es cím segítségével a megszakítást kikapcsoltuk. Ezért a már elindított program többé nem szakítható meg a RUN-STOP/RESTORE billentyűkkel.
Soha ne adjuk ki a POKE 1,51 utasítást a megszakítás kikapcsolása előtt, különben a számítógép üzemképtelenné válik!

Indítsuk el a módosított programot! Kb. 1 perc múlva rengeteg szám jelenik meg a képernyőn. Ez azonban már a karaktergenerátor valódi tartalma.
Ezzel a számhalmazzal viszont nem megyünk sokra. A szemléletesség kedvéért át kell még őket alakítani. A következő programrészlet látható jelekké alakítja a számokat.
A programsorokat fűzzük az előző programunkhoz. A régi lista 50–70-es sorai átíródnak.

A változók:

B(0)–B(4095)	A karaktergenerátor tartalma
A	Kapcsoló: 0 = pontbeültetés
X	Ciklusváltozó, pontokat hoz egy sorba
2↑X	A mindenkori bit-érték
Z	A mindenkori karakterérték

```
0 REM  **** P71.  ****
1 :
2 :
10 DIM B(4095)
20 FOR A=0 TO 4095: REM $D000-$DFFF
30 B(A)=PEEK(53248+A)
40 NEXT A: REM A KIOLVASAS BEFEJEZODOTT
50 REM ***A MEGVALTOZTATOTT KARAKTEREK ERTEKEI
55 Z=B(I)
60 FOR X=7 TO 0 STEP-1
70 A=Z AND 2↑X
80 IF A THEN PRINT ".";: GOTO 100: REM A=0, BEULTETETT PONT
90 PRINT " ";:REM NEM BEULTETETT PONT
100 NEXT X: PRINT
110 I=I+1: IF I=4096 THEN END
120 GOTO 50
```

Indítsuk el a bővített programunkat. Ez ismét kb. 1 percig fut, amíg a tároló tartalmát kiolvassa. Néhány dologban azonban eltér az előbbitől.
Az értelmetlennek tűnő számhalmaz helyett kinagyított karakterek jelennek meg a képernyőn.

5.4 A karaktergenerátor másolása

Mint már tudjuk, az eredeti karaktergenerátor a ROM-ban található. Ahhoz, hogy megváltoztathassuk, át kell másolnunk a módosítható RAM területre. Ez már nem okoz nehézséget, mert megismertük a generátor kiolvasásának módját. A gond már csak az, hogy hová érdemes az új, változtatható karakterkészletet elhelyezni. Erre több lehetőség kínálkozik.

a) **A BASIC területre**

A karaktergenerátor másolható pl. a BASIC terület kezdetére. Ehhez a BASIC startot megfelelően el kell tolni. A BASIC terület 2 kbyte-tal csökken. Ugyanilyen alapon másolhatjuk a karaktergenerátort a BASIC terület végére is. Ekkor nem kell bajlódnunk a körülményes BASIC start megváltoztatásával.

b) **A ROM alá**

Ha az operációs rendszert valamilyen felhasználói okból nem írtuk át a RAM területre, akkor itt 8 kbyte szabad RAM mindig a rendelkezésünkre áll. Mivel azonban a PEEK utasítás a ROM területet használja, nem tudjuk az ott levő átmásolt karakterkészletet kiolvasni (ami nem feltétlenül zavaró). A probléma megoldására használjuk az első kötetben* ismertetett módosított PEEK függvényt.

A következőkben két másolóprogramot találunk. Az egyik a karaktergenerátort a BASIC terület elejére, a másik az operációs rendszer ROM-ja alá helyezi át.

1. másolóprogram: a BASIC terület elejére

Írjuk be az alábbi utasításokat, majd töltsük be, s indítsuk el a programot:

```
POKE 44,16: POKE 16+256,0: NEW

0 REM  **** P72.  ****
1 :
2 :
10 POKE 56334,0: POKE 1,51: REM A KARAKTER-ROM ATIRASA
20 FOR K=0 TO 2047
30 POKE 2048+K, PEEK(53248+K)
40 NEXT
50 POKE 1,55: POKE 56334,1
60 POKE 53272,PEEK(53272) AND NOT 12 OR 2:REM ATKAPCSOLAS
```

A karaktergenerátor kezdőcíme: 2048 ($0800),
a képernyő-RAM: 1024.

(Belső okokból a karaktergenerátor számára a BASIC terület kezdetén csak 2 kbyte áll rendelkezésére. Ezért csak az egyik karakterkészletet választhatjuk.) Esetünkben a nagybetű/kisbetű üzemmódot választjuk, ezért írjuk át a 30-as sort a következőre:

```
30 POKE 2048+K, PEEK(55296+K)
```

* Tippek és trükkök a Commodore 64-eshez, DATA BECKER–NOVOTRADE 1985.

2. másolóprogram: a ROM alá

A BASIC startot a program nem érinti.

```
0 REM  **** P73.  ****
1 :
2 :
10 POKE 56334,0: POKE 1,51: REM ATTERES A KARAKTER-ROM-RA
20 FOR K=0 TO 4095
30 POKE 57344+K, PEEK(53248+K)
40 NEXT K
50 POKE 1,55: POKE 56334,1
60 POKE 56576,196:REM A FELSO 16-K-S BLOKK
70 POKE 648,3*64+4:REM KEPERNYO-RAM (3*64+4)*256-TOL
80 POKE 53272,PEEK(53272) AND NOT 6 OR 8: REM ATKAPCSOLAS

READY.
```

A karaktergenerátor kezdőcíme: 57344 ($E000),
a képernyő-RAM: 50176

További példáinkban a 2. másolóprogrammal átmásolt karaktergenerátort használjuk. Ezért a következőket vegyük figyelembe:
– a képernyőtároló kezdőcíme 1024 helyett 50176 lett;
– a normál üzemmódra való visszatérés utasításai

POKE 56576,199: POKE 648,4: POKE 53272,21

– a RUN-STOP/RESTORE hatására nem tér vissza a számítógép a normál üzemmódba. Ehhez még a

POKE 648,4

utasítást is be kell írni.

Töltsük be a 2. másolóprogramot és indítsuk el. Kb. 60 s múlva jelenik meg ismét a kurzor. Ha helyesen vittük be a programot, akkor megjelenik a kívánt karakterkészlet. Példaként változtassunk meg egy karaktert. A karaktergenerátor első jele az "egységár" jel. Ennek az alakját fogjuk megváltoztatni. A karaktergenerátor első 8 byte-ja az "egységár" jelet tartalmazza: 57344–57351. Az 57344-es cím a felső, a 57351-es cím az alsó sorát írja le.
Változtassuk meg a karakter felső részét egy kicsit:

POKE 57344,255

Nyomjuk le az "egységár" billentyűt! A felső sora teljesen ki van húzva. Az alsó sort hasonló módon megváltoztathatjuk:

POKE 57351,255

5.5 A karaktergenerátor átkapcsolása

Ha jól megnézzük az előző két másolóprogramot, akkor látjuk, hogy a tulajdonképpeni másolás mellett még három címmel találkozhatunk: 53272, 56576, 648. Mit jelentenek ezek a címek?
A számítógépnek tudnia kell, hogy hol helyezkedik el a tárolóban az új karakterkészlet. Erre való az 53272-es cím. Az 1.–3. bitek a karakterkészlet eltolását tartják számon, a 0. bit nincs felhasználva, a 4.–7. bitek a képernyőtárolót tolják el.

A következő táblázatban áttekintést adunk a lehetséges kezdőcímekről és hozzájuk tartozó bitkombinációkról.

Képernyőtároló	Karakterkészlet
0000xxxx 0	xxxx000x 0
0001xxxx 1024 (normál)	xxxx0001 2048
0010xxxx 2048	xxxx010x 4096
0011xxxx 3072	xxxx011x 6144 (normál)
0100xxxx 4096	xxxx100x 8192
0101xxxx 5120	xxxx101x 10240
0110xxxx 6144	xxxx110x 12288
0111xxxx 7168	xxxx111x 14336
1000xxxx 8192	
1001xxxx 9216	
1010xxxx 10240	
1011xxxx 11264	
1100xxxx 12288	
1101xxxx 13312	
1110xxxx 14336	
1111xxxx 15360	

Ha a képernyőtárolót eltoljuk, akkor egyidejűleg meg kell változtatni a video-RAM felső byte-ját is:

POKE 648,(kezdőcím)*256
Amint látjuk, mind a karakterkészlet, mind a képernyőtároló csak az első 16 kbyte-on belül tolható el. Hiányzik az a négy címbit, amelyek segítségével az egész tárolóban mozoghatnánk. E hiányosság kiküszöbölésére szolgál a 56576 cím.
Ha nem az első 16 kbyte-os területet választjuk, akkor a cím segítségével automatikusan átugorhatunk egy másik 16 kbyte-os tartományba. A sprite-blokkok és azok mutatói is a megfelelő lépéssel eltolódnak!
16 kbyte-os tartomány:
$0000 – $3FFF 0 – 16383 POKE 56576,199: POKE648,4
$4000 – $7FFF 16384 – 32767 POKE 56576,198: POKE648,4+1*64

$8000 – $BFFF 32768 – 49151 POKE 56576,197: POKE648,4+2*64
$C000 – $FFFF 49152 – 65535 POKE 56576,196: POKE648,4+3*64
Megjegyzés. Bemutatási célokra a POKE utasításokkal való bevitel még megfelelő. De ha pl. új játékhoz akarunk új karakterkészletet szerkeszteni, akkor a karaktermeghatározás igen hosszadalmas és unalmas útját választjuk. A következő oldalakon levő két lista leegyszerűsíti az új karakterkészlet előállítását.

5.6 Karakterszerkesztő segédprogram

A következőkben egy teljes karakterszerkesztő segédprogram terjedelmes listáját mutatjuk be. A lista valóban nagyon hosszú, de ha megismerkedünk a program nyújtotta széles körű lehetőségekkel, akkor megértjük a terjedelmesség okát.
A program úgy van felépítve, hogy a felhasználónak semmilyen előképzettséggel nem kell rendelkeznie a karaktergenerátorról.
Az egyes karakterek a billentyűzet vagy joystick segítségével hozhatók létre. Az új karakterkészletet lemezen is tárolhatjuk. A program nagy szerkesztői segítséget nyújt: pl. a karakter elforgatása 90, 180 vagy 270 fokkal, a jelek invertálása, felcserélése vagy megduplázása stb. tipusú feladatok megoldásához. A program tartalmaz még egy programgenerátort is, amelynek hatására a max. 256 karakteres karakterkészlet BASIC-betöltőjét is kiírja.

Végül lássuk a listát:

```
 0 REM **** P/4. ****
 1 :
 2 :
 5 REM KESZITETTE:T.WELTNER*BAHNHOFSALLEE 10*3260 RINTELN 1*TEL.:
   (05751) 5676
10 POKE45,255:POKE46,66:CLR
20 POKE788,52:O=53248:POKEO+32,1:POKEO+33,1:PRINT"■■":POKE214,4:
   PRINT
30 IFPEEK(51000)=1THENPOKE51000,0:GOTO350
40 POKE53247,0:PRINTTAB(7)"■                              ":PRINT
50 POKEO+42,3:PRINTTAB(7)"■ *KARAKTERSZERKESZTO*":PRINT
60 PRINTTAB(7)"■■                          ":POKE214,16:PRINT
70 PRINT"■■■L A KARAKTERKESZLET MEGVALTOZTATASA":PRINT
80 PRINT"■■■L BEEPITETT PROGRAMGENERATOR"
90 POKE214,21:PRINT:PRINT"■■■■  T.WELTNER / 1984. SZEPTEMBER 22."
110 :
120 REM *** SPRITE-INICIALIZALAS ***
130 FORK=0TO7:POKE704+K*3,255:POKE705+K*3,0:POKE706+K*3,0:NEXTK
140 FORK=0TO38:POKE704+8*3+K,0:NEXTK
150 FORK=0TO63:POKE832+K,0:NEXT:REM A 13-AS BLOKK TELE
170 :
180 REM *** KURZORVILLOGTATAS ***
190 REMDA:IFA=-1THEN240
200 POKE912+B,A:C=C+A:B=B+1:GOTO190
210 DATA174,142,3,232,224,16,240,6,142,142,3,76,52,234,169,0,
    141,142
220 DATA3,173,143,3,201,4,240,11,141,21,208,169,4,141,143,3,
    76,52,234
```

```
230 DATA141,21,208,169,0,76,175,3,120,169,144,160,3,141,20,3,140,21,
    3,88,96,-1
240 IFC<>6009THENLIST200-230
260 :
270 C=0:REM *** MASULORUTIN ***
280 FORH=0TO48:READB:POKE12*4096+H,B:C=C+B:NEXT
290 DATA162,16,169,0,141,14,220,169,51,133,1,169,208,160,0,132,34,133
300 DATA35,169,112,132,36,133,37,177,34,145,36,200,208,249,230,35,230
310 DATA37,202,208,242,169,55,133,1,169,1,141,14,220,96
320 IFC<>5798THENPRINT"? HIBA A DATAIBAN":LIST290-310
330 SYS49152:REM AKTIVIZALAS
350 :
360 CLR:U=53248:POKE198,0:REM *** MENU ***
370 PRINT"■■■                    * MENU *                              ■■"
380 PRINT"■■ 1 ■■  KARAKTER ELORALLITASA":PRINT
400 PRINT"■■ 2 ■■  ADATBEVITEL":PRINT
410 PRINT"■■ 3 ■■  KARAKTERKESZLET":PRINT
420 PRINT"■■ 4 ■■  KARAKTERCSERE":PRINT
430 PRINT"■■■■";:FORT=0TO34:PRINT"=";:NEXTT:PRINT:PRINT
440 PRINT"■■ 5 ■■  A KARAKTERKESZLET BETOLTESE":PRINT
450 PRINT"■■ 6 ■■  A KARAKTERKESZLET KIMENTESE":PRINT
455 PRINT"■■ 7 ■■  PROGRAMGENERATOR":PRINT
456 PRINT"■■■■";:FORT=0TO34:PRINT"=";:NEXTT:PRINT:PRINT
460 PRINT"■■ 8 ■■  PROGRAM VEGE"
470 GETH$:IFH$=""THEN470
480 IFH$=CHR$(3)THENPRINT"■? A MENU MEGSZAKITASA";:END
490 H=VAL(H$):IFH=0ORH>8THENGOTO470
500 ONHGOTO530,590,800,1020,1490,1270,3390,1230
520 :
530 REM *** KARAKTERSZERKESZTO ***
540 GOSUB1790: REM A KARAKTER LEKERDEZESE
550 GOSUB1970:GOTO2150:REM KEPERNYO INICIALIZALAS/MOZGATO RUTIN
580 :
590 REM *** ADATBEVITEL ***
600 PRINT"■■■                ADATBEVITEL                              "
610 PRINT:PRINT:PRINT"■ADJA MEG EGYMAS UTAN A 8 ADATOT !"
620 PRINT:PRINT:FORVV=0TO7:PRINTVV+1"■";:K(VV)=0:K$(VV)="":
    INPUT".";K$(VV):NEXT
621 FORVV=0TO7:K(VV)=VAL(K$(VV)):IFK(VV)<0ORK(VV)>255THENK(VV)=0
622 NEXTVV
630 F4=5:F3=1:FL=12:GOSUB3210:F3=0:FL=0:F4=0
640 POKE214,15:PRINT:PRINT"TAROLJUK A KARAKTERT (I/N) ?"
650 GETH$:IFH$=""THEN650
660 IFH$="I"THENKL$="■KARAKTERBEVITEL":GOTO720
670 PRINT:PRINT"VISSZA A MENURE (I/N)?"
680 GETH$:IFH$=""THEN680
690 IFH$="I"THENGOTO360
700 GOTO590
705 :
710 REM *** BEVITEL ***
720 P=53246:PRINT"■";KL$:PRINT"■■■KARAKTER?■■";:GOSUB785
721 POKE204,0:GETA$:IFA$=""THEN721
722 A=PEEK(207):IFATHEN722
723 PRINT:IFA$=CHR$(13)THENA$="  "
724 POKE204,1:PRINT"■IMEGVALTOZTATHATO KARAKTER: ";A$:FORT=0TO24:
    PRINTCHR$(101);
725 NEXTT:PRINT:PRINT:PRINT"■■ 1 ■■■■";E$
726 PRINT:PRINT"■■ 2 ■■■■";F$
```

```
727 PRINT:PRINT"■■ 3 ■■■HUTO ";X$
728 PRINT:PRINT:PRINT"   4 ■■■H1BH"
729 PRINT:PRINT:PRINT"   5 ■■■■.K.) TUVHBB"
731 GETC$:IFC$=""THEN731
733 IFC$="5"THEN750
734 IFC$="3"THENGOSUB780
735 IFPEEK(P-1)=1THENGOTO739
737 IFC$="1"THENRV=RV+128:POKE214,5:PRINT:PRINTTAB(5)"NEGA":POKEP-1,1:
    GOTO731
739 IFC$="1"THENRV=RV-128:POKE214,5:PRINT:PRINTTAB(5)"POZI":POKEP-1,0:
    GOTO731
741 IFPEEK(P-2)=1THENGOTO745
743 IFC$="2"THENRV=RV+256:POKE214,7:PRINT:PRINTTAB(5)"2.KAR.KESZL.":N=1:
    GOTO790
745 IFC$="2"THENRV=RV-256:POKE214,7:PRINT:PRINTTAB(5)"1.KAR.KESZL.":N=0:
    GOTO790
747 IFC$="4"THENGOTO720
750 POKE646,1:PRINT"■■"H$:POKE646,0:CO=PEEK(1024+40)+RV:
    IFCH<>1THENGOSUB3380
760 IFCH=1THENRETURN
770 POKE214,20:PRINT:PRINT"OK  ":PRINT"          ":GOTO670
780 IFPEEK(P)=0THENGOTO782
781 HU$="":POKE214,9:PRINT:PRINTTAB(5)"■■■■■K1":POKEP,0:GOTO731
782 HU$="H":POKE214,9:PRINT:PRINTTAB(5)"■■■■■BE":POKEP,1:GOTO731
785 RV=0:X$="K1":E$="POZITIV":F$="1.KAR.KESZL.":IFPEEK(P)=1THENX$="BE"
786 IFPEEK(P-1)=1THENE$="NEGATIV":RV=RV+128
787 IFPEEK(P-2)=1THENF$="2.KAR.KESZL.":RV=RV+256
788 RETURN
790 POKEP-2,N:GOTO731
791 :
800 MD=189:REM *** HTTEKINTES ***
810 POKE53272,MD:POKE56576,150:POKE648,108:PRINT"■":HQ=55296:FORH=0TO255
820 POKE27648+2*H,H:POKEHQ+2*H,14:IFCH<>1THENPOKE27648+H+600,H:
    POKEHQ+H+600,3
829 NEXTH:IFCH=1THENRETURN
830 GETA$:IFA$=""THEN830
840 POKE53272,21:POKE56576,151:POKE648,4
841 GOTO360
1000 :
1020 REM *** USERE ***
1030 PRINT"■■■              KARAKTERUSERE
1040 PRINT"■■■ 1 ■■■■■KARAKTER MASOLASA"
1041 PRINT"■■■ 2 ■■■■■KARAKTER USEREJE"
1042 PRINT"■■■■ 3 ■■■■■MENU"
1050 GET0$:IFV$=""THEN1050
1055 IFVAL(V$)<1ORVAL(V$)>2THENPOKE51000,1:RUN
1060 IFV$="2"THENKL$="■USERE: A <-> B":GOTO1120
1100 KL$="■MASOLAS : A(H) -> A(B)"
1120 CH=1:GOSUB720:CO=CO:GOSUB720:CH=0:PRINT
1130 IFV$="2"THENGOSUB1190:GOTO1150
1140 FORV=0TO7:POKE28672+CO*8+V,PEEK(28672+CO*8+V):NEXTV
1150 POKE214,15:PRINT:PRINT"VISSZA A MENURE (I/N) ?"
1160 GETH$:IFH$=""THEN1160
1170 IFH$="I"THENGOTO360
1180 PRINT"■":GOTO1060
1190 FORV=0TO7:C1(V)=PEEK(28672+CO*8+V):C2(V)=PEEK(28672+CO*8+V):NEXTV
1200 FORV=0TO7:POKE28672+CO*8+V,C2(V):POKE28672+CO*8+V,C1(V):NEXTV:RETURN
1210 :
1230 REM *** PROGRAM VEGE ***
1240 PRINT"■H VISZONTLATHSRA...":POKE788,49:PRINT"■■■NEW!!!":END
1250 :
```

```
1270 REM *** KIMENTES ***
1280 PRINT"■■■■       AZ ON ALTAL MEGVALTOZTATOTT       "
1290 PRINT"■■■■KARAKTERKESZLETET A PROGRAM A 28672     "
1300 PRINT"DECIMALIS CIMTOL TAROLJA."
1310 PRINT:IFH1=0THENH1=112:HI1=128
1320 PRINT"A LEMEZEN KB. 17(9) KBYTE-OT FOGLAL LE."
1330 PRINT"A KARAKTERKESZLET PROGRAMTECHNIKAI"
1340 PRINT"OKUKBOL NEM TAROLHATO KAZETTAS"
1350 PRINT"EGYSEGEN.":PRINT:PRINT
1360 IFPEEK(186)<>8THENPRINT"■USHK LEMEZESETEN !":FORT=1TO999:NEXTT:
     GOTO360
1380 PRINT:PRINT"MILYEN NEVEN TAROLJUK A KARAKTER-"
1385 PRINT"KESZLETET (PARAMETER VALTOZTATAS A"
1387 PRINT"'PARAMETER' MEGADASAVAL LEHETSE-"
1390 A$="":INPUT"GES)";A$:IFA$=""THENGOTO360
1395 IFA$="PARAMETER"URIGHT$(A$,3)="PARAMETER"THENGOTO1470
1400 IFKE=0THENIFLEFT$(A$,1)<>CHR$(215)THENB$=CHR$(215)+A$:GOTO1420
1410 B$=A$
1420 PRINT"■■■■"
1430 PRINT"POKE43,0:POKE44,";H2;":POKE45,0:POKE46,";H4;"■■"
1440 PRINT"SAVE"CHR$(34)"@:"B$CHR$(34)",8■■■■"
1450 PRINT"POKE43,1:POKE44,8:POKE51000,1:GOTO10":PRINT"■"
1460 FORH=0TO7:POKE631+H,13:NEXT:POKE198,8:END
1470 POKE646,12:PRINT"■■                    PARAMETER                    "
1471 PRINTW1$:PRINT:PRINT"■ 1 ■  1/1 TAROLAS":PRINT:PRINT"■ 2 ■
     1.FEL.":PRINT
1472 PRINT"■ 3 ■  2.FEL":PRINT:PRINT"■ 4 ■  ISMERTETES'K1 "
1473 PRINT:PRINT"■ 5 ■  HIBA":PRINT:PRINT"■ 6 ■  O.K.) TOVABB"
1474 GETW$:IFW$=""THEN1474
1475 IFW$="5"THENKE=0:H2=112:H4=128:POKE53247,0:W2$="":W3$="":
     GOTO1470
1476 IFW$="2"THENH2=112:H4=120:W2$="1.FEL MASOLASA.."
1477 IFW$="3"THENH2=120:H4=128:W2$="2.FEL MASOLASA.."
1478 IFW$="4"THENPOKE53247,1:W3$="ISMERTETES K1"
1479 IFW$="1"THENH2=112:H4=128:W2$="1/1 MASOLAS......"
1480 IFW$="6"THENGOTO1280
1481 PRINT"■■";W2$;"■■■";W3$
1482 W1$="■■■■":GOTO1471
1483 :
1490 REM *** A KARAKTERKESZLET BETOLTESE ***
1500 PRINT"■■■■   A KARAKTERKESZLET BETOLTESE            "
1520 PRINT"■■■■■■■HZ EGYSZER MAR LEMEZEN TAROLT KARAKTER-":
1530 PRINT"KESZLET BETOLTESE.":PRINT:PRINT"■('$' = DIRECTORY)"
1540 A$="":PRINT:PRINT:PRINT:INPUT"A KARAKTERKESZLET NEVE";A$
1550 IFA$=""THENGOTO360
1560 IFA$="$"ANDPEEK(186)=8THENGOSUB1640:PRINT:GOTO1540
1570 IFPEEK(53247)=0THENIFLEFT$(A$,1)<>CHR$(215)THENB$=CHR$(215)+A$:
     GOTO1600
1590 B$=A$
1600 IFPEEK(186)=1THENPRINT"■USHK LEMEZEGYSEGRE !  ":FORT=0TO999:NEXTT:
     GOTO360
1610 FORH=0TO8:POKE631+H,13:NEXT:PRINT"■■■■"
1620 PRINT"■■■■":PRINT"LOAD"CHR$(34)B$CHR$(34)",8,1■■■■■■■■"
1630 PRINT"POKE51000,1:GOTO10":PRINT"■":POKE198,8:END
1640 PRINT"■ 0=KARAKTERKESZLET"
1650 PRINT:OPEN1,8,15,"10":OPEN2,8,2,"#":Q=18:W=1
1660 PRINT#1,"B-R";2;0;Q;W:PRINT#1,"B-P";2;0
1670 GET#2,X$:IFX$=""THENX$=CHR$(0)
1680 Q=ASC(X$)
1690 GET#2,X$:IFX$=""THENX$=CHR$(0)
```

```
1700 W=ASC(X$)
1710 FORX=0TO7:PRINT#1,"B-P";2;X*32+5:Y$="":FORY=0TO15
1720 GET#2,X$:IFX$=""THENX$=CHR$(0)
1730 IFASC(X$)=160THEN1750
1740 Y$=Y$+X$:NEXTY
1750 PRINTY$:NEXTX:IFWK<>0THEN1660
1760 CLOSE1:CLOSE2:RETURN
1780 :
1790 REM *** MHS KARAKTERRE ***
1810 KL$="          KARAKTERSZERKESZTO                "
1840 CH=1:GOSUB720:CH=0
1850 RETURN
1950 :
1960 REM *** KEPERNYO-INICIALIZALAS ***
1970 PRINT"          KARAKTERSZERKESZTO                "
1980 RV$="K1":P=53246:IFPEEK(P-1)THENRV$="BE"
1981 ZS$="1,KAR.KESZL.":IFPEEK(P-2)=1THENZS$="2.KAR.KESZL."
1982 PRINT"KARAKTER:";H$;"INVERZ ";RV$;"";ZS$;"":
     IFPEEK(P)=1THENPRINT"A"
2010 PRINT"7654321076543210"
2020 PRINT"
2030 FORK=0TO7:PRINT"K" HHHHHHHHH"K" HHHHHHHHH"K:NEXTK
2040 PRINT"
2050 PRINT"7654321076543210"
2060 PRINT"  USER MATRIXOOO"
2070 IFFL=7THENRETURN
2080 PRINT" F1  F3  F5  F7  F2  "
2100 PRINT" TESZT HDAT ROM  RAM  ROTACIO"
2140 :
2150 REM *** MOZGATO RUTIN ***
2160 POKE959,144:POKE961,3:SYS957:POKEU+21,5:POKE2042,11
2170 SH=73:SV=106:BV=0:BH=0:REM SPRITE-POSITION
2180 GETH$:POKE198,0:JO=PEEK(56320):IFH$=""THENHS=0:GOTO2190
2185 HS=ASC(H$)
2190 IFHS=145OR(JOHND1)=0THENPOKE911,4:GOSUB2450:REM KURZOR FEL
2200 IFHS=17OR(JOHND2)=0THENPOKE911,4:GOSUB2480:REM KURZOR LE
2210 IFHS=29OR(JOHND8)=0THENPOKE911,4:GOSUB2510:REM KURZOR JOBBRA
2220 IFHS=157OR(JOHND4)=0THENPOKE911,4:GOSUB2540:REM KURZOR BALRA
2230 IFHS=19THENBH=0:BV=0:SH=73:SV=106
2240 IFHS=3THENPOKE959,52:POKE961,234:SYS957:POKEU+21,0:GOTO360
2250 IFHS=133THENGOSUB2820: REM F1
2260 IFHS=134THENGOSUB2570: REM F3
2270 IFHS=137THENGOSUB3700: REM F2
2280 IFHS=135THENGOSUB2780: REM F5
2290 IFHS=139THENF2=12:FL=1:GOSUB3210:F2=0:FL=0:GOTO2180:REM F6
2300 IFHS=140THENF2=12:GOSUB3210:F2=0:GOTO2180:REM F8
2310 IFHS=136THENGOSUB2800: REM F7
2320 IFHS=147THENGOSUB2760: REM CLEAR
2330 IFHS=138THENGOSUB3060: REM F4
2340 IFHS=32THENGOSUB3000: REM TORLES (SPACE)
2350 IFHS=13THENGOSUB2970: REM RETURN
2370 IFHS<128ANDHS>32OR(JOHND16)=0THENFL=1:GOSUB3020:FL=0
2380 POKEU+4,SH:POKEU+5,SV:GOTO2180
2450 REM *** FEL ***
2460 IFSV>106THENSV=SV-8:BV=BV-40
2470 RETURN
```

```
2480 REM *** LE ***
2490 IFSV<106+7*8THENSV=SV+8:BV=BV+40
2500 RETURN
2510 REM *** JOBBRH ***
2520 IFSH<73+7*8THENSH=SH+8:BH=BH+1
2530 RETURN
2540 REM *** BHLRH ***
2550 IFSH>73THENSH=SH-8:BH=BH-1
2560 RETURN
2570 REM *** HIHTUK KIIRHSH (F1) ***
2580 FORVV=0TO7:K(VV)=0:NEXT:GOSUB3130
2590 POKE646,12:PRINT"□■                VALTOZTATAS                    "
2600 PRINT:PRINT:PRINT:PRINT"■ 1 ■  HIHTUK KIIRHSH":PRINT
2610 PRINT"■ 2 ■  KARAKTER VALTOZTHTHSA":PRINT:PRINT"■ 3 ■  KHRAKTER
     THROLHSH"
2620 PRINT:PRINT:PRINT"■ 4 ■  MENU":POKE198,0
2621 PRINT:PRINT"■ 5 ■  EDITOR"
2630 GETS$:IFS$=""THEN2630
2633 IFS$="5"THENGOTO530
2640 IFS$="4"THENPOKE51000,1:RUN
2650 IFS$="1"THEN2700
2655 IFS$="2"THENKL$="■UJ KARAKTER BEVITELE":CH=1:GOSUB720:CH=0:GOTO2590
2660 IFS$="3"THENGOSUB3380:CLR:D=53248:GOTO2590
2700 PRINT"□■■■          HIHTKIIRHS                    "
2710 POKE214,4:PRINT:PRINT"■H KARAKTER HIHTHI :":PRINT
2720 FORVV=0TO7:PRINTVVTAB(12)K(VV)"=":NEXT:FL=12:F4=5:F3=1:GOSUB3210:
     FL=0:F3=0
2722 PRINT:POKE646,6:PRINT"NYOMJUN LE EGY BILLENTYUT !"
2730 GETF$:IFF$=""THEN2730
2731 GOTO2590
2760 REM *** USER CLEHR (CLEHR-TASTE) ***
2770 C2=CO:CO=32:F2=12:FL=1:GOSUB3210:F2=0:CO=C2:FL=0:RETURN
2780 REM *** MATRIX JOBBRH (F5) ***
2790 FL=1:GOTO3210
2800 REM *** USED JOBBRH (F7) ***
2810 FL=0:GOTO3210
2820 REM *** H KARAKTER TESZTELESE (F3) ***
2830 H=0:FORVV=0TO7:K(VV)=0:NEXT:GOSUB3130
2840 POKE199,1:POKE646,12:PRINT"□                    TEST                    "
2850 POKEU+21,3:POKEU,100:POKEU+1,100
2860 POKE2040,13:POKE2041,13:POKEU+39,6:POKEU+40,6
2870 POKEU+2,192:POKEU+3,94:POKEU+29,2:POKEU+23,2
2880 FORVV=0TO7:POKE832+ZH,K(VV):ZH=ZH+3:NEXT:ZH=0
2890 POKE214,7:PRINT:PRINTTHB(8)"1":"THB(20)"1:2"
2891 POKE214,15:PRINT:PRINT"THROLJUK H KARAKTERT ?■■■■(I/N)"
2900 GETA$:IFA$=""THEN2900
2910 IFA$<>"I"THENPOKEU+21,0:GOSUB1970:FL=12:F2=12:GOSUB3210:
     FL=0:F2=0:GOTO2150
2920 POKEU+21,0:GOSUB3380
2930 CLR:D=53248:PRINT"VISSZA A MENURE (I/N) ?"
2940 GETA$:IFA$=""THEN2940
2950 IFA$="J"THENGOTO360
2960 GOTO530
2970 REM *** RETURN BILLENTYU ***
2980 IFSV<106+7*8THENSH=73:SV=SV+8:BH=0:BV=BV+40:RETURN
2990 SH=73:BH=0:RETURN
3000 REM *** SPACE BILLENTYU (TORLES) ***
3010 POKE1024+6+7*40+BV+BH,32:RETURN
```

```
3020 REM *** PUNTBEVITELTES ***
3030 POKE1024+6+7*40+BV+BH,81:POKE55296+6+7*40+BV+BH,3:RETURN
3050 :
3060 REM *** A KARAKTER INVERTALASA ***
3070 POKE0+41,14
3080 FORK=0TO7:FORKK=0TO7:PE=PEEK(1024+286+KK+40*K):IFPE=32THENPE=81:
     GOTO3100
3090 IFPE=81THENPE=32
3100 IQ=286+KK+40*K:POKEIQ+1024,PE:POKEIQ+55296,7:NEXTKK:NEXTK:POKE0+41,3
3110 RETURN
3120 :
3130 REM *** A DATAK SZAMLALASA ***
3140 POKE959,52:POKE961,234:SYS957:POKE0+21,0:H=0:
     REM U-INTERRUPT & SPRITE OFF
3150 FORVV=0TO7:FORV=0TO7
3160 A(V)=PEEK(1024+H+293-V):M=2^V:IFA(V)=81THENK(VV)=K(VV)+M
3180 NEXTV:PRINTK(VV)"■";:V=0:H=H+40:NEXTVV:H=0:RETURN
3200 :
3210 REM *** A KARAKTER NAGYITASA ***
3220 POKE0+41,14:JJ=0:DD=293:HK=32:IFF3=1THENHK=87
3230 IFF2=1ZTHENDD=DD-13
3240 IFFL=1THENCG=13*4096:FA=5:GOTO3260
3250 CG=28672:FA=13
3260 IFF4=0THENF4=FA
3270 FORI=0TO7:REM KARAKTERSZAMLALO
3280 IFFL=1THENPOKE56334,0:POKE1,51:REM A KARAKTERGENERATOR KIOLVASASA
3290 IFFL=1ZTHENZZ=K(I):GOTO3310
3300 ZZ=PEEK(CG+8*CO+I)
3310 IFFL=1THENPOKE1,55:POKE56334,1
3311 POKE1028+6*40+JJ,32:POKE55300+7*40+JJ,FA:POKE1028+7*40+JJ,31
3320 FORJ=0TO7:AK=ZZANDZTJ:GETST$
3322 IFST$<>""THENIFASC(ST$)=3THENPOKE1028+7*40+JJ,32:POKE0+41,3:RETURN
3330 IFAKTHENPOKE1024+13+DD-J+JJ,81:POKE55296+13+DD-J+JJ,F4:GOTO3350
3340 POKE1024+13+DD-J+JJ,AK:POKE55296+13+DD-J+JJ,FA
3350 NEXT:JJ=JJ+40:NEXTI:FL=0:F4=0:POKE0+41,3:POKE1028+6*40+JJ,32:RETURN
3370 :
3380 REM *** A KARAKTER TAROLASA ***
3382 AU=PEEK(53246):FORVV=0TO7:POKE28672+8*CO+VV,K(VV)
3383 IFCO>383ORCO>127ANDCO<256THENKV=1
3384 IFAU=1THENIFRV=1THENPOKE28672+(8*(CO-128))+VV,255-K(VV):GOTO3389
3386 IFAU=1THENPOKE28672+(8*(CO+128))+VV,255-K(VV)
3389 NEXT:RETURN
3390 :
3391 REM *** PROGRAMGENERATOR ***
3392 PRINT"□KIS TURELMET!":WK=159:POKE53000,0:POKE53001,0
3393 DIMWH(255):DIMWI(255):FORA=0TO255:WH(A)=-1:WI(A)=-1:NEXT:MD=189
3394 CH=1:GOSUB810:CH=0
3396 POKE27648+512,WK:POKE55296+512,13:POKE27648+514,141:
     POKE55296+514,2
3397 POKE27648+516,144:POKE55296+516,2:GOTO4000
3399 XY=PEEK(53000):YX=PEEK(53001):IFCO>255THENGOTO3405
3400 FORG=0TO7:K(G)=PEEK(28672+8*CO+G):NEXTG:IFWH(CO)>-1THEN3402
3401 WH(CO)=20000+XY*9:XY=XY+1:POKE53000,XY:IFXY+YX>255THENGOTO3409
3402 POKEWH(CO),CO:FORYO=0TO7:POKEWH(CO)+1+YO,K(YO):NEXTYO:RETURN
3405 FORG=0TO7:K(G)=PEEK(28672+8*CO+G):NEXTG:KO=CO-256:IFWI(KO)>1THEN3407
3406 WI(KO)=22049+YX*9:YX=YX+1:POKE53001,YX:IFXY+YX>255THENGOTO3409
3407 POKEWI(KO),KO:FORI=0TO7:POKEWI(KO)+1+I,K(I):NEXTI:RETURN
3409 POKE53002,0:POKE53100,PEEK(53000):POKE53200,PEEK(53001)
```

```
3410 ZK=PEEK(53000):ZK=ZK-1:IFZK<0THEN3460
3420 POKE53000,ZK:CO=PEEK(20000+9*ZK):FORI=0TO7:
     K(I)=PEEK(20000+9*ZK+1+I):NEXTI
3430 PRINT"■■■■":ZC=PEEK(53002):POKE53002,ZC+1
3440 PRINTZC+370"DATA"CO"■■,";:FORI=0TO6:PRINTK(I)"■■,";:NEXTI:PRINTK(7)
3450 PRINT"GOTO3410":FORM=0TO9:POKE631+M,13:NEXTM:POKE198,10:PRINT"■":END
3460 ZL=PEEK(53001):ZL=ZL-1:IFZL<0THEN3510
3470 POKE53001,ZL:CO=PEEK(22049+9*ZL):FORI=0TO7:K(I)=PEEK(22049+9*ZL+1+I):
     NEXTI
3480 PRINT"■■■■":ZC=PEEK(53002):POKE53002,ZC+1
3490 PRINTZC+370"DATA"CO+256"■■,";:FORI=0TO6:PRINTK(I)"■■,";:NEXT:
     PRINTK(7)
3500 PRINT"GOTO3460":GOTO3525
3510 ZC=PEEK(53002)+370:PRINT"■■■■"ZC+1"DATA-1":PRINTZC+2":::"
3511 PRINT"340 READA:IFA=-1THEN360"
3520 PRINT"350 FORK=0TO7:READB:POKE2872+A*8+K,B:NEXTK:GOTO340"
3521 PRINT"360 POKE53272,189:POKE56576,150:POKE648,108:?CHR$(147)"
3522 PRINT"GOTO3530"
3525 FORM=0TO9:POKE631+M,13:NEXTM:POKE198,10:PRINT"■":END
3530 POKE53002,4:POKE53010,0
3540 HH=PEEK(53002):IFHH=0THENGOTO3620
3550 POKE53002,HH-1
3560 PRINT"■■■■":ZD=PEEK(53010):FORA=0TO7:PRINTZD+10*A:NEXTA:
     POKE53010,ZD+6*10
3570 PRINT"GOTO3540":GOTO3525
3620 KK=PEEK(43)+256*PEEK(44)+500+(PEEK(53100)+PEEK(53200)*10)
3622 FORJ=KKTOPEEK(45)+256*PEEK(46)
3630 IFPEEK(J)=58ANDPEEK(J+1)=58ANDPEEK(J+2)=58THENGOTO3650
3640 NEXTJ
3650 VH=INT((J/256)+1)
3655 PRINT"■■■■":PRINT"POKEJ-3,0:POKEJ-4,0:POKE45,0:POKE46,"VH":
     CLR":GOTO3525
3699 :
3700 REM *** ROTHC1U ***
3710 INPUT"H JUBBRH IOLHSUK SZAMH (1/2/3) ";H$
3720 POKE781,23:SYS59903:PRINT"J":FORH=0TO7:K(H)=0:W(H)=0:NEXTH
3730 D=VAL(H$):IFD=0THENGOTO2150
3731 GOSUB3130:POKE781,24:SYS59903:IFD=<IORD>3THEND=1
3732 POKEU+41,14:POKE959,144:POKE961,3:SYS957:FORC=1TOD:FORH=0TO7:
     W(H)=0:NEXTH
3825 FORB=0TO7:FORH=7TO0STEP-1:M=2^H:IFK(B)-M>=0THENW(H)=W(H)+2^(B):
     K(B)=K(B)-M
3830 NEXTH:NEXTB:FORH=0TO7:K(H)=W(H):NEXTH:FORH=0TO7:W(H)=K(H):NEXTH
3840 FORH=0TO7:K(H)=W(7-H):NEXTH:NEXTC
3850 F2=12:FL=12:GOSUB3210:PRINT"II":F2=0:FL=0:POKEU+41,3:GOTO2150
4000 FR=14:LE=0:R=0:IFKK=255THENPRINTCHR$(14)
4001 GETC$:JO=PEEK(56320):IFC$=""THENC$=CHR$(0)
4002 C=ASC(C$):IFC=157OR(JOANDU4)=0THEN1FLE>0THENR=-2:GOSUB4050
4004 IFC=29OR(JOANDU8)=0THEN1FLE<516THENR=2:GOSUB4050
4006 IFC=13OR(JOANDU16)=0THENCO=PEEK(27648+LE)+KR:FR=7:R=0:GOSUB4050:
     GOSUB4060
4007 IFFL=3THENPOKE55296+LE,1
4008 IFFL=6THENFL=0:POKE55296+LE,FR
4009 FL=FL+1
4010 IFC=145OR(JOANDU1)=0THEN1FLE>=40THENR=-40:GOSUB4050
4015 IFC=17OR(JOANDU2)=0THEN1FLE<=11*40+36THENR=+40:GOSUB4050
4030 GOTO4001
4050 FL=3:POKE55296+LE,FR:LE=LE+R:FR=PEEK(55296+LE):RETURN
4060 IFLE=512THENIFPEEK(53272)<>191THENKR=256:WK=32:MD=191:GOTO3894
```

```
4062 IFLE=514THENPRINTCHR$(142):POKE53272,21:POKE56576,151:
     POKE648,4:POKE51000,
:RUN
4064 IFLE=516THENGOSUB4070:GOTO3409
4065 GOSUB3399:RETURN
4070 POKE53272,21:POKE56576,151:POKE648,4
4080 PRINT"■":POKE214,10:PRINT:PRINT"A SZERKESZTO TORLESE...!"
4081 PRINT"BIZTOS BENNE (I/N) ??!"
4090 GETK$:IFK$=""THEN4090
4095 IFK$="I"THENRETURN
4096 GOTO 360
```

A program önmagát magyarázó menütechnikával íródott. Csak néhány adalék a megértéséhez:

Az AUTO utasítás gondoskodik arról, hogy a bevitt karakter mellett annak inverz változata is létrejöjjön. Ezzel jelentős idő takarítható meg.

A PROGRAMGENERATOR hatására kilistázódik az első karakterkészlet. A billentyűzet vagy joystick segítségével kiválaszthatjuk azokat a karaktereket, amelyek átkerüljenek a BASIC-betöltőbe.

Végül nyomjuk le a RETURN-t. A bevitelt a program a megjelölt karakter színváltozásával nyugtázza.

Alul jobb oldalt három művelet választható:

1. "balra nyíl": kilistázza a 2. karakterkészletet. Ebből is vehetünk át jeleket.
2. inverz M: hibás bevitelnél visszatérhetünk a menüre
3. inverz P: működteti a programgenerátort, amely létrehozza a BASIC-betöltőt.

Az új karakterkészlet a BASIC terület végén helyezkedik el.

5.7 A formatervezett lista

A karakterek megváltoztatásával kényelmes és áttekinthető módjára nyújt lehetőséget a következő, viszonylag rövid lista. Különlegessége, hogy a karaktert nem egy hagyományos BASIC-betöltőben tárolja, hanem az alakját magában a listában helyezi el.

Karaktermeghatározáskor a program ezt az alakot veszi elő, és számítja ki belőle a byte-értékeket.

Nem hallgathatjuk el, hogy a karaktermeghatározás ilyen módja lassabb, mint a BASIC-betöltő, mivel a byte-ok értékét előbb ki kell számítani. De vannak előnyei is: lényegesen áttekinthetőbb, és hibázni szinte lehetetlen, mivel a karakter eredeti formájában jelenik meg.

```
0 REM **** P/5. ****
1 :
2 :
5 PRINT"■■■■■■■■■  KEREM VARJON! A BEIRAS KB. 1 PERCIG  TART !"
10 POKE56334,0:POKE1,51
15 FORK=0TO4095
20 POKE57344+K,PEEK(53248+K)
25 NEXT
```

```
30 POKE1,55:POKE56334,1
50 POKE53272,PEEK(53272)AND(NOT8OR8
60 POKE56576,196:POKE648,8*64+4:PRINT"J"
100 READA$:IFA$<>"DESIGN"THENGOTO100
110 READB$:B$=LEFT$(B$,1)
120 PRINT"@";B$
130 CO=PEEK(50176)
140 FORK=0TO7:READC$
141 IFC$="VEGE"THENPRINT"? HIBAS ADAT A ";PEEK(63)+256*PEEK(64):
    SORBAN":END
150 IFLEN(C$)<>8THENPRINT"?SOK JEL VAN A ";PEEK(63)+256*PEEK(64).
    SORBAN":END
160 FORJ=7TO0STEP-1:K$=MID$(C$,8-J,1):IFK$="*"THENBY=BY+2^J
170 NEXTJ:POKE57344+8*CO+K,BY:BY=0:NEXTK
180 READA$:IFA$="DESIGN"THENGOTO110
200 DATA DESIGN,A
210 DATA*********
220 DATA*.......*
230 DATA*.......*
240 DATA*.......*
250 DATA*.......*
260 DATA*.......*
270 DATA*.......*
280 DATA*********
290 DATA DESIGN,B
300 DATA         *******.
310 DATA         *......*
320 DATA         *......*
330 DATA         *.....*.
340 DATA         ******..
350 DATA         *......*
360 DATA         *......*
370 DATA         *******.
390 DATA VEGE
```

Ha áttanulmányozzuk a listát, akkor láthatjuk, hogy az első DATA sor elején egy DESIGN utasítás áll, amelyet a megváltoztatni kívánt karakter követ. Végül jön a 8 DATA sor, amelyek a karakter alakját meghatározzák. A csillagok jelentik a beültetett pontokat.

5.8 A többszínű üzemmód

Bizonyára mindannyian jól ismerjük a sprite-ok többszínű megjelenítésének lehetőségét, az MC üzemmódot: a vízszintes felbontás a felére csökken. A karakter megjelenítéséhez egyidejűleg három színt használhatunk. Ez az általunk meghatározott karakterekre is érvényes.
Jelentős szerepet tölt be ebben a folyamatban a VIC 22-es regisztere (2-es vezérlőregiszter). Ez a regiszter (pontosabban a regiszter 4. bitje) gondoskodik a karakterek felépítéséről a többszínű üzemmódban.

A többszínű üzemmódot a következő sorral kapcsolhatjuk be:

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

Alapállapotba a

```
POKE 53270,PEEK(53270) AND NOT 16
```

utasítással térhetünk vissza.
A többszínű üzemmódnak van egy érdekes tulajdonsága: alapállapotban a karakterek megjelenítésére 16 szín áll rendelkezésünkre. Többszínű üzemmódban ez a lehetőség 8-ra csökken (a fekete és a sárga közötti alapszínek). Ha nem ezekre a színekre hivatkozunk a többszínű üzemmódban, akkor a karakter a képernyőn felismerhetetlenné válhat. Sarkossabbá válik, és néhány színkombináció esetén még árnyéka is lehet.
Többszínű üzemmódban egy megjelenített pont többé nem csupán egy pont a képernyőn. Két egymás melletti pont egy párt alkot.
Ebből négy kombinációs lehetőség adódik:

00 = 0-ás szín 0-ás háttérszín regiszter 53281-es cím
01 = 1-es szín 1-es háttérszín regiszter 53282-es cím
10 = 2-es szín 2-es háttérszín regiszter 53283-as cím
11 = szín-RAM Karakterszín 646-os cím

Az imént említettük, hogy többszínű üzemmódban csak 8 szín használható. Nos, ez nem teljesen igaz.
Pótlólag még megadhatunk 8 színt. Ehhez a szín-RAM-ból 8 különböző szín áll rendelkezésre.
A többi színregiszter ugyanúgy 16 színt vehet fel, mint eddig.
A következőkben az átdolgozott listát láthatjuk. Programunkkal kibővítettük a DESIGN utasítást: ezen túl a DESIGN-MC utasítást is használhatjuk, amelynek segítségével többszínű karaktereket is tervezhetünk. Ilyenre mutat példát a következő program:
Jelölések: A szín-RAM
A 2-es színregiszter
Az 1-es színregiszter

```
0 REM  **** P76.  ****
1 :
2 :
300 DESIGN-MC,B
310 DATA 1111
320 DATA 2..2
330 DATA 2..2
340 DATA 3..3
350 DATA 3..3
360 DATA 1111
370 DATA VEGE

READY.
```

A kibővített lista:

```
0 REM **** P77. ****
1 :
2 :
5 PRINT"■■■■■ KEREM VARJON KB. 1 PERCET !":POKE646,6
10 POKE56334,0:POKE1,51
15 FORK=0TO4095
20 POKE57344+K,PEEK(53248+K)
25 NEXT
30 POKE1,55:POKE56334,1
50 POKE53272,PEEK(53272)AND(NOT60)OR8
60 POKE56576,196:POKE648,3*64+4:PRINT"■"
100 READA$:IFLEFT$(A$,6)<>"DESIGN"THENGOTO100
101 LL=8:MM=7
102 IFA$="DESIGN-MC"THENMC=1:LL=4:MM=3
110 READB$:B$=LEFT$(B$,1)
120 PRINT"■";B$
130 CC=PEEK(50176)
140 FORK=0TO7:READC$
141 IFC$="VEGE"THENPRINT"? HIBAS ADAT A ";PEEK(63)+256*PEEK(64);
    ". SORBAN":END
150 IFLEN(C$)<>LLTHENPRINT"?SOK JEL VAN A ";PEEK(63)+256*PEEK(64);
    ".SORBAN":END
160 FORJ=MMTO0STEP-1:K$=MID$(C$,LL-J,1):IFK$="*"THENBY=BY+2↑J
161 IFMC=0THENGOTO170
162 IFK$="1"THENBY=BY+2↑(2*J):BY=BY+2↑(2*J+1)
163 IFK$="2"THENBY=BY+2↑(2*J)
164 IFK$="3"THENBY=BY+2↑(2*J+1)
170 NEXTJ:POKE57344+8*CC+K,BY:BY=0:NEXTK
180 READA$:IFLEFT$(A$,6)="DESIGN"THENGOTO101
190 POKE 53270,PEEK(53270)OR16:POKE646,8:PRINT"■■■■":REM MC BE
200 DATA DESIGN,A
210 DATA*********
220 DATA*.......*
230 DATA*.......*
240 DATA*.......*
250 DATA*.......*
260 DATA*.......*
270 DATA*.......*
280 DATA*********
290 DATA DESIGN-MC,B
300 DATA          1111
310 DATA          2222
320 DATA          3333
330 DATA          ....
340 DATA          3333
350 DATA          2222
360 DATA          1111
370 DATA          ....
390 DATA VEGE
```

5.9 A feliratnyomtatás egy könnyű módszere

A következő program egy tetszőleges hosszú felirat listázását teszi lehetővé. Az elve nagyon egyszerű: meg kell adni a kiírandó szöveget, majd meg kell választani a nagyítás mértékét (a szélesség tetszőleges, a magasság max. 10 karakter).
Ezután a program karakterről karakterre beolvassa a szöveget. A megjelenítési formát kiolvassa a ROM-ból, és a megadott paraméterekkel felnagyítja.
Végül a program kiírja a karaktert a nyomtatóra. Ezzel a módszerrel tetszőleges hosszúságú szöveget nyomtathatunk.

A program listája önmagáért beszél:

```
0 REM ***** P78. *****
1 :
2 :
5 PRINT"□":POKE53280,14:POKE53281,14:POKE646,1
10 REM *** FELIRATNYOMTATO ***
15 PRINT"■       FELIRATNYOMTATO          ■■■■■"
16 PRINT"KAPCSOLJA BE A NYOMTATOT !■■■■■"
20 OPEN4,4
21 INPUT"SZELESSEG";BR
22 INPUT"MAGASSAG";HO
23 PRINT"□"
25 POKE214,21:POKE211,VV:SYS58732:POKE204,0
26 GETT$:IFT$=""THEN26
27 IFPEEK(207)THEN 27
28 POKE204,1:IFT$=CHR$(133)THENPRINT"□U.K.":END
29 K$=K$+T$:IFLEN(K$)>39THENK$=RIGHT$(K$,(LEN(K$)-1))
30 POKE214,20:PRINT:PRINTK$:IFVV<40THENVV=VV+1
31 TX=0:IFPEEK(53272)=23THENTX=256
51 POKE646,14:PRINT"■";T$:CO=PEEK(1024):POKE646,1
60 REM *** A KARAKTER KIOLVASASA ***
70 POKE56334,0:POKE1,51
80 FORA=0TO7
90 CH(A)=PEEK(53248+(8*CO)+A+(TX*8))
100 NEXTA
110 POKE1,55:POKE56334,1
120 REM *** A KARAKTER MEGVALTOZTATASA ***
130 FORA=0TO7:K(A)=0:NEXT
140 FORA=0TO7
150 FORB=7TO0STEP-1:W=2↑B:IFCH(A)>=WTHENCH(A)=CH(A)-W:
    K(7-B)=K(7-B)+2↑A
160 NEXTB,A
200 REM *** A KARAKTER KIIRASA ***
210 FORI=0TO7:Q$="":FORJ=7TO0STEP-1:WI=K(I)AND2↑J
220 IFWITHENFORU=1TOHO:Q$=Q$+"■":NEXTU:GOTO250
230 FORU=1TOHO:Q$=Q$+" ":NEXTU
250 NEXTJ:XY$="":FORU=1TOBR:XY$=XY$+Q$:NEXTU
255 LX=LEN(XY$)-1:IF RIGHT$(XY$,1)=" "THENXY$=LEFT$(XY$,LX):
    GOTO255
260 PRINT#4,XY$:NEXTI:GOTO25
270 END
```

5.10 Felnagyított fotófelirat

A következő program egy, a TX$ füzérben tárolt szöveg futó feliratként való megjelenítését teszi lehetővé. A program különlegessége, hogy a megjelenő karakterek nyolcszoros nagyításúak.

```
0 REM **** P79. ****
1 :
2 :
5 POKE53280,1:POKE53281,1:POKE646,6:PRINT"□"
10 INPUT"SZOVEG";TX$:TX$=TX$+"      "
20 PRINT"□":FORZ=1TOLEN(TX$)
30 A$=MID$(TX$,Z,1)
40 GOSUB1000:NEXT Z
50 END
60 :
70 :
80 :
90 :
100 :
1000 POKE646,1:PRINT"■";A$:CO=PEEK(1024):POKE646,6
1010 POKE56334,0:POKE1,51
1020 FORJ=0TO7
1030 CO(J)=PEEK(53248+(8*CO)+J)
1040 NEXTJ
1050 POKE1,55:POKE56334,1
1060 FORX=7TO0STEP-1
1070 FORY=0TO7
1080 A=CO(Y)AND2↑X:IFATHENW$(Y)=W$(Y)+"O":GOTO2000
1090 W$(Y)=W$(Y)+" "
2000 NEXTY:P=P+1
4000 REM
4010 PRINT"■";:FORW=0TO7
4020 IFQ>37THENQ=38
4021 PRINTTAB(38-Q)W$(W)
4030 NEXTW:Q=Q+1
4040 IFP<39THEN4100
4050 FORT=0TO7
4060 LE=LEN(W$(T))-1:W$(T)=RIGHT$(W$(T),LE)
4070 NEXTT
4100 NEXTX:RETURN

READY.
```

BASIC-ben ez a program nagyon lassú. A lista csak a szerkesztés elvét igyekszik bemutatni. Próbáljuk megírni egyszer a gépi kódú változatát!

5.11 Blokkok a sprite-ok számára

A sprite-ok programozása során a legkellemetlenebb feladat a sprite-ok felépítését tartalmazó blokkok elhelyezése a gép memóriájában. Erre a célra csak az alábbi négy blokk vehető igénybe.

11-es blokk 704–766-os cím
13-as blokk 832–894-es cím
14-es blokk 896–958-as cím
15-ös blokk 960–1022-es cím
Az is nehezíti a helyzetet, hogy a 13–15-ös blokkok a kazettapufferben helyezkednek el, amely normál esetben programfutás közben nem használatos.
Ez a négy szabad blokk csak négy különböző sprite meghatározását teszi lehetővé.
Ahhoz, hogy a képernyőn mind a nyolc, egymástól különböző sprite-ot megjeleníthessük, a további blokkok számára helyre van szükség. Ezt a BASIC terület kezdetének elállításával érhetjük el. Példánkban így nyolc blokk (32–39) áll rendelkezésünkre, és a kazettapufferre nincs szükségünk.

A BASIC startot 2048-as címről a 2560-as címre toltuk el:

POKE 44,10	(a 43-as cím tartalma változatlan marad, az új BASIC kezdőcím 10*256)
POKE 10*256,0	(a BASIC-kezdet)
NEW	(mutató-beállítás)

A NEW parancsból látható, hogy ezeket az utasításokat a program betöltése előtt kell megadni.

6. FEJEZET
JÁTÉK

Ebben a fejezetben a programozás folyamatát fogjuk bemutatni a LIFTES FIÚ című játékon.
A kezdők ebből megtanulhatják, hogyan kell egy hosszabb játékot programozni. A haladók néhány apró trükköt vehetnek át. Reméljük, hogy példánk mindenkinek segítséget nyújt további próbálkozásaihoz.

A játék rövid leírása (a leírás a programban is meg fog jelenni).

A játékos egy szállodai liftes fiú, akinek az a feladata, hogy a szobákból összeszedje a tisztításra váró cipőket. Minden párért 1 Ft borravalót kap a vendégektől. Ha 50 párnál többet sikerül a portára eljuttatni, akkor azért külön jutalom jár.
Ha a liftes fiú egyszer rosszul ugrik fel a liftre, és a liftaknába zuhan, akkor az összes cipőjét elveszti. A játék a 2-es porton keresztül vezérelhető a joystickkel. A joystick fel és le irányú működése ki van iktatva. A tűzgomb megnyomásával a lift mozgási iránya változtatható meg. Az első alkalommal nagyon nehéz a liftbe beszállni. Be kell gyakorolni a joystick használatát. A képernyő jobb oldalán egy függőleges csíkot láthatunk. Ennek rövidülése jelzi az idő múlását. A liftes fiúnak el kell jutni a portára, mielőtt a csík eltűnik. Ha ez sikerül, akkor új időt kapunk.
Ha a liftes fiú nem éri el a portát időben, akkor egy játékossal kevesebbel játszhatunk a továbbiakban.
Először építsük föl a játék vázát. Ezt bővítsük a szükséges programrészletekkel, míg el nem érünk a kész játékig.

Az első listában a grafika elég gyenge. Az lesz a legjobb, ha a feladathoz a karaktereket újraszerkesztjük.
A későbbiekben a következőt használjuk:

Karakter	Jelentés
£	szoba egy pár cipővel
↑	zárt ajtó
#	játékos
%	portás
"	lift

A bővítések sorrendje, amely egyben fontossági sorrend is:
(0. váz)
1. grafika
2. hangok
3. bevezetés
4. kezdőkép

6.1 A váz

A program váza a legfontosabb rész. Ezért hagyjunk jó nagy helyet a grafika stb. elhelyezésére.

A 100-as sorral kezdünk, hogy a kezdőképnek is maradjon hely. Ez a lista értelmes játékra még nem alkalmas, de azért indítsuk el. Így láthatjuk, honnan indul a programfejlesztés, kb. miből kapjuk a kész programot.

A játék teljes listája a fejezet végén található.

A váz listája:

```
0 REM **** P80. ****
1 :
2 :
6 PRINT"■■":POKE53281,3
7 KKK=1024:R=54272
9 X=KKK+536:L=KKK+487:LI=40:BO=23:CI=0
10 MS=10:JSZ=3
11 PRINT"■■■■■■■■■■■■■■■                LIFTES FIU"
12 IFPEEK(56320)<>111 THEN12
100 KEP$(1)=" !↑↑£££ ↑↑£££ ↑↑££↑ £££££ £££££!"
110 KEP$(2)=" !↑↑↑↑£ £££££ £↑£££ ££££↑ £££££!"
120 KEP$(3)=" !↑££££ ↑↑↑££ £££££ £££££ ££££↑!"
130 KEP$(4)=" !£££££ £££££ £↑£££ ££↑↑£ ££↑↑↑!"
140 KEP$(5)=" !↑££££ £££££ £££££ £££££ ↑↑↑↑↑!"
150 KEP$(6)=" !£££££ ££↑↑↑ ££££↑ ££££↑ ££££↑!"
160 KEP$(7)=" !£££££ £↑↑££ ££↑££ ££££↑ £££↑↑!"
170 KEP$(8)=" !£££££ ↑↑£££ £££££ ££↑££ ££↑↑↑!"
180 KEP$(9)=" !£££££ ↑↑£££ £££££ ↑↑↑↑£ £££££!"
183 PRINT"■";
184 PRINT" ■←←←←←←←←←←←←←←←←←←←←←←←←←←←←←←←"
185 PRINT" ■←←←←←←←←←←←←←←←←←←←←←←←←←←←←←←←"
187 PRINT" ■!■←←←←←← ←←←←←← ←←←←←← ←←←←←← ←←←←←←■!"
190 FORC=1TO9
195 V=INT(RND(1)*8)+1
200 PRINTKEP$(V)
205 PRINT" ■!■←←←←←← ←←←←←← ←←←←←← ←←←←←← ←←←←←←■!"
210 NEXT
215 PRINT"■ !                %                  !"
217 PRINT"■ !■←←←←←← ←←←←←← ←←←←←← ←←←←←← ←←←←←←■!"
220 PRINT"■ ←←←←←←←←←←←←←←←←←←←←←←←←←←←←←←←"
221 FORC=KKK+77TOKKK+917STEP40:POKEC,38:POKEC+R,6:NEXT
230 POKEX,35:POKEX+R,0:POKEX+1,32:POKEX+2,32:POKEX-1,32:
    POKEX-2,32
270 PRINT"        OSSZ PONTSZAM ";LEG;
280 PRINT":IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII"
1000 GOSUB3000
1003 FORC=1TOMS:NEXT
1005 POKEL,32:POKEL+6,32:POKEL+12,32:POKEL+18,32
1010 IFPEEK(L+LI)=31THENL=-L
1020 L=L+LI
1030 POKEL,34:POKEL+6,34:POKEL+12,34:POKEL+18,34
1040 IFFL=0THEN2000
1050 POKEX,32:X=X+LI
1060 IFPEEK(X)=31THENX=X+40:GOTO9000
```

```
1070 POKEX,35:POKEX+R,0
2000 BO=BO-0.025
2010 POKEINT(BO)*40+KKK+37,32:IFBO=<1THEN9000
2990 GOTO1000
3000 IFPEEK(56320)=127THENMI=0:RETURN
3010 IFPEEK(56320)=<111THEN3500
3020 IFPEEK(56320)=123THENY=-1:GOTO3050
3030 IFPEEK(56320)=119THENY=1:GOTO3050
3040 RETURN
3050 IFPEEK(X+Y)=31THENY=Y-40:IFPEEK(X+Y)=31THEN9000
3053 IFPEEK(X+Y)=28THENCI=CI+1:GOTO3100
3054 IFPEEK(X+Y)=33THENRETURN
3055 IFPEEK(X+Y)=30THENRETURN
3057 IFPEEK(X+Y)=34ORPEEK(X+Y+40)=34THEN3900
3060 IFPEEK(X+Y)=37THEN3800
3065 IFPEEK(X+Y)=32ANDPEEK(X+Y+40)=32THEN:POKEX,32:X=X+Y:GOTO9000
3070 IFPEEK(X+Y)=32ANDPEEK(X+Y+40)=102THEN3100
3100 POKEX,32
3110 X=X+Y:FL=0
3120 POKEX,35:POKEX+R,0
3130 PRINT"] ←←PONTOK←";CI+FT;:PRINT"←←CIPOK  ←";CI
3190 RETURN
3500 IFMI=0THENLI=-LI:H=1
3510 RETURN
3800 MS=MS-1:IFCI<50THENFT=FT+CI:GOTO3810
3805 FT=FT+INT(CI*(BO*7))
3810 X=KKK+536:L=KKK+487:LI=40:BO=23:CI=0
3830 GOTO183
3900 POKEX,32
3905 X=X+Y
3910 IFPEEK(X)=34THENX=X-40:IFPEEK(X)=34THEN9000
3920 FL=1
3930 POKEX,35:POKEX+R,0
3940 RETURN
9000 FT=FT+CI:JSZ=JSZ-1
9010 IFBO=<1THENGOTO9020
9015 POKEX,32:X=X+40:IFPEEK(X+40)=31THEN9100
9017 POKEX,35:POKEX+R,0:GOTO9015
9020 POKEX,32
9030 BO=23:FORC=1TO23:POKEC*40+KKK+37,160:NEXT:FL=0
9035 L=KKK+487:LI=40:BO=23:CI=0
9036 IFJSZ=0THEN9800
9040 X=KKK+536:POKEX,35:POKEX+R,0:GOTO183
9100 POKEX,39:POKEX+R,0
9190 GOTO9020
9800 PRINT"□";:POKE53281,1
9830 PRINT"QQQQQQQQQQQQQQQQQ                    ";FT+CI;"FT-OT KERESTEL"
9840 IFFT+CI>LEGTHENLEG=FT+CI
9850 FT=0:CI=0
9880 GETA$:IFA$<>""ORPEEK(56320)=111THEN6
9890 GOTO9880

READY.
```

Valójában nem teljesen igaz, hogy a program a 100-as sortól kezdődik. Az első sorokban található a név és a változók meghatározása, de azért a kezdőképnek így is hagytunk elég helyet. A változókat mindig a programok elején határozzuk meg, mivel így könnyen áttekinthetők.

A fontosabb változók:

X — A tároló egy címe, a kezdőértéke 1560. Ez kb. a képernyőtároló közepe

KKK — A képernyőtároló kezdőcíme. Megváltoztatásával a karakterkészlet is megváltoztatható. Ez a változó mutatóként működik, az értéke állandó (konstans)

R — A KKK változóhoz hasonló. A képernyőtároló és a színtároló kezdőcímei különbségét tartalmazza. Ezt a változót (konstanst) az új grafikus karakterek megváltoztatására használjuk. A szemléletesség kedvéért a színváltoztatás a karakter alatt jelenik meg a képernyőn. Ha pl. a játékos alatti színt akarjuk megváltoztatni, akkor be kell írni a következő sort: POKE X+R, (a játékos színe)

L — Az első lift címe. A további liftek az L segítségével érhetők el. A második lift az L+6, a harmadik az L+12, a negyedik az L+18-as címen található. Így csak egy változót kell módosítani, s ezzel programsorokat takarítunk meg

BONUS — A jutalompont változója. Hosszú játékokban a változónevet úgy szokták megadni, hogy az később felismerhető legyen, és utaljon a funkciójára. Ezért a célért érdemes a változónevet egy kicsit hosszabbra választani

Y — A játékos koordinátájának segédváltozója. Pl. a szomszédos mező lekérdezésénél használjuk

FT — Az elért forintösszeg

CI — A játékosnál levő cipők száma. Ha a játékos a liftaknába esik, akkor a változó értéke 0 lesz

JSZ — A még rendelkezésre álló játékosok számát adja. Baleset esetén, vagy a jutalompontok elvesztésekor JSZ értéke eggyel csökken. A változó értékének módosulása után a program ellenőrzi, hogy értéke elérte-e már a nullát

MS — Munkasebesség. A változó kezdőértéke 10, ha a játékos elérte a portát, akkor értéke 5 lesz. A kezdők növelhetik vagy csökkenthetik a változó értékét

LI — A lift irányát mutató változó. Tartalma lefelé menetben 40, felfelé menetben –40. Az L változóhoz LI hozzászámítódik. Ezekből alakul ki a lift mozgása

C — Ciklusváltozó. Praktikus minden programban ugyanazt a változót választani ciklusváltozónak

Ha egy programban mindig ugyanazt a ciklusváltozót használjuk, helyet takarítunk meg vele. Csak arra kell vigyázni, hogy a ciklus

le legyen mindig zárva, mielőtt egy másik ciklusban a változót használjuk

LEG Az eddigi legmagasabb pontszámot tartalmazó változó. Bár nem fontos változóról van szó, mégis bekerült a program vázába

MI Ez a változó akadályozza meg, hogy egy gombnyomásra a menetirány többször megváltozzon. Gombnyomásra a menetirány megváltozik, a változó értéke 1 lesz, majd ismét 0, ha a joystick nyugalomban van.
MI=1 esetén a gép nem reagál a joystick állapotváltozásaira. Hasonló módon lehet a mozgás hosszát is korlátozni, mint pl. a FROGGER játékban

FL Ha a liftes fiú a liftben van, akkor a változó értéke 1, egyébként 0

V Véletlenszám a különböző képek előállításához

KÉP$(V) A KÉP$(V) kilenc különböző színhelyet tud előállítani. A változó dimenzionálva van, és mindig egy másik összképet állít elő. Minél több különböző színhelye van egy játéknak, annál érdekesebb a játék

Egy hosszú program leírásakor nemcsak a változók listájára, hanem az egyes programlépések leírására is szükség van.

Sorszám	Feladat
0–5	Itt lesz később a karakterkészlet beolvasva (lásd GRAFIKA)
6–99	A változók meghatározására fenntartott hely. Ezenkívül itt helyezkedik el a címkép. Később itt lesz a kijavított kezdőkép is.
100–180	A KÉP$(W) meghatározása
183–280	Az összkép előállítása
1000–2990	A lift mozgásának szabályozása
1050–1070	A játékos mozgási tartománya a liftben. Ez a mozgás különbözik a játékos normál mozgásaitól
3000–3190	A játékos vezérlése. A 1050–1070 sorszámú mozgatórutinnal ellentétben itt a mozgást a joystick befolyásolja
3500–3510	Itt változik meg a lift mozgási iránya
3800–3830	A program ezeket a sorokat átugorja, ha a játékos a portán van
3900–3940	A játékos mozgatásának egy részprogramja
9000–9190	Ez a rész figyeli a játékos liftaknába esését
9800–9890	Ez a zárórutin. Ide akkor ágazik el a program, ha elfogytak a játékosok
50000–vége	Ebben a tartományban vannak a DATA sorok, amelyek a grafika adatait tartalmazzák

A hangeffektusokat tartalmazó sorok ebben a felsorolásban nem szerepelnek. Ezt a későbbiekben részletesen fogjuk tárgyalni. Az egyes programrészletek sorszámozása csak az egyes rutinok hosszát igyekszik szemléltetni. Utólag a program egy jó RENUMBER (átsorszámozó) programmal tömörebbé írható.

6.2 A grafika

A program vázának első bővítménye a grafika. Ez tartalmazza az új karakterek meghatározását és az ebből következő DATA sorokat. Hossza elég terjedelmes. A 9-essel bezárólag minden karaktert, tehát összesen 58 jelet változtattunk meg. Egészen új írásképet kaptunk, amellyel élvezetesebbé tettük a játékot.
Programunkat a következő sorokkal egészítsük ki:

```
0 REM **** P81. ****
1 :
2 :
3 READA:IFA=-1THEN5
4 POKE57344+C,A:C=C+1:GOTO3
5 POKE53272,24:POKE56576,148:POKE648,196
6 GOTO8
7 POKE53272,21:POKE56576,151:POKE648,4:END
8 PRINT"■J":POKE53281,3
9 KKK=50176:K=5120
50000 DATA127,127,81,22,20,22,56,0
50001 DATA60,60,102,102,126,126,102,0:A
50002 DATA120,124,102,124,126,103,126,0:B
50003 DATA60,126,102,96,102,126,60,0:C
50004 DATA124,126,99,99,99,126,124,0:D
50005 DATA127,126,96,124,96,126,127,0:E
50006 DATA127,126,124,96,124,120,112,0:F
50007 DATA62,121,112,115,115,63,31,0:G
50008 DATA115,115,115,127,127,115,115,0:H
50009 DATA28,28,28,28,28,28,28,0:I
50010 DATA127,127,7,7,103,63,31,0:J
50011 DATA115,119,126,126,126,119,115,0:K
50012 DATA112,112,112,112,112,126,127,0:L
50013 DATA119,127,107,107,107,99,119,0:M
50014 DATA103,115,123,127,127,119,115,0:N
50015 DATA62,127,119,119,119,127,62,0:O
50016 DATA126,127,115,127,126,112,112,0:P
50017 DATA62,127,99,111,111,127,62,0:Q
50018 DATA126,127,115,127,124,126,119,0:R
50019 DATA63,126,96,62,3,63,126,0:S
50020 DATA127,127,93,28,28,28,28,0:T
50021 DATA119,119,119,119,127,127,28,0:U
50022 DATA119,119,50,58,58,28,8,0:V
50023 DATA119,99,107,107,107,127,54,0:W
50024 DATA99,119,54,62,62,119,99,0:X
50025 DATA119,59,31,14,14,14,14,0:Y
50026 DATA127,63,7,28,112,126,127,0:Z
50027 DATA240,240,192,192,192,192,240,240:[
50028 DATA0,36,102,102,102,102,36,0:£
50029 DATA15,15,3,3,3,3,15,15:]
50030 DATA126,126,126,78,62,126,126,126:↑
50031 DATA239,239,239,0,251,251,251,0:←
50032 DATA0,0,0,0,0,0,0,0
50033 DATA255,0,255,0,255,0,255,0:!
```

```
50034 DATA255,255,255,189,60,60,60,60:"
50035 DATA0,60,60,24,255,24,60,36:#
50036 DATA101,85,87,87,85,85,85,101:$
50037 DATA24,24,24,60,60,126,255,48:%
50038 DATA127,254,127,254,127,254,127,254:&
50039 DATA0,24,24,126,126,24,24,24:'
50040 DATA24,48,96,96,96,96,48,24:(
```

A program elindítása után meg kell várni, amíg a számítógép a DATA-kat beolvassa.
A játék nem használja az összes karaktert. Ezek a DATA sorokban csak az egyöntetűség miatt szerepelnek. A módosított karakterkészletet más programokban is használhatjuk. A listában minden DATA sor végén van egy jel. Ennek a karakternek az átírását tartalmazza a sor 8 adata.
A LIFTES FIÚ grafikája csak a képernyőkódok megváltoztatását tartalmazza.
A hatékony grafikai megjelenítés egy másik módszere a sprite-ok használata. Programunkban nincsenek sprite-ok, ezért a sprite-grafikát nem részletezzük. Ugyanez vonatkozik a nagyfelbontású grafikára is.
Ha már elindítottuk a programot, és vissza akarunk térni a normál üzemmódba, akkor a

```
RUN 7
```

parancsot adjuk ki.

Ezzel kikerüljük a sok POKE utasítást. Mindezt kipróbálhatjuk, ha lenyomjuk a RUN-STOP/RESTORE-t és töröljük a képernyőt. Ha a programot ismét az új grafikus jelekkel akarjuk játszani, írjuk be:

```
RUN 5
```

Így a már egyszer beírt karakterkészlet nem íródik be még egyszer, nem kell annyit várnunk a kezdésre.
Játékunkba beépíthetünk különböző grafikai effektusokat is. Pl. a liftaknában való zuhanáskor az akna mélyén egy kereszt jelenik meg.
Később ez a rész hangeffektussal is kiegészíthető. A jelek, a keret és a háttér színeinek beállításakor ki-ki hagyatkozzon a saját fantáziájára, és válassza ki a legjobb kombinációt. Csak arra kell ügyelni, hogy a színek a fekete-fehér készülékeken is jól elkülöníthetők legyenek. A kék és a piros pl. fekete-fehérben nem különböztethető meg egymástól.

6.3 A hangok

Aki az új karaktereket sikeresen megalkotta, rátérhet a hangeffektusok tanulmányozására.
Ha játék közben valami változás történik a képernyőn, ehhez hozzárendelhetünk egy hangeffektust. Hangeffektus alatt itt egy dallamot, vagy egy egyszerű hangot értünk. Azoknak, akik már több játékprogrammal találkoztak, bizonyára nem ismeretlen ez az eljárás. Még a C 64-es felhasználói kézikönyve is tartalmaz néhány zenei rutint.
A LIFTES FIÚ-ban akkor hallhatunk hangokat, ha a liftes fiú elvesz egy pár cipőt, vagy ha beleesik a liftaknába. Ahhoz, hogy ez megvalósuljon, propgramunkat a következő sorokkal kell kiegészíteni.

```
0 REM  **** P82.  ****
1 :
2 :
250 POKE 54296,15
252 POKE 54277,31
254 POKE 54278,128
256 POKE 54272,1
260 POKE 54273,1
3053 IF PEEK(X+Y)=28 THEN POKE 54276,129:C1=C1+1:GOTO3100
3180 POKE 54276,0
9110 POKE 54277,31
9112 POKE 54278,255
9114 POKE 54272,0
9116 POKE 54273,2
9118 POKE 54276,33
9120 FOR C=1 TO 500:NEXT C:POKE 54276,0

READY.
```

A 250–260-as sorokban a számítógép a hangok megszólaltatásához szükséges értékeket helyezi el. Ha a liftes fiú elér egy pár cipőt, a hullámformát be majd ki kell kapcsolni (3053–3180-as sorok). A 9110–9120-as sorokban a liftaknába esés zenéje található. Ez egy másik hang, tehát a paramétereket újra be kell állítani. Hangeffektusokkal egy dallamot is le lehet jászani. A megszakításokkal foglalkozó fejezetben egy külön program található erre a célra.
Mialatt a hangeffektus első hangja hallható, ez a program lejátszik egy dallamot. Bővebbet a program kiírásánál találunk.

6.4 A bevezetés

A bevezetés jelentősége nyilvánvaló. A játékos a bevezetésből ismeri meg a játékszabályokat. Rendszerint a bevezetést a programok leírás formájában tartalmazzák. Ily módon egy kicsit védhetjük is programunkat.
Ahhoz, hogy a játék magától érthetődő legyen, a következő sorokat kell a programban elhelyezni:

```
0 REM  ****  PB3.  ****
1 :
2 :
25 PRINT "!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!"
26 PRINT"KERSZ SEGITSEGET ? (I/N) :TTTT"
28 GETLI$:IFLI$="N"ORPEEK(56320)=111THEN100
29 IFLI$="I"THEN42
38 GOTO28
42 PRINT"LIFTES FIU VAGY EGY NAGY HOTELBEN, ES "
43 PRINT"KI KELL HUZNUD A CIPOKET A SZUBAKBOL."
44 PRINT"MINDEN PARERT 1 FT JUTALMAT KAPSZ"
45 PRINT"A VENDEGEKTOL."
46 PRINT"HA 50 PARNAL TOBB CIPOT JUTTATSZ EL A"
47 PRINT"PORTARA, KULON JUTALMAT KAPSZ."
49 PRINT:PRINT" [£]   SZOBA EGY PAR CIPOVEL"
50 PRINT:PRINT" [↑]   ZART AJTO"
51 PRINT:PRINT" [#]   JATEKOS"
52 PRINT:PRINT" [%]   PORTAS"
53 PRINT:PRINT" [";CHR$(34);"]   LIFT"
56 IFPEEK(56320)<>111THEN56
100 END

READY.
```

Ezenkívül a 12-es sort ki kell törölni.

Mindig törekedni kell arra, hogy a program használata lehetőleg ne legyen bonyolult. A játékos hamar megértse, és ne kelljen minden játék után a szabályokat újra elolvasni. Ehhez a felvilágosítást kérő kérdésre N betűvel kell válaszolni. De az is elég, ha a joystick tűzgombját nyomjuk le.

6.5 A kezdőkép

A kezdőkép már egy kicsit tájékoztatja a játékost. Egy impozáns kezdőképpel megnyerhetjük a játékos tetszését. Célszerű a kezdő képet a játék egyik jelenetéből kiválasztani. Ajánlatos mozgó kezdőképet alkalmazni. Játékunkban mindkét jó tanácsot megfogadtuk.
Most a kezdőkép egy részletét mutatjuk be. A BASIC nyelvű részlet egy futó feliratot tartalmaz:

```
0 REM  ****  PB4.  ****
1 :
2 :
11 LF$="LIFTES FIU
27 PRINT":TT"LF$
30 FORO=0TO150:NEXT
31 LF$=RIGHT$(LF$,1)+LEFT$(LF$,LEN(LF$)-1)
39 GOTO27

READY.
```

Ezeket a sorokat is beilleszthetjük a már alakuló programunk kezdőképébe. De ez a pár sor önmagában is életképes program. Ki is lehet próbálni. A program indítása után a LIFTES FIÚ felirat fut végig a képernyőn. LF$-ban természetesen bármilyen szöveget megadhatunk. A 30-as sorban állíthatjuk be a futás sebességét. Ha ezt a sort elhagyjuk, vagy 0-ra állítjuk, akkor a felirat úgy felgyorsul, hogy egybefolyik. A futás irányát a LEFT$ ill. RIGHT$ utasítások helyes megválasztásával határozhatjuk meg.
Egészítsük ki főprogramunkat a következő sorokkal is. A kiegészített programmal mozgó kezdőképet kapunk.

```
0 REM **** P85. ****
1 :
2 :
12 PRINT:PRINTTAB(14)"!<<<<< <<<<<!"
13 PRINTTAB(14)"!TTTEE EEEEE!"
14 PRINTTAB(14)"!<<<<< <<<<<!"
15 PRINTTAB(14)"!TEEEE ETTTT!"
16 PRINTTAB(14)"!<<<<< <<<<<!"
17 PRINTTAB(14)"!EEEEE EEEEE!"
18 PRINTTAB(14)"!<<<<< <<<<<!"
19 PRINTTAB(14)"!TTEEE EEEET!"
20 PRINTTAB(14)"!<<<<< <<<<<!"
21 PRINTTAB(14)"!TTTTE EEETT!"
22 PRINTTAB(14)"!<<<<< <<<<<!"
23 PRINT:PRINTTAB(14)"!<<<<< <<<<<!"
32 PL=KKK+100+M
33 POKEPL-40,32
34 POKEPL,35:POKEPL+R,0
35 POKEPL+40,34:POKEPL+R+40,2
36 POKEPL+80,0
37 M=M+L1:IFM>400ORM<=0THENL1=-L1

READY.
```

6.6 A bővített játékprogram

Elkészült a programunk. Ha valaki nem kísérte figyelemmel a program összeállítását, az egyszerűen írja be a következő listát, és már játszhat is.
Ez a lista természetesen azokat is segíti, akik figyelemmel kísérték a program születését, mert ellenőrizhetik a munkájukat, kijavíthatják esetleges hibáikat.
A lista:

```
0 REM **** P86. ****
1 :
2 :
3 READA:IFA=-1THEN5
4 POKE57344+C,A:C=C+1:GOTO3
5 POKE53272,24:POKE56576,148:POKE648,196
6 GOTO8
7 POKE53272,21:POKE56576,151:POKE648,4:END
```

```
8 PRINT"■":POKE53281,3
9 KKK=50176:R=5120
10 X=KKK+536:L=KKK+487:L1=40:BU=23:C1=0:MS=10:JSZ=3
11 LF$="  LIFTES FIU                               "
12 PRINT:PRINTTAB(14)"!←←←←← ←←←←←!"
13 PRINTTAB(14)"!↑↑↑££ £££££!"
14 PRINTTAB(14)"!←←←←← ←←←←←!"
15 PRINTTAB(14)"!↑££££ £↑↑↑↑!"
16 PRINTTAB(14)"!←←←←← ←←←←←!"
17 PRINTTAB(14)"!£££££ £££££!"
18 PRINTTAB(14)"!←←←←← ←←←←←!"
19 PRINTTAB(14)"!↑↑£££ ££££↑!"
20 PRINTTAB(14)"!←←←←← ←←←←←!"
21 PRINTTAB(14)"!↑↑↑↑£ £££↑↑!"
22 PRINTTAB(14)"!←←←←← ←←←←←!"
23 PRINT:PRINTTAB(14)"!←←←←← ←←←←←!"
25 PRINT:PRINT" !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!"
26 PRINT" KERSZ SEGITSEGET ? (I/N)■■■■"
27 PRINT"■ "LF$
28 GETL1$:IFL1$="N"ORPEEK(56320)=111THEN100
29 IFL1$="I"THEN42
30 FORO=0TO150:NEXT
31 LF$=RIGHT$(LF$,1)+LEFT$(LF$,LEN(LF$)-1)
32 PL=KKK+100+M
33 POKEPL-40,32
34 POKEPL,35:POKEPL+R,0
35 POKEPL+40,34:POKEPL+R+40,2
36 POKEPL+80,32
37 M=M+LI:IFM>400ORM<=0THENLI=-LI
39 GOTO27
42 PRINT"■LIFTES FIU VAGY EGY NAGY SZALLODABAN,"
43 PRINT"ES A CIPOKET KELL A SZOBAKBOL KIHOZNOD."
44 PRINT"MINDEN PARERT 1 FT BORRAVALOT KAPSZ"
45 PRINT"A VENDEGEKTOL."
46 PRINT"HA 50 PAR FOLOTT SIKERUL A PORTARA "
47 PRINT"JUTTATNOD,KULON BORRAVALOT KAPSZ."
49 PRINT"■   [£] SZOBA EGY PAR CIPOVEL"
50 PRINT"■■   [↑] ZART AJTO"
51 PRINT"■■   [ ] URES SZOBA"
52 PRINT"■■   [#] JATEKOS"
53 PRINT"■■   [%] PORTAS"
54 PRINT"■■   ["CHR$(34)"] LIFT"
56 IFPEEK(56320)<>111THEN56
100 KEP$(1)=" !↑↑£££ ↑↑£££ ↑↑££↑ £££££ £££££!"
110 KEP$(2)=" !↑↑↑↑£ £££££ £↑£££ ££££↑ £££££!"
120 KEP$(3)=" !↑££££ ↑↑↑££ £££££ £££££ ££££↑!"
130 KEP$(4)=" !£££££ £££££ £↑£££ ££↑↑£ ££↑↑↑!"
140 KEP$(5)=" !↑££££ £££££ £££££ £££££ ↑↑↑↑↑!"
150 KEP$(6)=" !£££££ ££↑↑↑ ££££↑ ££££↑ ££££↑!"
160 KEP$(7)=" !£££££ £↑↑££ ££↑££ ££££↑ £££↑↑!"
170 KEP$(8)=" !£££££ ↑↑£££ £££££ ££↑££ ££↑↑↑!"
180 KEP$(9)=" !£££££ ↑↑£££ £££££ ↑↑↑↑£ £££££!"
183 PRINT"■";
184 PRINT" ■←←←←←←←←←←←←←←←←←←←←←←←←←←←←←←←←←←←"
185 PRINT" ■←←←←←←←←←←←←←←←←←←←←←←←←←←←←←←←←←←←"
```

```
187 PRINT" ▒!▒←←←←← ←←←←← ←←←←← ←←←←← ←←←←▒!"
190 FORC=1TO9
195 V=INT(RND(1)*8)+1
200 PRINTKEP$(V)
205 PRINT" ▒!▒←←←←← ←←←←← ←←←←← ←←←←← ←←←←▒!"
210 NEXT
215 PRINT"▒ !                %                  !"
217 PRINT"▒ !▒←←←←← ←←←←← ←←←←← ←←←←← ←←←←▒!"
220 PRINT"▒ ←←←←←←←←←←←←←←←←←←←←←←←←←←←←←←←←←"
221 FORC=KKK+77TOKKK+917STEP40:POKEC,38:POKEC+R,6:NEXT
230 POKEX,35:POKEX+R,0:POKEX+1,32:POKEX+2,32:POKEX-1,32:POKEX-2,32
250 POKE54296,15
260 POKE54277,31:POKE54278,128:POKE54272,1:POKE54273,1
270 PRINT"     OSSZ PONTSZAM  ";LEG;
280 PRINT"TTTTTTTTTTTTTTTTTTTTTTTTTTTTTT"
1000 GOSUB3000
1003 FORC=1TOMS:NEXT
1005 POKEL,32:POKEL+6,32:POKEL+12,32:POKEL+18,32
1010 IFPEEK(L+LI)=31THENLI=-LI
1020 L=L+LI
1030 POKEL,34:POKEL+6,34:POKEL+12,34:POKEL+18,34
1040 IFFL=0THEN2000
1050 POKEX,32:X=X+LI
1060 IFPEEK(X)=31THENX=X+40:GOTO9000
1070 POKEX,35:POKEX+R,0
2000 BO=BO-0.05
2010 POKEINT(BO)*40+KKK+37,32:IFBO=<1THEN9000
2990 GOTO1000
3000 IFPEEK(56320)=127THENMI=0:RETURN
3010 IFPEEK(56320)=<111THEN3500
3020 IFPEEK(56320)=123THENY=-1:GOTO3050
3030 IFPEEK(56320)=119THENY=1:GOTO3050
3040 RETURN
3050 IFPEEK(X+Y)=31THENY=Y-40:IFPEEK(X+Y)=31THEN9000
3053 :IFPEEK(X+Y)=28THENPOKE54276,129:CI=CI+1:GOTO3100
3054 IFPEEK(X+Y)=33THENRETURN
3055 IFPEEK(X+Y)=30THENRETURN
3057 IFPEEK(X+Y)=34ORPEEK(X+Y+40)=34THEN3900
3060 IFPEEK(X+Y)=37THEN3800
3065 IFPEEK(X+Y)=32ANDPEEK(X+Y+40)=32THEN:POKEX,32:X=X+Y:GOTO9000
3070 IFPEEK(X+Y)=32ANDPEEK(X+Y+40)=102THEN3100
3100 POKEX,32
3110 X=X+Y:FL=0
3120 POKEX,35:POKEX+R,0
3130 PRINT"] ←PONTOK←";CI+FT;:PRINT"←←CIPOK ←";CI
3180 POKE54276,0
3190 RETURN
3500 IFMI=0THENLI=-LI:MI=1
3510 RETURN
3800 MS=MS-1:IFCI<50THENFT=FT+CI:GOTO3810
3805 FT=FT+INT(CI*(BO*7))
3810 X=KKK+536:L=KKK+487:LI=40:BO=23:CI=0
3830 GOTO183
3900 POKEX,32
3905 X=X+Y
```

```
3910 IFPEEK(X)=34THENX=X-40:IFPEEK(X)=34THEN9000
3920 FL=1
3930 POKEX,35:POKEX+R,0
3940 RETURN
9000 FT=FT+CI:JSZ=JSZ-1
9010 IFBU=<1THENGOTO9020
9015 POKEX,32:X=X+40:IFPEEK(X+40)=31THEN9100
9017 POKEX,35:POKEX+R,0:GOTO9015
9020 POKEX,32
9030 BU=23:FORC=1TO23:POKEC*40+KKK+37,32:NEXT:FL=0
9035 L=KKK+487:L1=40:BU=23:CI=0
9036 IFJSZ=0THEN9800
9040 X=KKK+536:POKEX,35:POKEX+R,0:GOTO183
9100 POKEX,39:POKEX+R,0
9110 POKE54277,31:POKE54278,255:POKE54272,0:POKE54273,2:
     POKE54276,33
9120 FORC=1TO500:NEXT:POKE54276,0
9190 GOTO9020
9800 PRINT"▣";:POKE53281,1
9830 PRINT"▦▦▦▦▦▦▦▦▦▦▦▦▦▦▦                    ";FT+CI;"FT-UT KERESTEL";
9840 IFFT+CI>LEGTHENLEG=FT+CI
9850 FT=0:CI=0
9880 GETA$:IFA$<>""ORPEEK(56320)=111THEN6
9890 GOTO9880
50000 DATA127,127,81,22,20,22,56,0
50001 DATA60,60,102,102,126,126,102,0:A
50002 DATA120,124,102,124,126,103,126,0:B
50003 DATA60,126,102,96,102,126,60,0:C
50004 DATA124,126,99,99,99,126,124,0:D
50005 DATA127,126,96,124,96,126,127,0:E
50006 DATA127,126,124,96,124,120,112,0:F
50007 DATA62,121,112,115,115,63,31,0:G
50008 DATA115,115,115,127,127,115,115,0:H
50009 DATA28,28,28,28,28,28,28,0:I
50010 DATA127,127,7,7,103,63,31,0:J
50011 DATA115,119,126,126,126,119,115,0:K
50012 DATA112,112,112,112,112,126,127,0:L
50013 DATA119,127,107,107,107,99,119,0:M
50014 DATA103,115,123,127,127,119,115,0:N
50015 DATA62,127,119,119,119,127,62,0:O
50016 DATA126,127,115,127,126,112,112,0:P
50017 DATA62,127,99,111,111,127,62,0:Q
50018 DATA126,127,115,127,124,126,119,0:R
50019 DATA63,126,96,62,3,63,126,0:S
50020 DATA127,127,93,28,28,28,28,0:T
50021 DATA119,119,119,119,127,127,28,0:U
50022 DATA119,119,50,58,58,28,8,0:V
50023 DATA119,99,107,107,107,127,54,0:W
50024 DATA99,119,54,62,62,119,99,0:X
50025 DATA119,59,31,14,14,14,14,0:Y
50026 DATA127,63,7,28,112,126,127,0:Z
50027 DATA240,240,192,192,192,192,240,240:[
50028 DATA0,36,102,102,102,102,36,0:£
50029 DATA15,15,3,3,3,3,15,15:]
50030 DATA126,126,126,78,62,126,126,126:↑
50031 DATA239,239,239,0,251,251,251,0:←
```

```
50032 DATA0,0,0,0,0,0,0,0
50033 DATA255,0,255,0,255,0,255,0:!
50034 DATA255,255,255,189,60,60,60,60:"
50035 DATA0,60,60,24,255,24,60,36:#
50036 DATA101,85,87,87,85,85,85,101:$
50037 DATA24,24,24,60,60,126,255,48:%
50038 DATA127,254,127,254,127,254,127,254:&
50039 DATA0,24,24,126,126,24,24,24:'
50040 DATA24,48,96,96,96,96,48,24:(
50041 DATA24,12,6,6,6,6,12,24:)
50042 DATA48,72,72,16,16,0,16,0:*
50043 DATA0,60,60,126,126,60,60,0:+
50044 DATA0,0,0,0,0,12,24,48:,
50045 DATA0,0,0,126,126,0,0,0:-
50046 DATA0,0,0,0,0,96,96,0:.
50047 DATA0,6,12,24,24,48,96,0:/
50048 DATA0,24,60,102,102,60,24,0:0
50049 DATA0,12,28,60,12,12,12,0:1
50050 DATA0,12,18,4,8,16,30,0:2
50051 DATA0,30,2,4,2,18,30,0:3
50052 DATA0,16,16,16,20,30,4,0:4
50053 DATA0,30,16,28,2,2,28,0:5
50054 DATA0,12,18,16,28,18,12,0:6
50055 DATA0,30,18,4,8,16,16,0:7
50056 DATA0,12,18,12,18,18,12,0:8
50057 DATA0,24,36,28,4,36,24,0:9
50058 DATA-1

READY.
```

Reméljük, programunkkal sikerült örömet szereznünk.

7. FEJEZET
MEGSZAKÍTÁSOK (INTERRUPTS)

A megszakítós rutinok egyik meghatározása az lehetne, hogy a számítógép egy utasítására automatikusan végrehajtódnak. A C 64-es több megszakítási módot ismer:

a legnagyobb prioritású: RESET
a közepes prioritású: NMI
és a kis prioritású: IRQ

A prioritás (elsőbbség) a rutinok végrehajtási sorrendjét határozza meg, ha egyszerre több rutin is előfordul a programban.

7.1 RESET

Szoftver úton a RESET a SYS 64738 utasítással hajtható végre. Ez a cím a $FFFC–$FFFD alsó és felső byte-jaiból keletkezett érték.
Gyakran előfordul, hogy egy program félbeszakad, és a billentyűzetről nem tudunk utasítást bevinni. Ilyenkor a hardver-RESET-et kell végrehajtani. Ez a gyakorlatban a föld- és a RESET-vezeték rövidre zárásával valósul meg.
Erre a célra célszerű egy nyomógombot fixen a gépbe építeni. Érdemes egy összegzőt is elhelyezni a vezetékben, mivel néha előfordulhat, hogy véletlenül zárjuk az áramkört, és így a programunk megsérülhet.
Mi játszódik le a számítógépben a RESET végrehajtásakor? A $FFFC–$FFFD címek tartalma alapján a számítógép a $FCE2 címre ugrik.
Az ettől kezdődően lejátszódó folyamatot a következő program mutatja be.

```
0 REM  ****  P87.  ****
1 :
2 :
3 OPEN1,8,1,"@0:P 870"
4 OPEN2,4
5 SYS36864
6 .OPT O1,P2
7 *= $FCE2
10 LDX #$FF
20 SEI
30 TXS
40 CLD
50 JSR $FD02
60 BNE $FCEF
70 JMP ($8000)
80 STX $D016
```

```
90 JSR $FDA3
100 JSR $FD50
110 JSR $FD15
120 JSR $FF5B
130 CLI
140 JMP ($A000)
```

Az első négy cím számunkra nem fontos, csak annyit tartalmaz, hogy $FCE4 címen az Interrupt-Disable-Flag beállítódik. Ez a számítógép működését a továbbiakban nem befolyásolja.
Próbáljuk ki!
A kiadott RESET utasítás már nem állítható meg a RUN-STOP/RESTORE billentyűkkel.
Számunkra csak a $FCE7 címtől válik érdekessé a program. Itt ugrik a számítógép a $FD02 címen kezdődő alprogramba. Ez a alprogram megér egy kis odafigyelést.

```
0 REM  **** P88.  ****
1 :
2 :
3 OPEN1,8,1,"@0:P 88G"
4 OPEN2,4
5 SYS36864
6 .OPT O1,P2
7 *= $FD02
10 LDX #$0F
20 LDA $FD0F,X
30 CMP $8003,X
40 BNE $FD0F
50 DEX
60 BNE $FD04
70 RTS
```

Az alprogram a meghatározott sorrendben értékeket olvas ki a ROM-ból, és összehasonlítja őket a $8004–$8008 címekkel. Ha az értékek megegyeznek, akkor egy indirekt ugrást hajt végre. A számítógép ezenkívül kiolvassa a $88000–$001 címek tartalmát alsó és felső byte formájában, és a két byte által megadott címre ugrik.
Ha pl. a $8000 címen $00, a $8001 címen $60 áll, akkor az alprogram a $6000 címre ugrik tovább.
Ha tehát egy programot RESET-tel akarunk indítani, akkor a $8000–$8008 címeknek a következőképpen kell kinézni:

8000 LB	RESET ugróvektor
8001 HB	
8002 LB	NMI ugróvektor
8003 HB	
8004 195	C beállított 7. bittel
8005 194	B beállított 7. bittel
8006 205	M beállított 7. bittel

8007 56 8
8008 48 0

Ha azt akarjuk, hogy valamilyen programunk a bekapcsolás után azonnal fusson, címezzünk meg egy EPROM-ot az előbbiekben megadott értékkel. Ezen az elven működnek azok a modulok, amelyek a bekapcsolás után azonnal bejelentkeznek.

De térjünk vissza a RESET-hez!

Ha a $8000 címtől nem talál programot, akkor a $FCEF címen folytatja a számítógép a munkát. Az ezután következő alprogramok újra inicializálják a 0, 2, 3 lapokat, az I/O és a videotartományt. Végül egy ugrást hajt végre a BASIC hidegstartja.

Egy BASIC program indítható RESET-tel is. Ehhez a program 8000, 8001 címek alatti mutatóinak a következő programra kell mutatni:

```
0 REM  **** P89.  ****
1 :
2 :
10 :     JSR $A659
20 :     JMP $A7AE

READY.
```

Ez a két sor a BASIC RUN gépi kódú megfelelője.

7.2 NMI

Az NMI alacsonyabb prioritású, mint a RESET. Az NMI ugrási címe a $0318–$0319 (792–793) címeken helyezkedik el. Alapállapotban a címek tartalma: $FE47 (65095).

Mivel e tulajdonképpeni rutin a $FE43 (65091) címen kezdődik, nézzük innen az utasításokat:

```
0 REM  **** P90.  ****
1 :
2 :
3 OPEN1,8,1,"00:P 900"
4 OPEN2,4
5 SYS36864
6 .OPT O1,P2
7 *= $FE43
10 SEI
20 JMP ($0318)
30 PHA
40 TXA
50 PHA
60 TYA
70 PHA
80 LDA #$7F
90 STA $DD0D
```

```
100 LDY $DD0D
110 BMI $FE72
120 JSR $FD02
130 BNE $FE5E
140 JMP ($8002)
150 JSR $F6BC
160 JSR $FFE1
170 BNE $FE72
180 JSR $FD15
190 JSR $FDA3
200 JSR $E518
210 JMP ($A002)

READY.
```

Mint előbb is láttuk, a $FE43 címen az Interrupt-Disable-Flag beállítódik.
Ha a $0318–$0319 címekre saját programunk kezdőcímét írjuk, akkor a programnak kell egy CLI utasítást is tartalmazni, különben megakadályozzuk az IRQ működését (ennek következményeiről lásd az IRQ-val foglalkozó fejezetet).

Példa saját programra:

POKE 972,226: POKE 793,252

Most az NMI-vektor a RESET-re mutat.
A RESTORE billentyű lenyomásával (=NMI) egy RESET hajtódik végre. Ha a RESET után ismét lenyomjuk a RESTORE billentyűt, akkor ez a hatás már nem érvényesül, mert a $0318–$0319 címek tartalma már az alapállapotnak megfelelő.

Visszatérve az NMI rutinhoz
A $FE47–$FE48 (65095–65099) címeken levő utasítások kimentik az akkumulátor X és Y regiszter tartalmát. Az $FE54 (65108) vizsgálja meg az RS 232-es állapotát. Ha aktiv, akkor az $FE72 (65138) címre ugrik, ahol az RS 232 NMI rutinja található.
A $FE56 (65110) cím funkciója a RESET-ből már ismert: megvizsgálja, hogy a $8000 címtől helyezkedik-e el modul. Ha igen, akkor a $8002 alacsony és az $8003 magas byte-ja az ugrási címet tartalmazza, és erre a címre ágazik el a JMP($8002).
Mivel a RESET és az NMI a $8000-tól meghatározott értékekre reagálnak, nagyon egyszerűvé vált két különböző program indítása. Az egyik egy RESET hatására, a másik a RESTORE billentyűre indul el.
A $FE61 cím (65121) kérdezi le a STOP billentyűt. Ha lenyomjuk, akkor a program a $FE66 (65126) címre ugrik, és újra inicializálja az I/O egységet, törli a képernyőt és ugrást hajt végre a BASIC melegstartra.
Ez a magyarázata annak, hogy a RESTORE billentyű magában nem hatásos. A számítógép ui. először a STOP billentyűt kérdezi le. Ha nincs lenyomva, akkor az NMI elágazik az RS 232-re.

Próbáljuk ki az elmondottakat:

```
0 REM  ****  P91.  ****
1 :
2 :
10 FORX=0TO8:READA:POKE32768+X,A:NEXTX
20 DATA68,229,53,164,195,194,205,56,48

READY.
```

RUN

A $8000–$8008 (32768–32776) címek tartalma:

32768 68	32772 195
32769 229	32773 194
32770 53	32774 205
32771 164	32775 56
	32776 48

A 32768–32769-es címek tartalma megadja, hogy a vezérlés egy RESET után hová adódjon. Az 58692-es címen a képernyőtörlés alprogram kezdődik.
A 32770–32771-es címek tartalma azt adja meg, hogy egy NMI utasítás után hová ugorjon a számítógép. Ez jelen esetben a 42037-es cím. Ezen a címen kezdődik az OUT OF MEMORY ERROR hibajelzés kiadása.
A 32772–32776-os címeken állítható be, hogy a ROM $8000-től futtatható legyen. Ha a megadott két sor beírása és indítása után lenyomjuk a RESTORE billentyűt, akkor az OUT OF MEMORY ERROR hibajelzés fog kiíródni.
Előfordulhat az is, hogy a kurzor a helyén marad. Ez nem meglepő, hiszen a kurzort az IRQ rutin kezeli. Mivel azonban az NMI rutin nagyobb prioritású, mint az IRQ, ezért az NMI rutinra ugrik, mielőtt az IRQ rutin végrehajtódott volna. Ha kiadunk egy RESET utasítást, akkor a képernyő törlődik. Ezzel a programindítás egy új módszerét fedeztük fel.
Emlékeztetőül: ha a programot a $8000-es címtől tároltuk, akkor betöltés után a RESTORE vagy egy RESET hatására elindul.
Mivel ezen a címen nincs BASIC sor, amely a program kezdetét megmutatná, jogtalan felhasználónak elég nehéz ezt a programot elindítani. Még biztonságosabb lenne ez a megoldás, ha a RESTORE vagy a RESET kiadása után egy jelszót kellene megadni.
Azok számára, akik nem mélyedtek el ebben a témakörben, elég hatásosnak bizonyulhat ez a védelmi módszer. Még nehezebbé tehetjük a programhoz való hozzáférést, ha ezt a megoldást más megoldással kombináljuk.

7.3 IRQ

Ez a megszakítási mód a prioritási sorrendben az utolsó helyen áll, de azért éppen olyan fontos, mint a többi. Sőt, mivel a gyakorlatban ezt használjuk a leggyakrabban, ezért ezt a megszakítási módot tartjuk a legfontosabbnak. Az IRQ rutin kezdőcíme $0314–$0315 (788–789) címeken helyezkedik el. Ezeket a címeket egyszerűen megváltoztathatjuk.
Prbóljuk ki:

```
POKE 788,226: POKE 789,252
```

Ha most lenyomjuk a RETURN-t, akkor a számítógép egy RESET-et hajt végre. Hogyan lehetséges ez?
Úgy, hogy 226+252*256 = 64738 a RESET címe. Nagyon fontos tudni, hogy az IRQ nem egy meghatározott jel hatására hajtódik végre (mint az NMI vagy a RESET), hanem minden 1/60 s-ban. Ebből adódik az IRQ rutin használhatósága olyan feladatok megoldásában, ahol valamit állandóan végre kell hajtani. Pl.: mindig megadja a pontos időt, villogtatja a kurzort és lekérdezi a STOP billentyűt. A rutin felépítése:

```
0 REM  ****  F92.  ****
1 :
2 :
3 OPEN1,8,1,"@0:P 926"
4 OPEN2,4
5 SYS36864
6 .OPT O1,P2
7 *= $EA31
10 JSR $FFEA
20 LDA $CC
30 BNE $EA61
40 DEC $CD
50 BNE $EA61
60 LDA #$14
70 STA $CD
80 LDY $D3
90 LSR $CF
100 LDX $0287
110 LDA ($D1),Y
120 BCS $EA5C
130 INC $CF
140 STA $CE
150 JSR $EA24
160 LDA ($F3),Y
170 STA $0287
180 LDX $0286
190 LDA $CE
200 EOR #$80
210 JSR $EA1C
220 LDA $01
230 AND #$10
```

```
240 BEQ $EA71
250 LDY #$00
260 STY $C0
270 LDA $01
280 ORA #$20
290 BNE $EA79
300 LDA $C0
310 BNE $EA7B
320 LDA $01
330 AND #$1F
340 STA $01
350 JSR $EA87
360 LDA $DC0D
370 PLA
380 TAY
390 PLA
400 TAX
410 PLA
420 RTI

READY.
```

A $EA31-es (59953) címek a program az $F69B címtől kezdődő alprogramra ugrik, ahol átállítja az órát, és lekérdezi a STOP billentyűt. Ha átugorjuk ezt a címet, akkor az óra nem meg tovább és a RUN-STOP sem működik (kivéve a kazettás egységgel való kapcsolat esetén, ahol a RUN-STOP billentyűnek más funkciója van).
Próbáljuk ki:

POKE 788,52

Ez az utasítás az IRQ kezdőcímét a $EA34-es (59956) címre helyezi át. Írjuk be a következő kis programot:

```
0 REM  ****  P93.  ****
1 :
2 :
10 PRINT TI$
20 GOTO 10

READY.
```

A program elindítása után láthatjuk, hogy a TI$ változó értéke nem változik, ha előtte a POKE 788,52 utasítást kiadtuk. A STOP billentyű sem működik.
Mi a helyzet a RUN–STOP/RESTORE billentyűkkel?
Ismételjük át az NMI-ről olvasottakat: a RESTORE billentyű megnyomása után a program az NMI rutinra ugrik. A rutin kérdezi le a STOP billentyű állapotát. Az IRQ rutin így nem játszik szerepet, de ez a probléma feloldható. A $EA36-os (59958) címen vizsgálja meg a rendszer, hogy a kurzor be van-e kapcsolva. Ha nincs, akkor rögtön a $EA61-es (60001) címre ugrik. Ha be van kapcsolva, akkor

a program végrehajtása folytatódik. De egy pillanat! A $CC (204) cím tájékoztat arról, hogy a kurzor be vagy ki van-e kapcsolva, tehát ezzel is lehet befolyásolni a folyamatot.
Programban pl. ilyen módon:

```
0 REM **** P93/1. ****
1 :
2 :
10 POKE204,0:REM KURZOR BE
20 GETA$:IFA$=""THEN20
30 POKE204,1:REM KURZOR KI

READY.
```

Ha kipróbáljuk a programot, akkor lesz egy kurzorunk. Így tudjuk pl. a felhasználó figyelmét felhívni egy GET utasításra. A programnak azonban van egy hátránya. Nyomjunk le egy billentyűt (tehát hagyjuk el a 20-as sort). Ha a kurzor megjelenik a képernyőn, akkor ott is marad, és egy másik kurzor jelenik meg egy újabb billentyű lenyomása után. Írjuk a programunkhoz a következő sort:

```
0 REM **** P93/2. ****
1 :
2 :
25 IFPEEK(207)=1THEN25
```

A 207-es címen lekérdezhető, hogy a kurzor a képernyőn van-e (=1) vagy nincs (=2).

Foglalkozzunk tovább az IRQ-rutinnal!
A $EA 38-as (59960) címen a kurzor időszámlálója eggyel csökken. Ha értéke nem nulla, akkor a $EA61-es címre ugrik a program. Ha nulla, akkor a címe $14 (#20) lesz (mivel a számítógép az IRQ rutint másodpercenként 60-szor hajtja végre, és ez a ciklus 20-szor fut le, mialatt a kurzor megváltozik, így könnyen kiszámíthatjuk, hogy a kurzor 1/3 s-onként villog).
Az IRQ rutin egyszerű módosításával tehát meg tudjuk változtatni a kurzor villogási sebességét:

```
0 REM  **** P94. ****
1 :
2 :
10 :      JSR   $FFEA     ;IDO,STOP BILLENTYU
20 :      LDA   $CC       ;KURZOR BE
30 :      BNE   *         ;NEM
40 :      DEC   $CD       ;AZ IDOSZAMLALO CSOKKENTESE
50 :      BNE   *         ;HA NEM NULLA, TOVABB
60 :      LDA   #FF       ;AZ IDOSZAMLALO UJRAALLITASA
70 :      JMP   $EAE      ;IRQ TOVABB
80 :   *  JMP   $EA61     ;TOVABB

READY.
```

A BASIC betöltőprogram:

```
0 REM  **** P95.  ****
1 :
2 :
10 FORX=0TO18:READA:POKE24625+X,A:NEXTX
20 DATA 32,234,255,165,204,208,9,198,205,208,5,165,255,76,62,
   234,76,97,234
```

A RUN parancs után még írjuk be a

POKE 788,96

utasítást, mivel az IRQ rutin helyett most is saját rutinunkat használjuk.
Most már a 255-ös cím tartalmának 0-tól (a kurzor nem villog) 1-en keresztül (nagyon gyorsan villog) egészen 255-ig (nagyon lassan villog) változtathatjuk a villogási sebességet. Mivel a 255-ös cím tartalmát a rutin folyamatosan olvassa, a cím tartalmának megváltoztatása azonnal végrehajtódik.
Az utolsó címeken, egészen az $EA5E (59998) címig a számítógép a kurzorral foglalkozik. Felvillantja a képernyőn, figyeli az alatta levő karaktert és annak színét stb. Érdemes kipróbálni azt az esetet is, amikor az IRQ rutinban ezt az egész részt átugorjuk:

POKE 788,97

Ekkor a következőket tapasztalhatjuk:

a) az idő számlálása megáll;
b) a STOP billentyű nem működik;
c) a kurzor eltűnik a képernyőről.

A karakterek viszont ezután is megjelennek, mivel az IRQ rutin a billentyűzetet csak később kérdezi le. A $EA61–$EA79-es (60001–60025) címek a kazettás egységre vonatkoznak. Ha a kazettás egységen lenyomunk egy billentyűt, akkor a motor elindul. Ha nem nyomjuk le, akkor áll.
Próbáljuk ki!

a) Írjuk be: POKE 788,123 (IRQ kezdőcíme $EA7B).
   Ha most lenyomjuk a PLAY billentyűt, a motor nem indul el.

b) Ha lenyomjuk a RUN-STOP/RESTORE billentyűt, akkor ismét visszakerülünk az alapállapotba.
   Nyomjuk le a kazettás egység PLAY billentyűjét és utána írjuk be a POKE 788,123 utasítást.
   Most nyomjuk le a kazettás egység STOP billentyűjét: a motor nem áll le.

A $EA7B cím elágazik egy alprogramba, ahol megtörténik a billentyűzet lekérdezése és a megfelelő jelek kiírása. Ha most ezt a címet átugorjuk:

```
POKE 788,126
```

utasítással, akkor a számítógép segítség nélkül marad.
A RUN-STOP/RESTORE azonban ilyenkor is működik. Ez azért lehetséges, mert a RESTORE billentyű lekérdezése különbözik a többi billentyű lekérdezési módjától. A RESTORE billentyűtől egy külön vezeték megy a CPU 4. csatlakozótűjére (NMI). Így a RESTORE-t a központi egység közvetlenül kérdezi le.
A $EA81–$EA85-ös címeken állítódnak vissza az eredeti regisztertartalmak. Az RTI (Return From Interrupt) utasítással lezárul az IRQ rutin.
Az IRQ rutin könnyen megváltoztatható vagy egyszerűen átugorható. Ezt a programozásban sok célra fel lehet használni. A következő oldalakon ehhez szeretnénk ötleteket adni. Az átíráskor egy szabályt azonban mindig tartsunk szem előtt: a saját rutinunktól mindig térjünk vissza az eredeti IRQ rutinra, mert ha ezt nem tesszük, akkor

a) a STOP billentyű;
b) az óra;
c) a kurzor;
d) a kazettás egység;
e) a billentyűzet

működése megszűnik (kivéve, ha ez a célunk).

7.4 Így kell az IRQ-t programozni!

A következőkben néhány tippet adunk a programozóknak az IRQ alkalmazására.
Ha egy megszakító rutint programozunk, akkor a 788–789-es címen levő vektort kell megváltoztatni. A későbbiekben bemutatjuk, hogyan kell ezt megvalósítani.
Az IRQ rutin nagy előnye, hogy mind program, mind parancs üzemmódban 1/60 s-onként végrehajtódik.
Ez széles körű felhasználást tesz lehetővé. Ezek közül mutatunk be néhány példát.

7.4.1 Mindig aktív

Első példánk érthetővé teszi, hogy milyen lehetőségek rejlenek az IRQ programozásában, és milyen sebességgel dolgozik az IRQ.

```
0 REM  ****  P96.  ****
1 :
2 :
10 DATA 120,169,15,141,20,3,169
20 DATA 159,141,21,3,133,56,88
30 DATA 96,238,32,208,76,49,234
40 FOR I= 40704 TO40724
50 READ A
60 POKE I,A
70 S=S+A
80 NEXT I
90 IF S <>2171 THEN PRINT "A HIBA A DATA SORBAN VAN":END
100 PRINT "OK": SYS 40704
110 NEW

READY.
```

A rutin tulajdonképpen nem csinál mást, mint minden megszakítási ciklusban növeli 1-gyel a képernyő keretszínét. Ez olyan gyorsan játszódik le, hogy egyetlen keretszín sem ismerhető fel a képernyőn. Belátható, hogy egy jól programozott megszakító rutin nem csökkenti a végrehajtási sebességet. A rutin futása alatt nyugodtan vihetünk be BASIC sorokat, vagy futtathatjuk programjainkat.
Ha zavaró a képernyő keretének villogása, akkor a RUN-STOP/RESTORE billentyűkkel leállíthatjuk a rutint. Az újraindítás a

SYS 40704

paranccsal lehetséges.
A rutint lényegében két részre oszthatjuk. Az első részben megváltoztattuk az IRQ vektort, és levédtük az átírás ellen. A vektorok átírása BASIC-ben nem is olyan egyszerű dolog. Ha azonban a MINDIG AKTIV rutin kezdőcímét 40719-re választjuk (a második rész kezdete), és beírjuk a

POKE 788,15: POKE 789,159

utasításokat, a következőket fogjuk tapasztalni: amint a 788-as címet megváltoztattuk, de a 789-est még nem, a program a $EA31 helyett a $EAOF címre ugrik. Így a számítógép működése felfüggesztődik.
Ezek a rutinok a gépi kódú program első részében automatikusan átirják az értékeket. A későbbiekben azt is megmutatjuk majd, hogyan lehet a vektorokat BASIC-ben megváltoztatni. Gépi kódban van egy utasítás, a SEI (Set Interrupt), amely megakadályozza a megszakítás végrehajtását. Ezután már akadály nélkül átírhatjuk a vektorokat. A gépi kódú CLI (Clear Interrupt) hatására a megszakító rutin az új vektorokkal folytatódik. Ezután az első részben a rutint még leblokkoltuk, így az sem változókkal, sem BASIC programmal nem írható át olyan egyszerűen. Ezért állítottuk a BASIC tároló végét a rutin kezdetére.

A jobb érthetőség kedvéért felsoroljuk a programban használt összes gépi kódú utasítás jelentését:

SEI	Megakadályozza a megszakítást
LDA	Meghatározzuk a 2. rész kezdőcímének alacsony byte-ját
STA $ 0314	Az alacsony byte értékét az IRQ mutatóban tároljuk
LDA	Meghatározzuk a 2. rész magas byte-ját
STA $ 0315	A magas byte értékét az IRQ mutatóban tároljuk
STA $ 38	A tárolóvég magas byte-ját a rutin elején tároljuk, így blokkol
CLI	A megszakítás újraindítása

Ezenkívül az első részben, mint a hangjelzés rutinnál látni fogjuk, beállíthatunk olyan értékeket, amelyeket a második részben alapadatként felhasználhatunk. A második rész a tulajdonképpeni rutin. A számítógép erre a részre ugrik. A könyvünkben bemutatott megszakító rutinok csak példák, a programozó elengedheti a fantáziáját és beírhatja az elképzeléseit.
Ügyelni kell azonban arra, hogy a rutin ne legyen túl hosszú, vagyis ne lassítsa a parancsvégrehajtás sebességét. Nagyon fontos, hogy a rutin végén visszaugorjunk a $EA 31-es címre, különben a számítógép üzemképtelenné válik.

7.4.2 Billentyűzet-hangjelzés

A sok unalmas elmélet után lássunk egy igazi programot. Néhány számítógépen a billentyű lenyomását hangjelzés kíséri, ami programok és táblázatok bevitelekor nagyon hasznos lehet. A C 64-es nem rendelkezik ilyen lehetőséggel. Az IRQ rutin segítségével azonban megoldható a feladat.

```
0 REM  **** P97.  ****
1 :
2 :
5 REM HANG RUTIN
10 FOR I=0 TO 61
20 READ A
30 S=S+A
40 POKE 40704+I,A
50 NEXT I
60 IF S<>6973 THEN PRINT "HIBA A DATA SORBAN": END
70 PRINT "OK"
80 SYS 40704
90 DATA 169,255,141,6,212,141,24,212,169
100 DATA 9,141,5,212,169,103,141,1,212
110 DATA 169,33,141,0,212,120,169,38,141
120 DATA 20,3,169,159,141,21,3,133,56,88
130 DATA 96,72,165,203,201,64,208,9,169
140 DATA 0,141,4,212,104,76,49,234,169
150 DATA 17,141,4,212,76,50,159

READY.
```

A jobb megértés kedvéért a gépi kódú lista:

```
0 REM  **** P98.  ****
1 :
2 :
3 OPEN1,8,1,"@0:P 98@"
4 OPEN2,4
5 SYS36864
6 .OPT O1,P2
7 *= $9F00
10 LDA #$FF
20 STA $D406
30 STA $D418
40 LDA #$09
50 STA $D405
60 LDA #$67
70 STA $D401
80 LDA #$21
90 STA $D400
100 SEI
110 LDA #$26
120 STA $0314
130 LDA #$9F
140 STA $0315
150 STA $38
160 CLI
170 RTS
180 PHA
190 LDA $CB
200 CMP #$40
210 BNE $9F36
220 LDA #$00
230 STA $D404
240 PLA
250 JMP $EA31
260 LDA #$11
270 STA $D404
280 JMP $9F32

READY.
```

A rutin első részének $9F17–$9F25 közötti részletét a "mindig aktív" programból vettük. Ezzel szemben a megelőző rész csaknem teljes egészében a hangleírást tartalmazza, a hullámforma kivételével. Ezekre az értékekre u tulajdonképpeni rutinban már nincs is szükség.
A rutin második része lekérdezi a 203-as címet, hogy lenyomtunk-e valamilyen billentyűt. Ha itt 64 áll, akkor nem nyomtunk le billentyűt, a hullámforma értéke 0 lesz, vagyis nem hallunk hangot. Ha lenyomunk egy billentyűt, akkor ez az érték 64-ről 17-re változik, és egy hangjelzést hallunk. A lekérdezés után a program a $EA31-es (59953) címre ugrik vissza.

7.4.3 Aláfestő zene

Ha nagyon szeretjük a zenét, és programozás közben sem akarunk lemondani róla, de a közelünkben nincs semmilyen zenedoboz, és a családban sem akar senki énekelni (vagy mi nem akarjuk, hogy énekeljen), akkor ez a program megoldja gondjainkat. A program nagyon hosszú, ami abból adódik, hogy minden hangot 3 értékkel kell meghatározni, a frekvencia magas-, a frekvencia alacsony- és a hangjegy értékkel.
A BASIC betöltőprogram két részből áll: az első rész betölti a rutint, a második pedig a hangjegyeket olvassa be.
A programot a kipróbálás előtt tároljuk!

```
0 REM **** P99. ****
1 :
2 :
5 REM ZENE A MEGSZAKITO RUTIN SEGITSEGEVEL
10 DATA120,169,38,141,20,3,169,144
20 DATA141,21,3,88,133,56,169,0
30 DATA141,243,159,169,1,141,241,159
40 DATA169,0,141,20,212,169,31,141
50 DATA24,212,141,19,212,96,206,241
60 DATA159,208,48,72,138,72,169,0
70 DATA141,18,212,174,243,159,189,0
80 DATA145,141,14,212,189,0,146,141
90 DATA15,212,189,0,147,141,241,159
100 DATA169,33,141,18,212,232,56,224
110 DATA60,144,2,162,0,142,243,159
120 DATA104,170,104,76,49,234
130 FORI=36864TO36957
140 READA
150 POKEI,A
160 S=S+A
170 NEXT
180 IFS<>11531THENPRINT"HIBAS DATA SOR..!!":END
190 INPUT"A HANGJEGYEK SZAMA";N:POKE36944,N
200 REM HANGJEGYEK: T=60 HANGJEGY !!
210 FORI=0TO 59
220 READL
230 POKE37120+I,L
240 READH
250 POKE37376+I,H
260 READW
270 POKE37632+I,W
280 NEXT
290 SYS36864
300 REM ZENE-DATAK
310 DATA196, 9,10, 10,13,10, 10,13,10
320 DATA162,14,10,109,16,10, 10,13,10
330 DATA  109,16,10,162,14,10,196, 9,10
340 DATA   10,13,10, 10,13,10,162,14,10
350 DATA  109,16,10, 10,13,20,158,11,10
360 DATA  196, 9,10, 10,13,10, 10,13,10
```

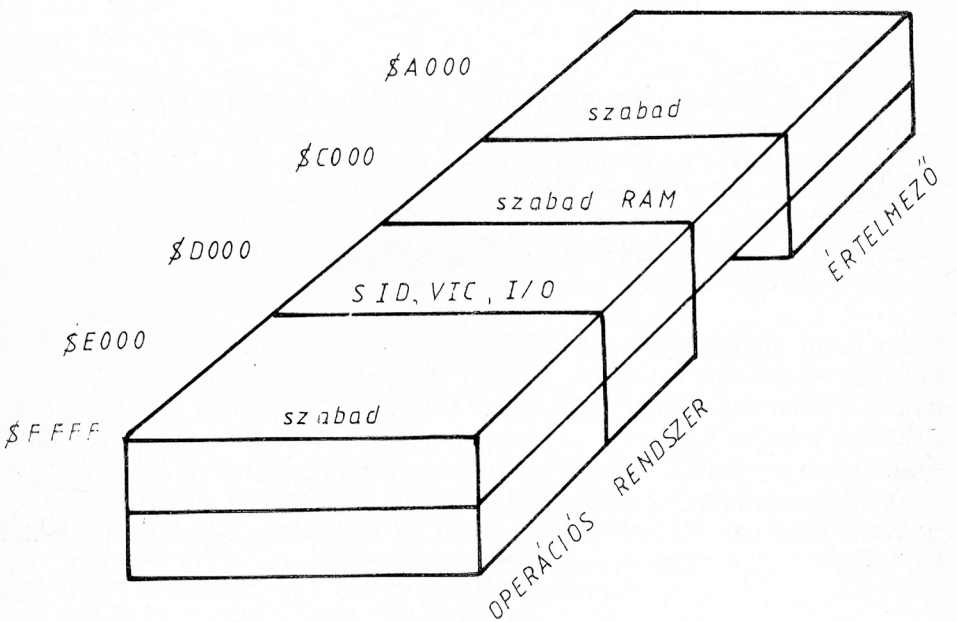
```
370 DATA  162,14,10,109,16,10,103,17,10
380 DATA  109,16,10,162,14,10, 10,13,10
390 DATA  158,11,10,196, 9,10,247,10,10
400 DATA  158,11,10, 10,13,20, 10,13,10
410 DATA    0, 0,10,247,10,15,158,11, 5
420 DATA  247,10,10,196, 9,10,247,10,10
430 DATA  158,11,10, 10,13,20,196, 9,15
440 DATA  247,10, 5,196, 9,10,180, 8,10
450 DATA   55, 8,10,180, 8,10,196, 9,20
460 DATA  247,10,15,158,11, 5,247,10,10
470 DATA  196, 9,10,247,10,10,158,11,10
480 DATA   10,13,10,247,10,10,196, 9,10
490 DATA   10,13,10,158,11,10,162,14,10
500 DATA   10,13,20, 10,13,10,  0, 0,99

READY.
```

A program az indítása után megkérdezi, hány hangjegyből álljon a zenedarab. A mi kis dalocskánknál ez a szám 59. Ezután a számítógép beolvassa a hangjegyek adatait, és elindítja a rutint.
Ha mindent jól csináltunk, és nincs hiba a DATA sorokban, akkor felhangzik a YANKEE DOODLE című dal. Mint már korábban említettük, nyugodtan programozhatunk, a zene tovább szól. Felhasználhatjuk rutinunkat egy játékprogram speciális effektusaihoz is. Ilyenkor célszerű csak az első 3 hangot használni.

Ezt a rutint önállóan is használhatjuk. A gépi kódú programozás ismeretei nélkül is meg tudjuk változtatni a zenét. Ehhez csak a frekvencia alsó és felső byte-jait, valamint a hangjegyértékeket kell beírnunk a 300-as sortól a YANKEE DOODLE hangjai helyett. A program indítása után már csak a hangjegyek számát kell megadni. A program maximum 256 hangot tud feldolgozni.
A hangjegyértékeket szabadon választhatjuk. A YANKEE DOODLE esetén a negyedhangoknak 10, a félhangoknak 20, az egész hangoknak 40 értéket adtunk. Ezektől az értékektől azonban el lehet térni.

8. FEJEZET
AZ OPERÁCIÓS RENDSZER: ROM A RAM-BAN

A COMMODORE 64-esnek a többi személyi számítógéppel szemben van egy nagyon érdekes tulajdonsága: operációs rendszere átmásolható a RAM területre.
A jobb megértés végett tanulmányozzuk át a következő ábrát, amely a $A000-tól elterülő memóriaterület felosztását mutatja:



1. ábra

Az ábrából látható, hogy bizonyos tárolóterületek duplán vannak lefedve. Ez azt jelenti, hogy ezekben a címtartományokban egy ROM és egy RAM tároló is található. Másképp kifejezve, pl. a $A000 címnek két megfelelője van: az egyik a ROM, a másik a RAM területen fekszik.
A számítógép természetesen csak az egyik tárolótartományt tudja megcímezni. Ebben a fejezetben közelebbről két tartománnyal fogunk megismerkedni: az egyik a $A000–$BFFF (a BASIC értelmező), a másik a $E000–$FFFF (I/O egység).

8.1 Másolórutinok

A két tartomány ROM-ja az értelmezőt és az I/0 egységet tartalmazza. Az "alattuk levő" RAM terület üres. A RAM a számítógép kikapcsolásakor mindig törlődik. Nem állunk messze a megoldástól, amely lehetővé teszi, hogy az értelmezőt és/vagy az I/0 egységet a RAM területre átmásoljuk, és átkapcsoljunk erre az átmásolt területre.
Csak annyit kell tudnunk, hogy a PEEK utasítással a ROM-ot ki lehet olvasni, míg a POKE utasítással a RAM-ba lehet írni. Az értelmező másolórutinja tehát így néz ki:

```
0 REM  ****  P100.  ****
1 :
2 :
10 FOR A=10*4096 TO 12*4096-1
20 POKE A,PEEK(A)
30 NEXTA
40 END

READY.
```

Az operációs rendszer másolórutinja pedig:

```
0 REM  ****  P101.  ****
1 :
2 :
10 FOR A=14*4096 TO 16*4096-1
20 POKE A,PEEK(A)
30 NEXTA
40 END
```

Csak a másolás után kapcsoljunk át a RAM területre, különben a számítógép üzemképtelenné válik.

ért. ROM	*	ért. RAM	*	op. r. ROM	*	op. r. RAM	*	átkapcsolás
1	*	0	*	1	*	0	*	POKE 1,55
0	*	1	*	1	*	0	*	POKE 1,54
0	*	1	*	0	*	1	*	POKE 1,53
1	*	0	*	0	*	1	*	.....

A táblázatból kitűnik, hogy BASIC-ből nem tudjuk csak az operációs rendszert átkapcsolni.

Bizonyára feltűnt az Olvasóknak, hogy a BASIC másolórutin milyen lassú. A következő gépi kódú programmal 1 s-nál rövidebb időre csökkenthetjük ezt az időt!

A gépi kódú lista:

```
0 REM  ****  P102.  ****
1 :
2 :
3 OPEN1,8,1,"@0:P102G"
4 OPEN2,4
5 SYS36864
6 .OPT O1,P2
7 *= $033C
10 LDX #$20
20 LDA #$A0
30 LDY #$00
40 STY $22
50 STA $23
60 LDA ($22),Y
70 STA ($22),Y
80 INY
90 BNE $0346
100 INC $23
110 DEX
120 BNE $0346
130 RTS

READY.
```

A BASIC betöltőprogram:

```
0 REM  ****  P103.  ****
1 :
2 :
10 FOR I=828 TO 850
20 READX:POKEI,X:NEXT
30 DATA162,32,169,160,160,0,132,34,133,35,177,34
40 DATA145,34,200,208,249,230,35,202,208,244,96
50 SYS828

READY.
```

A rutin SYS 828-cal indítható.

Ha az értelmezőt is és az I/0 egységet is át akarjuk ezzel a rutinnal másolni, változtassuk meg a BASIC betöltőprogram 50. sorát:

```
0 REM  ****  P104.  ****
1 :
2 :
50 SYS828:POKE831,224:SYS828
READY.
```

Ezek után felmerül a kérdés: mit kezdhetünk a RAM területre átmásolt operációs rendszerrel?
*A Commodore 64-es belső felépítése** c. könyv tartalmazza mindkét terület teljes assembler-listáját. Ezt tudjuk ezután kívánságainknak megfelelően átalakítani. Megváltoztathatjuk a BASIC hibajelzéseket, vagy a parancsokat, átírhatjuk a BASIC utasítások rutinjait stb.
Lehetővé vált, hogy egy saját értelmezőt írjunk!

* A Commodore 64-es belső felépítése, DATA BECKER–NOVOTRADE 1985.

9. FEJEZET
AZ OPERÁCIÓS RENDSZER RUTINJAI

Gépi kódú programozás során gyakran okoz nehézséget, hogy az assembler-utasítások nem olyan sokoldalúak, mint a BASIC utasítások. A PRINT utasítás gépi kódban pl. komoly problémát jelent. Sok minden egészen egyszerűvé válik azonban, ha megismerjük az operációs rendszer rutinjait, és munkánk során használjuk ezeket.
Az operációs rendszernek két része van:

a) az értelmező;
b) a be- és kimeneti egység (I/O).

Az értelmező feladata a BASIC utasítások lefordítása a számítógép által érthető gépi kódra. A be- és kimeneti egység tartja a "külvilággal" a kapcsolatot. A következő oldalak elolvasása után biztosan találunk olyan rutinokat, amelyeket a későbbiekben jól lehet hasznosítani.

Tömbeltoló rutin
Ugrási cím: $A3B8 (41912)
A rutinnal gyorsan és pontosan lehet egy tárolóterületet egy másik helyre eltolni. Ehhez a következőket kell megadni: a régi blokk kezdőcíme a $5F–$60 (95–96) címre, a régi blokk végcíme (+1!) a $5A–$5B (90–91) címre, az új blokk végcíme (+1!) a $58–$59 (88–89) címre.

A verem vizsgálata
Ugrási cím: $A3FB (41979)
A rutin megvizsgálja, hogy van-e még szabad hely a veremben. Ha nincs, akkor OUT OF MEMORY hibajelzést ad.

OUT OF MEMORY hibajelzés
Ugrási cím: $A435 (42037)
A rutin kiírja az OUT OF MEMORY hibajelzést, és visszatér a parancs-üzemmódba.

Hibajelzés
Ugrási cím: $A43A (42042)
A rutinnal ki lehet íratni a hibajelzéseket. A kiírás után a számítógép visszatér a parancs-üzemmódba. A hiba sorszámát az X regiszterben kell elhelyezni. A következő táblázatban a hibasorszámokhoz tartozó hibajelzéseket láthatjuk.

A hiba sorszáma dec.	hex.	Hibajelzés
1	1	TOO MANY FILES
2	2	FILE OPEN
3	3	FILE NOT OPEN
4	4	FILE NOT FOUND
5	5	DEVICE NOT PRESENT
6	6	NOT INPUT FILE
7	7	NOT OUTPUT FILE
8	8	MISSING FILE NAME
9	9	ILLEGAL DEVICE NUMBER
10	A	NEXT WITHOUT FOR
11	B	SYNTAX
12	C	RETURN WITHOUT GOSUB
13	D	OUT OF DATA
14	E	ILLEGAL QUANTITY
15	F	OVERFLOW
16	10	OUT OF MEMORY
17	11	UNDEF/D STATEMENT
18	12	BAD SUBSCRIPT
19	13	REDIM/D ARRAY
20	14	DEVISION BY ZERO
21	15	ILLEGAL DIRECT
22	16	TYPE MISMATCH
23	17	STRING TOO LONG
24	18	FILE DATA
25	19	FORMULA TOO COMPLEX
26	1A	CAN/T CONTINUE
27	1B	UNDEF/D FUNCTION
28	1C	VERIFY
29	1D	LOAD
30	1E	BREAK

Ez a rutin BASIC-ből is hívható:

```
POKE 781,21: SYS 42042
```

Az utasítások hatására az ILLEGAL DIRECT ERROR felirat jelenik meg.

A BASIC programsorok újrarendezése

Ugrási cím: $A533 (42291)

A rutin újrarendezi a BASIC sorokat. Hatására a sorok növekvő sorrendben rendeződnek.

NEW
Ugrási cím: $A644 (42564)
A rutin megfelel a BASIC NEW parancsnak. Hatására a BASIC program törlődik (ill. eltűnik).

CLR
Ugrási cím: $A660 (42592)
A rutin megfelel a BASIC CLR parancsnak. Hatására a változók törlődnek.

RUN

Egyetlen paranccsal nem lehet megadni. Gépi kódban a RUN BASIC parancs a következőképpen néz ki:

```
JSR $A659
JSR $A7AE
```

STOP billentyű
Ugrási cím: $A82C (43052)
A rutin lekérdezi a STOP billentyűt. Ha le van nyomva, akkor a futó BASIC program félbeszakad.
Ha a STOP utasítást BASIC programban használjuk, akkor a program a $A82F (43055) címre ugrik. Ha tehát gépi kódból akarunk egy BASIC programot megszakítani, akkor erre a címre kell ugornunk.

GOTO
Ugrási cím: $A8A3 (43171)
A tulajdonképpeni GOTO rutin a $A8A0 (43168) címen kezdődik. A $A8A0 címről a rutin egy alprogramra ugrik, amely az érvényes sorszámot a $14–$15 (20–21) címre helyezi. Ha tehát a $A8A3 címre ugrunk, akkor mi határozhatjuk meg azt a sorszámot, amelyre a programnak ugrania kell. A sorszámot alsó és felső byte-ra kell bontani, és a $14–$15 címen kell elhelyezni.

Füzér kiírása
Ugrási cím: $AB1E (43806)
A rutin hatására a füzér kiíródik. Fontos, hogy a füzér utolsó karaktere 0 legyen.
A füzér címét az Y regiszterbe és az akkumulátorba kell bevinni.
Akkumulátor: alsó byte
Y regiszter: felső byte
A jobb megértés végett lássunk egy példát:

```
0 REM  ****  P105.  ****
1 :
2 :
10 LDA #$41
20 LDY #$03
30 JSR $AB1E
40 RTS
READY.
```

Hatására a kazettapuffer tartalma kiíródik az első 0-ás karakterig. A rutin segítségével pl. programneveket adhatunk meg. Mindezt BASIC-ben is megtehetjük:

```
POKE 780,65: POKE 782,3: SYS 43806
```

Néhány, a ROM-ban megtalálható füzér látható a következő táblázatban.

Jelzés	Akkumulátor (alsó byte)	Y regiszter (felső byte)	Cím
O.K.	100	163	$A364
ERROR (RETURN nélkül)	105	163	$A369
IN	113	163	$A371
READY	118	163	$A376
BREAK (RETURN nélkül)	129	163	$A381
BASIC BYTES FREE	96	228	$E460
(Bejelentkezés)	115	228	$E473

További keresgéléssel újabb jelzésekre bukkanhatunk, és ezekkel saját füzért is képezhetünk.

Egy üres karakter kiírása (ill. kurzor jobbra)
Ugrási cím: $AB3B (43835)
A rutin egy üres karaktert vagy egy kurzor jobbra jelet ír ki. Üres karaktert ír ki, ha egy file-ban vagyunk (a $13 (19) cím tartalma nem nulla).
Ez a rutin még két másikkal van kapcsolatban.

a) $AB45 (43845)
   Kérdőjel kiírása
b) $AB47 (43847)
   Az akkumulátorban levő karakter kiírása

Zárójel-ellenőrző rutin
Ugrási cím: $AEF7 (44791)
A rutin megvizsgálja azt a karaktert, amelyikre a $7A–$7B (122–123) vektor mutat, vagyis a zárójel záró végét jelző karaktert. Ha itt nem ez a jel van, akkor SYNTAX ERROR hibajelzés íródik ki, és a számítógép visszatér a parancsüzemmódba.
A rutinnal más rutinok is kapcsolatban vannak.
a) $AEFA (44794)
   ua. mint a $AEF7, csak a nyitó zárójelre
b) $AEFD (44797)
   ua. mint a $AEF7, csak a vesszőre
c) $AEFF (44799)
   ua. mint a $AEF7, csak az akkumulátorban levő karakterre

SYNTAX ERROR
Ugrási cím: $AFO8 (44808)
A rutin a SYNTAX ERROR hibajelzést írja ki, majd a számítógép visszatér a parancs-üzemmódba.

Karakter vizsgálata betűre
Ugrási cím: $B113 (45331)
A rutin megvizsgálja, hogy a címre való beugráskor az akkumulátor tartalma betű-e. Ha igen, akkor a CARRY kapcsoló törlődik.

Lebegőpontos szám integerré alakítása
Ugrási cím: $B1AA (45482)
A rutin a lebegőpontos akkumulátorban (FAC) levő számot egész típusú számmá alakítja. Ez az egész szám alsó és felső byte formájában a $64–$65 (100–101) címeken tárolódik.

BAD SUBSCRIPT ERROR
Ugrási cím: $B245 (45637)
A rutin kiírja a BAD SUBSCRIPT ERROR hibajelzést, és visszatér a parancs-üzemmódba.

ILLEGAL QUANTITY ERROR
Ugrási cím: $B248 (45640)
A rutin kiírja az ILLEGAL QUANTITY ERROR hibajelzést, és visszatér a parancs-üzemmódba.

A parancs-üzemmód tesztelése
Ugrási cím: $B3A6 (45990)
A rutin megvizsgálja, hogy a számítógép parancs-üzemmódban van-e. Ha igen, akkor ILLEGAL DIRECT ERROR hibajelzést ad.
A rutinhoz két másik rutin is kapcsolódik.

a) $B3AB (45995)
   ILLEGAL DIRECT ERROR
b) $B3AE (45998)
   UNDEF D FUNCTION ERROR

FORMULA TOO COMPLEX ERROR
Ugrási cím: $B4DO (46288)

Lebegőpontos szám alsó és felső byte-tá alakítása
Ugrási cím: $B7F7 (47095)
A rutin az akkumulátorban (FAC) levő pozitív és 65536-nál nem nagyobb lebegőpontos számot egy 16 bites számmá alakítja át. Ezt alsó és felső byte-ra

bontja, és a $14–$15 (20–21) címeken az Y regiszterbe és az akkumulátorba helyezi el.

OVERFLOW ERROR
Ugrási cím: $B97E (47486)

DEVISION BY ZERO ERROR
Ugrási cím: $BB8A (48010)

Alsó és felső byte integerré alakítása
Ugrási cím: $BDCD (48589)
A rutinnal egy alsó és felső byte formájú értéket egész számmá alakíthatunk, és kiírathatjuk.
Az alsó byte-nak az akkumulátorban, a felső byte-nak az X regiszterben kell elhelyezkedni.
Ezt a rutint BASIC-ből is jól tudjuk használni.

```
0 REM  **** P106.  ****
1 :
2 :
10 INPUT"ALSO-BYTE";LB
20 INPUT"FELSO-BYTE";HB
30 POKE781,LB:POKE780,HB:SYS48589
40 PRINT:GOTO10
READY.
```

A rutin másik két rutint használ:

a) az egész számot lebegőpontos számmá alakító rutint ($BC49);
b) a lebegőpontos számot füzérré alakító rutint ($BDDF).

BREAK ERROR
Ugrási cím: $E107 (57607)

További karakterek vizsgálata
Ugrási cím: $E211 (57873)
A rutin megvizsgálja, hogy van-e még karakter. Ha nincs, akkor SYNTAX ERROR hibajelzést ad.
Ha előbb a vesszőt akarjuk vizsgálni, akkor használjuk a $E20E (57870) ugrási címet.

BASIC hidegstart
Ugrási cím: $E394 (58260)
A rutin részleges RESET-ként működik:

a) törli a BASIC programokat és változókat;
b) a RAM a bekapcsolás utáni állapotba kerül;
c) a $0300–$030B (768–779) tartomány eredeti állapotba kerül;
d) megjelenik a kezdőkép;
e) a BASIC melegstartra ugrik.

Néhány érték azonban nem változik meg: pl. a színek és a $0314–$0333 (788–819) tartomány, ahol többek közt az IRQ és az NMI vektorok vannak.

Várakozás a COMMODORE billentyűre
Ugrási cím: $E4E0 (58592)
A számítógép a COMMODORE billentyű lenyomására vár. Ha a billentyűt nem nyomjuk le, akkor kis idő múlva a program visszaugrik a rutin elejére.

Képernyő–RESET
Ugrási cím: $E518 (58648)
A rutin visszaállítja a képernyőt. A színek az eredetiekre változnak, a képernyő törlődik, a kurzor felveszi a bekapcsolás utáni helyzetét.
A rutin lépései:

a) Képernyő–RESET a Video-Controller megváltoztatása nélkül
   ugrási cím: $E51B (58651)
b) Képernyőtörlés
   ugrási cím: $E544 (58692)
c) Kurzor alaphelyzetben
   ugrási cím: $E566 (58726)
d) Kurzor alaphelyzetben és a Video-Controller inicializálása
   ugrási cím: $E59A (58778)
e) Csak a Video-Controller inicializálása
   ugrási cím: $E5A0 (58784)

Visszalépés az előző sorra
Ugrási cím: $E701 (59137)
A rutin végrehajtásakor a kurzor egy sorral feljebb ugrik.

Képernyőgörgetés (SCROLLING)
Ugrási cím: $E8EA (59626)
A rutin hatására a képernyő egy sorral feljebb tolódik. A legfelső sor eltűnik, alul pedig egy üres sor jelenik meg.

Képernyősor törlése I.
Ugrási cím: $E9FF (59903)
A rutin hatására törlődik a képernyő egy sora. A törlendő sor számát az

X regiszterbe kell beírni. (A legfelső sor a 0 sorszámú.) A rutint BASIC-ből is használhatjuk:

POKE 781, (sorszám): SYS 59903

Képernyősor törlése II.
Ugrási cím: $E9FF (59905)
Ez a rutin is az X regiszterben tárolt sorszám szerinti sort törli. A különbség csak annyi, hogy ennél a rutinnál azt is meghatározhatjuk, hogy hányadik karakterig töröljön (0–39).
Az oszlopszámot az Y regiszterben kell tárolni.
Pl.:

```
0 REM  **** P107.  ****
1 :
2 :
10 LDA #$0A
20 LDX #$00
30 JSR $E9FF
40 RTS
READY.
```

POKE 781,0: POKE 782,10: SYS 59905

Ez a gépi kódú program és BASIC megfelelője törli a képernyő legfelső (0-s sorszámú) sorának első tíz karakterét.

1 ms-os késleltetés
Ugrási cím: $EEB3 (60958)
A rutin hatására a számítógép 1/1000 s-ot vár.

A rendszerjelzések kiírása
Ugrási cím: $F12B (61739)
A rutinnal kiírathatjuk azokat a füzéreket, amelyek a lemezegységgel, valamint a kazettás egységgel való kapcsolat során felléphetnek.
A rutin először megvizsgálja, hogy program-üzemmódban vagyunk-e.
Ha igen, akkor nem ír ki semmit.
A jelzés ofsetje az Y regiszterbe kerül.

Y regiszter dec.	hex.	Jelzés
0– 1	00–01	I/O ERROR #
12– 13	OC–OD	SEARCHING
23	17	FOR

27– 28	1B–1C	PRESS PLAY ON TAPE
46	2E	PRESS RECORD & PLAY ON TAPE
73– 74	49–4A	LOADING
81– 82	51–52	SAVING
89– 90	59–5A	VERIFYING
99–100	63–64	FOUND
106–107	6A–6B	O.K.

Ha a rutint az üzemmód vizsgálata nélkül akarjuk használni, akkor az $F12F (61743) ugrási címet kell megadni.

A SEARCHING (FOR file-név) kiírása

Ugrási cím: $F5AF (62895)

A rutin a belépésekor rögtön megvizsgálja, hogy program-üzemmódban vagyunk-e.

Ha igen, akkor azonnal visszaugrik, egyébként SEARCHING feliratot ír ki. Megvizsgálja, hogy a file-név hossza nulla-e ($B7 tároló). Ha igen, akkor a rutin végetért; ha nem, akkor kiíródik a FOR jelzés és a file-név ($BB–$BC).

Pl.:

```
POKE 183,2: POKE 187,39: POKE 188,241: SYS 62895
```

Ennek hatására a SEARCHING FOR O.K. felirat jelenik meg, mivel a 187–188-as mutatót ($BB–$BC) az O.K. rendszerjelzésre ($F127) állítottuk.

Ha nem akarjuk az üzemmódot vizsgálni, akkor az ugrási cím $F5B3 (62899).

A LOADING és a VERIFYING

Ugrási cím: $F5D2 (62930)

A rutin hatására a LOADING, ill. a VERIFYING felirat íródik ki. A $93-as cím tartalmától függ, hogy melyik. $93=0 esetén a LOADING, $93=1 esetén a VERIFYING felirat lesz olvasható a képernyőn.

A SAVING kiírása

Ugrási cím: $F68F (63119)

A rutin a SAVING jelzést írja ki. Ha az üzemmódot nem akarjuk vizsgálni, akkor a $F693 (63123) címre kell ugranunk.

TOO MANY FILES

Ugrási cím: $F6FB (63227)

FILE OPEN

Ugrási cím: $F6FE (63230)

FILE NOT FOUND

Ugrási cím: $F701 (63233)

DEVICE NOT PRESENT
Ugrási cím: $F707 (63239)

NOT INPUT FILE
Ugrási cím: $F70A (63242)

NOT OUTPUT FILE
Ugrási cím: $F70D (63245)

MISSING FILE NAME
Ugrási cím: $F710 (63248)

ILLEGAL DEVICE NUMBER
Ugrási cím: $F713 (63251)

A programfej kiolvasása a szalagról
Ugrási cím: $F72C (63276)
A rutin beolvassa a szalagon levő program fejlécét (l. a *LOAD és SAVE a gépi kódú programoknál* c. fejezetben).

A programfej generálása és szalagra írása
Ugrási cím: $F76A (63338)
Ezzel a rutinnal a program fejlécét a szalagra írhatjuk.
Az alábbi címekre a következő adatokat kell beírni:

$C1–$C2 (193–194)	A program kezdőcíme
$AE–$AF (174–175)	A program végcíme
$B7 (183)	A file-név karaktereinek száma
$BB–$BC (187–188)	A file-név alsó és felső byte mutatója

Ezeket az adatokat a rutinba való beugrás előtt kell beállítani.

A file-név keresése a szalagon
Ugrási cím: $F7EA (63466)
A rutin egy megadott file-nevet megkeres a szalagon.
A file-nevet a következőképpen kell megadni:

$B7 (183)	A keresett file-név hossza (ha a következő programot keressük, akkor ezt a címet 0-ra kell állítani)
$BB–$BC (187–188)	A keresett file-név címe

A rutin hatására a számítógép addig keres a szalagon, amíg meg nem találja a program fejlécét. Ha közben szalagvég jelet talál (EOT jel), akkor megszakítja a keresést.

A kazettás egység billentyűinek lekérdezése I.
Ugrási cím: $F817 (63511)
Ez a rutin a kazettás egységet kérdezi le. Ha a billentyű le van nyomva, akkor azonnal visszaugrik, egyébként PRESS PLAY ON TAPE felszólítással figyelmeztet. A billentyű lenyomására O.K. választ kapunk.
A rutin a STOP billentyűt is lekérdezi.

A kazettás egység billentyűinek lekérdezése II.
Ugrási cím: $F82E (63534)
Ez a rutin is a billentyűket kérdezi le. Ez azonban nem ír ki semmit, hanem az Y kapcsolót állítja be.

a) A billentyű lenyomva: Y = 1
b) Nincs lenyomva: Y = 0

A kazettás egység billentyűinek lekérdezése III.
Ugrási cím: $F838 (63544)
A rutin úgy működik, mint a $F817. A különbség csak annyi, hogy ez a PRESS PLAY ON TAPE helyett PRESS RECORD & PLAY feliratot ír ki.

STOP billentyű
Ugrási cím: $F8D0 (63696)
A rutin azonnal végetér, ha a STOP billentyűt nem nyomtuk le.
Ha lenyomtuk, akkor a kazettás egység motorja leáll, az IRQ rutin ismét működik, a CARRY kapcsoló beállítódik, és csak ekkor ér véget a rutin. Fontos megjegyezni, hogy a rutinban két PLA utasítás is van. Ez azt jelenti, hogy lenyomott STOP billentyű esetén az első ugrási cím törlődik.

Szalagos üzemmód vége
Ugrási cím: $FC93 (64659)
A rutin megszakítja a kapcsolatot a kazettás egységgel. Ez azt jelenti, hogy a képernyőt visszaállítja, a kazettás egységet kikapcsolja, a Video-Controllert pedig alapállapotba helyezi.

A kazettás egység motorjának kikapcsolása
Ugrási cím: $FCCA (64714)

10. FEJEZET
KERNAL

Aki már egyszer vette a fáradtságot, és tüzetesen átnézte az operációs rendszert, tapasztalhatta, hogy a $FF81–$FFF3 (65409–65523) címeken a különböző alprogramok kezdőcímei helyezkednek el. Felmerül a kérdés, hogy a programozók miért alkották meg ezt az ugrási táblázatot?
Nem lenne egyszerűbb a rutinokra direkt módon ugrani?

Ennek a megoldásnak van egy nagy előnye: minden Commodore számítógépen ugyanez az ugrási táblázat található (pl. a $FF31 mindig a video-RESET, csak a tényleges ugrási cím változó). Így az egyik számítógépen írt programokat egyszerűen át tudjuk ültetni a másik gépre. Ez természetesen csak a táblázatba való ugrás címére vonatkozik, a táblázatban található címek eltérőek lehetnek. Ezeket a rutinokat gyakran szoktuk használni, mivel egyrészt így programjaink egy másik számítógépbe is átültethetők, másrészt ezzel a módszerrel a rutinok a felhasználó igényeinek megfelelően megváltoztathatók.
Ezt az ugrási táblázatot nevezzük KERNAL-nak.
A következő felsorolásban a legtöbb cím két részből áll: az első a KERNAL cím, amelyre nekünk kell ugranunk, a második az a cím, amelyre a végrehajtás során a program ugrik.

KERNAL cím: $FF81 (65409)
Funkció: video-RESET
Valódi ugrási cím: $FF5B
A rutin a képernyőt alapállapotba állítja. $FF5B-től egy programot találunk:

```
0 REM  **** P108.  ****
1 :
2 :
3 OPEN1,8,1,"@0:P108G"
4 OPEN2,4
5 SYS36864
6 .OPT O1,P2
7 *= $FF5B
10 JSR $E518
20 LDA $D012
30 BNE $FF5E
40 LDA $D019
50 AND #$01
60 JMP $EDDD

READY.
```

A $FF5B címen a rutin egy alprogramba ugrik, amely

a) a kurzort alapértékre állítja;
b) a kisbetű/nagybetű átváltás ismét lehetővé válik;
c) a billentyűzetpuffer hosszát 10 karakterre állítja;
d) törli a képernyőt;
e) a kurzort a képernyő bal felső sarkába viszi.

A $FF5E–$FF61 címeken megvizsgálja, hogy a képernyőre kiíródó sor, a rasztersor kiürült-e már. Ha nem, akkor megvárja ezt az állapotot.
Aztán a $D019 (53272) cím 0. bitjét a $02A6 címre beírja (az 1–7. bitek az AND $01 utasítással törlődnek, csak a 0. bit értéke lehet 1 vagy 0). Ezután egy programra ugrik, amely megvizsgálja, hogy PAL- vagy NTSC-rendszerű televízióval van-e dolga. Ezt dönti el a $02A6 cím, amelyet éppen most állítottunk be. Próbáljuk ki:

```
PRINT PEEK(53272) AND 1
```

Ha az eredmény 1, akkor a PAL-rendszerrel (16421 ciklus); ha 0, akkor az NTSC-rendszerrel (17048 ciklus) dolgozunk. Ha más eredményre jutunk, akkor valamit elrontottunk. Nyugat-Európában a PAL-rendszer (kivéve Franciaországot, ahol a SECAM-rendszer), az USA-ban az NTSC-rendszer (a PAL elődje) honos. A C 64-es gépeket mindkét földrészen használják, ezért kellett az operációs rendszer rutinjaiba ezt a lehetőséget is beépíteni. Ezzel az átváltás nehézség nélkül megoldható.

KERNAL cím: $FF84 (65412)
Funkció: a CIA-k inicializálása
Valódi ugrási cím: $FDA3
A rutin a CIA-kat (kiviteli IC-ket) helyezi alapállapotba. Ez a program is megvizsgálja, hogy PAL- vagy NTSC-rendszerrel van-e dolga, és így állítja be a ciklusokat.

KERNAL cím: $FF87 (65415)
Funkció: a RAM törlése, ill. vizsgálata
Valódi ugrási cím: $FD50
A rutin törli a 0-ás, 1-es, 2-es lapokat a $00 és $01 címek kivételével.
Ezenkívül a kazettapuffer ($B2–$B3 (178–179) mutatóját is az eredeti értékére állítja úgy, hogy a $033C (828) értéken kezdődjön. A továbbiakban megvizsgálja a RAM terület végét $0400-tól (1024).
A cím alsó és felső byte-ra bontva a $0283–$0284 (643–644) címekre kerül. Ezután beállítja a RAM kezdetet a $0800 (2048), a video-RAM kezdetet pedig a $0400 (1024) címekre. A rutint könnyen átalakíthatjuk más célra is, a segítségével egyszerűen meg tudjuk állapítani a BASIC-RAM felső határát.

eee :$FE20
X regiszter: alsó byte
Y regiszter: felső byte

A felső határ másik kiolvasási módja:
eee : $FE27
X regiszter : alsó byte
Y regiszter : felső byte

KERNAL cím: $FF8A (65418)
Funkció: az I/O inicializálása
Valódi ugrási cím: $FD15
A rutin az I/O egységet állítja alapállapotba.

KERNAL cím: $FF8D (65421)
Funkció: I/O vektorok inicializálása
Valódi ugrási cím: $FD1A
A rutin a $0314–$0333 címeket (788–819) alapállapotba állítja.

KERNAL cím: $FF90 (65424)
Funkció: a státusz beállítása
Valódi ugrási cím: $FE18
A rutin beállítja a státuszt:

```
0 REM  **** P109.  ****
1 :
2 :
3 OPEN1,8,1,"00:P1090"
4 OPEN2,4
5 SYS38864
6 .OPT O1,P2
7 ** $FE18
10 STA $9D    ;A PARANCS-UZEMMOD KAPCSOLOJA
20 LDA $90    ;($80=PARANCS-, $00=PROGRAM-UZEMMOD)
30 ORA $90
40 STA $90
50 RTS

READY.
```

KERNAL cím: $FF93 (65427)
Funkció: másodlagos cím küldése, LISTEN
Valódi ugrási cím: $EDB9
A rutin a másodlagos címet kiadja az IEC-(soros) buszra. Az IRQ megszakad, a másodlagos cím az akkumulátor tartalma lesz. LISTEN jelet küld arra a lemezegységre, amellyel kapcsolatba akar lépni. A rutin részleteit később ismertetjük.

KERNAL cím: $FF96 (65430)
Funkció: másodlagos cím küldése, TALK
Valódi ugrási cím: $EDC7
A rutin ugyanúgy működik, mint a $FF93-as rutin. A különbség csak annyi, hogy ez a másodlagos címet arra a lemezegységre küldi ki, amelyről adatot vár.

KERNAL cím: $FF99 (65433)
Funkció: a RAM terület végének beállítása, olvasása
Valódi ugrási cím: $FE25
A rutin beállítja (CARRY kapcsoló = 0) vagy kiolvassa (CARRY kapcsoló = 1) a RAM terület végét. A regiszterek tartalma mindkét esetben azonos.

X regiszter: alsó byte
Y regiszter: felső byte

Használatuk (írás vagy olvasás) a CARRY kapcsoló értékétől függ.

KERNAL cím: $FF9C (65436)
Funkció: a RAM terület kezdetének beállítása, olvasása
Valódi ugrási cím: $FE34
A rutin ugyanúgy működik, mint a $FF99-es rutin. A különbség csak annyi, hogy ez a RAM terület kezdetével dolgozik.

KERNAL cím: $FF9F (65439)
Funkció: a billentyűzet lekérdezése
Valódi ugrási cím: $EA87
A rutin lekérdezi a billentyűzetet, átkódolja, és a megfelelő karaktert kiírja. A rutint elsősorban a $EF87-et) az IRQ rutin is használja.

KERNAL cím: $FFA2 (65442)
Funkció: az IEC-busz time-out (időki) kapcsolójának beállítása
Valódi ugrási cím: $FE21
A rutin beállítja a time-out kapcsoló (ki) értékét a $0285 (645) címen.

```
0 REM  **** P110.  ****
1 :
2 :
3 OPEN1,8,1,"@0:P1100"
4 OPEN2,4
5 SYS36864
6 .OPT O1,P2
7 ** $FE21
10 STA $0285
20 RTS
```

Mint látható, a kapcsoló értékét az akkumulátorba kell beírni.

KERNAL cím: $FFA5 (65445)
Funkció: bevitel az IEC-buszról (IECIN)
Valódi ugrási cím: $EE13
A rutin beolvas egy karaktert a lemezegységről. A művelethez előkészítő rutinokra is szükség van.

KERNAL cím: $FFA8 (65448)
Funkció: kivitel az IEC-buszra (IECOUT)
Valódi ugrási cím: $EDDD
A rutin kiküld egy karaktert (ATN jellel együtt) az IEC-buszra. A $FFA5 rutinhoz hasonlóan itt is szükség van előkészítő rutinokra.

KERNAL cím: $FFAB (65451)
Funkció: UNTALK küldése
Valódi ugrási cím: $EDEF
A rutin egy UNTALK jelet küld ki az I/O egységre, amellyel éppen kapcsolatban van. Hatására a számítógép felé irányuló adatáram megszakad.

KERNAL cím: $FFAE (65454)
Funkció: UNLISTEN küldése
Valódi ugrási cím: $FDFE
A rutin hatása megegyezik az UNTALK rutinéval ($FFAB), de csak arra az egységre vonatkozik, amely a számítógéptől éppen adatokat kap.

KERNAL cím: $FFB1 (65457)
Funkció: LISTEN küldése
Valódi ugrási cím: $EDOC
A rutin az UNLISTEN rutin ellentetje. Hatására kezdődik a kapcsolattartás a kiválasztott egységgel. Ehhez a rutin hívása előtt az akkumulátorba be kell tölteni a lemezegység egységszámát, majd tárolni kell a $BA (186) címen. Ha tehát egy lemezegységre adatokat akarunk kiküldeni, akkor a program első három sora:

```
0 REM  ****  P111.  ****
1 :
2 :
10 :      LDA #$08        ;8=A FLOPPY EGYSEGSZAMA
20 :      STA $BA         ;TAROLAS
30 :      JSR $FFB1       ;LISTEN RUTIN

READY.
```

KERNAL cím: $FFB4 (65460)
Funkció: TALK küldése
Valódi ugrási cím: $EDO9
A rutin a LISTEN rutinhoz hasonlóan működik, csak ez már a közvetlen adatkül-

désre hívja fel a lemezegység figyelmét. Mindkét rutin (a TALK és a LISTEN) ugyanazt a rutint használja.

```
0 REM  ****  P112.  ****
1 :
2 :
3 OPEN1,8,1,"@0:P112G"
4 OPEN2,4
5 SYS36864
7 *= $ED09
10 ORA #$40
20 .BYTE $2C
30 ORA #$20
40 JSR $F0A4

READY.
```

A TALK esetén a $EDO9, a LISTEN esetén a $EDOC az ugrási cím.
Ha a TALK rutinra ugrunk, akkor az egységszám és a 64 összehasonlítódik logikai VAGY utasítással, vagyis az egységszámban (amely a 0–3 bitet foglalja le) a 6. bit beállítódik. Ezután egy programozási trükkel a $EDOC sort átugorhatjuk, amelyben az 5. bit is beállítódna.
Ha a LISTEN rutinra ugrunk, akkor logikai VAGY utasítással az egységszám és a 32 hasonlítódik össze, így az 5. bit fog beállítódni.
A $EDOE címtől a két rutin megegyezik egymással.
Most már tudjuk, hogy a lemezegység a beállított

a) 5. bittel adatokat fogad a készülékről (LISTEN);
b) 6. bittel adatokat küld a készülékre.

KERNAL cím: $FFB7 (65463)
Funkció: a státusz beolvasása
Valódi ugrási cím: $FEO7
A rutin beolvassa a státuszt az akkumulátorba, és 0-ra állítja. Ha a $BA (186) egységszám 2, akkor az RS–232 kimenetre is használható.

KERNAL cím: $FFBA (65466)
Funkció: a file-paraméterek beállítása
Valódi ugrási cím: $FEOO
A rutin beállítja a megadott file összes paraméterét. A logikai file-számot, az egységszámot és a másodlagos címet az alrutinba kell beírni.

Paraméter	Regiszter	Tárolási cím
Logikai file-szám	Akkumulátor	$B8 (184)
Egységszám	X regiszter	$BA (186)
Másodlagos cím	Y regiszter	$B9 (185)

KERNAL cím: $FFBD (65469)
Funkció: a file-név paraméterek beállítása
Valódi ugrási cím: $FDF9
A rutin a file-név összes paraméterét beállítja.

Paraméter	Regiszter	Tárolási cím
A név hossza	Akkumulátor	$B7 (183)
Cím alsó byte	X regiszter	$BB (187)
Cím felső byte	Y regiszter	$BC (188)

A $FFBA KERNAL rutinhoz hasonlóan a megfelelő értékeket az akkumulátorba, az X és Y regiszterbe kell írni.

KERNAL cím: $FFCO (65472)
Funkció: OPEN
Valódi ugrási cím: a $031A–$031B címekből kiolvasható.
Alapállapotban: $F34A
Az utasítás fontossága miatt a rutint teljes részletességgel közöljük, és magyarázzuk:

```
0 REM  **** P113. ****
1 :
2 :
5 SYS36864
6 .OPT OO,P1
7 *= $F34A
10 LDX $B8        ;A LOGIKAI FILE-SZAM BETOLTESE
20 BNE $F351      ;NEM NULLA, TEHAT TOVABB
30 JMP $F70A      ;"NOT INPUT FILE" KIADASA
40 JSR $F30F      ;MEGVAN A LOGIKAI FILE-SZAM
50 BNE $F359      ;NINCS MEG
60 JMP $F6FE      ;"FILE OPEN" KIADASA
70 LDX $98        ;A NYITOTT FILE-OK SZAMA
80 CPX #$0A       ;10-ZEL OSSZEHASONLITJA
90 BCC $F362      ;KISEBB MINT 10, TEHAT TOVABB
100 JMP $F6FB     ;TUL NAGY, TEHAT "TOO MANY FILES"
110 INC $98       ;A SZAM NOVELESE 1-GYEL
120 LDA $B8       ;LOGIKAI FILE-SZAM
130 STA $0259,X   ;TAROLJA A TABLAZATBAN
140 LDA $B9       ;MASODLAGOS CIM
150 ORA #60       ;5. 6. BIT A LEMEZEGYSEGRE
160 STA $B9       ;UJRA TAROLAS
170 STA $026D,X   ;A MASODLAGOS CIM TAROLASA A TABLAZATBAN
180 LDA $BA       ;EGYSEGSZAM
190 STA $0264,X   ;TAROLAS A TABLAZATBAN
200 BEQ $F3DB     ;BILLENTYU, TEHAT VISSZA
210 CMP #$03      ;KEPERNYO
220 BEQ $F3D3     ;IGEN, TEHAT VISSZA
230 BCC $F384      ;NINCS FILE AZ IEC-BUSZRA
240 JSR $F3D5     ;FILE-NYITAS AZ IEC-BUSZRA
250 BCC $F3D3     ;KESZ
260 CMP #$02      ;RS 232
270 BNE $F38B     ;NEM, TEHAT SZALAG
```

```
280 JMP F409     ;RS 232 OPEN
290 JSR $F7D0    ;A KAZETTAPUFFER-KEZDOCIM BEOLVASASA
300 BCS $F393    ;KAZETTA MEGNYITAS TOVABB
310 JMP $F713    ;"ILLEGAL DEVICE NUMBER" KIIRASA
320 LDA $B9      ;MASODLAGOS CIM
330 AND #$0F     ;A 4-7 BIT TORLESE
340 BNE $F388    ;MASODLAGOS CIM=0, TEHAT IRAS
350 JSR $F817    ;A PLAY BILLENTYU LEKERDEZESE
360 BCS $F3D4    ;A STOP BILLENTYU LENYOMVA, TEHAT MEGSZAKITAS
370 JSR $F5AF    ;"SEARCHING (FOR NAME)" KIIRASA
380 LDA $B7      ;A FILE-NEV HOSSZA
390 BEQ $F3AF    ;NINCS FILE-NEV, TEHAT TOVABB
400 JSR $F7EA    ;A KIVANT SZALAGFEJ KIVALASZTASA
410 BCC $F3C2    ;MEGTALALTA
420 BEQ $F3D4    ;MEGSZAKITAS
430 JMP $F704    ;EOT, VAGYIS "FILE NOT FOUND"
440 JSR $F72C    ;TOVABB KERES
450 BEQ $F3D4    ;EOT, TEHAT VEGE
460 BCC $F3C2    ;MEGTALALTA
470 BCS $F3AC    ;TOVABB KERES
480 JSR $F838    ;VAR A RECORD & PLAY BILLENTYURE
490 BCS $F3D4    ;A STOP LENYOMVA, TEHAT MEGSZAKITAS
500 LDA #$04     ;SZALAGFEJ-ELLENORZO BYTE
510 JSR $F7A6    ;A FEJ FELIRASA A SZALAGRA
520 LDA #$BF     ;MUTATO A KAZETTAPUFFER VEGERE
530 LDY $B9      ;MASODLAGOS CIM
540 CPY #$60     ;OSSZEHASONLITAS 96-TAL (5. 6. BIT)
550 BEQ $F3D1    ;A MASODLAGOS CIM=0, TEHAT TOVABB
560 LDZ #$00
570 LDA #$02     ;ADATBLOKK KONTROLL BYTE
580 STA (#B2),Z  ;IRAS A KAZETTAPUFFERBE
590 TZA          ;AKKU = 0
600 STA $A6      ;KAZETTAPUFFER MUTATO
610 CLC
620 RTS
630 FILE-NYITAS AZ IEC-BUSZRA

READY.
```

A rutin olvasásakor bizonyára feltűnt, hogy az OPEN rutin néhány paramétert (file-név és file) már eleve beállítottnak feltételez.

1. File-név

a) hossz: $B7 (183)
b) alsó byte $BB (187)
c) felső byte $BC (188)

2. File

a) logikai file-szám: $B8 (184)
b) másodlagos cím: $B9 (185)
c) egységszám: $BA (186)

Ezekhez a KERNAL-ban már két rutint találtunk:

a) a file-paraméterek beállítása: $FFBA
b) a file-név paraméterek beállítása: $FFBD

Az OPEN rutin előtt ezt a két rutint kell meghívni.
Lássunk egy példát: meg akarunk nyitni egy $ (tartalomjegyzék) nevű file-t a lemezegységen:

```
0 REM  **** P113/1. ****
1 :
2 :
10 :      LDA #$01    ;A FILE-NEV HOSSZA
20 :      LDX #$D0    ;CIM ALSO BYTE
30 :      LDY #$FF    ;CIM FELSO BYTE
40 :      JSR $FFBD   ;A FILE-NEV PARAMETEREINEK BEALLITASA
50 :
60 :-------------------------------------------------------
70 :
80 :      LDA #$01    ;LOGIKAI FILE-SZAM
90 :      LDY #$00    ;MASODLAGOS CIM
100 :     LDX #$08    ;EGYSEGSZAM
110 :     JSR $FFBA   ;A FILE-PARAMETEREK BEALLITASA
120 :
130 :-------------------------------------------------------
140 :
150 :     JSR $FFC0   ;OPEN

READY.
```

Magyarázat: a filenév címe $FFDO. Ezen a címen decimálisan 36 áll, mivel ez a $ jel ASCII-kódja.

KERNAL cím: $FFC3 (65475)
Funkció: CLOSE
Valódi ugrási cím: a $031C–$031D (796–797) vektor tartalma.
Alapállapotban: $F291
Ez a fontos rutin az OPEN rutin párja. A CLOSE rutin használatakor azonban csak egy paramétert kell megadni: a logikai file-számot. Erre valószínűleg mindenki emlékszik a BASIC-ből. A BASIC-ben csak a CLOSE 1 utasítást kell kiadni, ha file 1-es logikai file-számmal lett megnyitva. Az OPEN rutinra ugrás előtt az akkumulátorban levő logikai file-számot tárolnunk kell. Ha a 10-es logikai file-számmal akarunk egy file-t lezárni, akkor gépi kódban a következőképpen kell eljárni:

```
0 REM  ****  P114.  ****
1 :
2 :
10 :      LDA #$0A
20 :      JSR $FFC3
30 :      RTS

READY.
```

A rutinhoz nem kell magyarázat. A felhasználó számára csak az a fontos, hogy működjön. Az már több szót érdemel, hogy honnan veszi a számítógép a többi paraméter értékét (pl. egységszám stb.).
A rutin folyamatosan figyeli az éppen nyitott file-okat a $98-as (158) címen. A többi érték táblázatokban található.

601–610 A logikai fileszámok táblázata
611–620 Az egységszámok táblázata
621–630 A másodlagos címek táblázata

Mindhárom táblázat kezdeti értéke 0.
Tételezzük fel, hogy megnyitunk egy file-t 1-es logikai file-számmal, 0 másodlagos címmel és 8-as egységszámmal. Ezzel a nyitott file-ok száma automatikusan 1-gyel nő, tehát a $98 (158) címen 1 található. A három táblázat pedig a következőképpen néz ki:

	1	2	3	4	5	6	7	8	9	10	
601	1	0	0	0	0	0	0	0	0	0	file-név
611	0	0	0	0	0	0	0	0	0	0	másodlagos cím
621	8	0	0	0	0	0	0	0	0	0	egységszám

Tételezzük fel, hogy megnyitunk egy újabb file-t 3-as logikai file-számmal, 3-as másodlagos címmel, 1-es egységszámmal:

	1	2	3	4	5	6	7	8	9	10	
601	1	3	0	0	0	0	0	0	0	0	file-szám
611	0	1	0	0	0	0	0	0	0	0	másodlagos cím
621	8	1	0	0	0	0	0	0	0	0	egységszám

A többi file is hasonló elven tárolódik.
Ha tehát a számítógépnek szüksége van a 3-as logikai file-számú file többi paraméterére, akkor először végignézi a 601–610-es táblázatot. Egy számláló minden adat megvizsgálása után eggyel nő. Ha megvan az érték, akkor a számítógép megnézi a számláló értékét. Ezen a helyen keresi a másik két táblázatban a megfelelő paramétereket.

KERNAL cím: $FFC6 (65478)
Funkció: a beviteli egység beállítása (CHKIN)
Valódi ugrási cím: a $01E–$031F (798–799) vektor tartalma.
Alapállapotban: $FEOE
A rutin beállítja a beviteli egységet egy file segítségével. A logikai file-számot az X regiszterbe kell írni. Ha nincs ilyen számú nyitott file, akkor FILE NOT OPEN hibajelzést kapunk.
Ha talált ilyen számot, akkor a megfelelő egységszámot kikeresi a táblázatból, és beírja a $99-es (153) címre. Ha egy lemezegységet, vagyis az IEC-buszt meg

akarjuk szólítani, akkor először még egy TALK jelet kell kiküldeni. Ezután következik a státusz vizsgálata. Ha ez nincs rendben, akkor DEVICE NOT PRESENT hibajelzést kapunk. A készülék nem válaszolt, következésképpen nincs is a rendszerben.
Ha a kazettás egységet beviteli egységként állítjuk be (1-es egységszám) és a másodlagos cím nem 0, akkor NOT INPUT FILE hibajelzést kapunk, mivel beviteli egységnek nem adhatunk meg kimeneti file-t (másodlagos címnek 1-et feltételez).
Erre a rutinra akkor is ugorhatunk, ha nem nyitottunk meg semmilyen file-t. Ilyen esetben azonban a táblázatba közvetlenül be kell írni a megfelelő értékeket (egységszám, másodlagos cím) a $BA (186) és a $B9 (197) címekre. Az ugrási cím ilyen esetben: $F219 (61977)
A beviteli egység száma a $99 (153) címre kerül.

KERNAL cím: $FFC9 (65481)
Funkció: a kiviteli egység beállítása (CKOUT)
Valódi ugrási cím: a $0320–$0321 (800–801) vektor tartalma.
Alapállapotban: $F250
A logikai file-számot ennél a rutinnál is az X regiszterbe kell írni.
Ha nincs ilyen számú file, akkor FILE NOT OPEN hibajelzést kapunk.
Ha a kazettás egységen, mint kiviteli egységen nyitottunk meg egy file-t, és a másodlagos cím 0 (vagyis olvasás), akkor a NOT OUTPUT FILE hibajelzést kapjuk.
Ezt a rutint is használhatjuk anélkül, hogy egy file-t meg kellene nyitnunk.

Adjuk meg a kívánt paramétereket (egységszám, másodlagos cím) a $BA (186) és a $B9 (187) címeken, és ugorjunk a rutin $F25B (62043) címére.
Amikor a rutinban megszólítjuk a lemezegységet, akkor egy LISTEN jelet küldünk, és lekérdezzük a státuszt. Nem megfelelő eredmény esetén DEVICE NOT PRESENT hibajelzést kapunk.
A kiviteli egység száma a $9A (154) címre kerül.

KERNAL cím: $FFCC (65484)
Funkció: az I/O visszaállítása (CLRCH)
Valódi ugrási cím: a $0322–$0323 (802–803) vektor tartalma.
Alapállapotban: $F333
A rutinnal az IEC-buszon lezárhatunk egy aktív I/O-csatornát.

```
0 REM  **** P115. ****
1 :
2 :
5 SYS36864
6 .OPT OO,P1
7 ** $F333
10 LDX #$03    ;3=KEPERNYO
20 CPX $9A     ;A KIVITELI EGYSEG SZAMA
```

```
30 BCS $FB3C   ;KISEBB, MINT 3
40 JSR $EDFE   ;UNLISTEN KULDESE A LEMEZEGYSEGNEK
50 CPX $99     ;A BEVITELI EGYSEG SZAMA
60 BCS #F343   ;KISEBB, MINT 3
70 JSR $EDEF   ;UNTALK KULDESE A LEMEZEGYSEGNEK
80 STX $9A     ;KIVITEL ISMET A KEPERNYORE
90 LDA #$00    ;0=BILLENTYUZET
100 STX $99    ;BEVITEL ISMET A BILLENTYUZETRUL
110 RTS
```

A rutin a következőképpen működik: először megnézi, hogy az IEC-busz-e a kiviteli egység. Ha igen, akkor az adatforgalmat megszakító rutinra ugrik (UNLISTEN). Ha nem, akkor megvizsgálja, hogy a beviteli egység a lemezegység-e. Ha igen, akkor küld egy UNTALK jelet a lemezegységnek, ami arra figyelmezteti a lemezegységet, hogy nemsokára adatokat kell küldenie a számítógépnek. Végül mindkét lehetőség esetén (a lemezegység mint beviteli egység vagy mint kiviteli egység) a képernyőt kiviteli, a billentyűzetet beviteli egységre írja vissza.

KERNAL cím: $FFCF (65487)
Funkció: egy karakter bevitele (BASIN)
Valódi ugrási cím: a $0324–$0325 (804–805) vektor tartalma.
Alapállapotban: $F157

A rutinnal tetszőleges beviteli egységről beolvashatunk egy karaktert. A beolvasott karakter a visszaugrás után az akkumulátorban található. Mivel ez a rutin egy kicsit bonyolult, lássunk néhány példát! Egy füzért akarunk egy tartományban (legyen most $033C-től) elhelyezni. A füzért a billentyűzetről kell bevinni. A programot a $6000-es (24576) címtől helyezzük el a tárolóban.

```
0 REM  **** P116. ****
1 :
2 :
10 :      LDX #$00          ;A SZAMLALO NULLAZASA
20 :    * JSR $FFCF         ;BASIN (KARAKTER BEVITELE)
30 :      STA $033C         ;A KARAKTER TAROLASA
40 :      INX               ;A SZAMLALO NOVELESE
50 :      CMP #$0D          ;RETURN
60 :      BNE *             ;NEM
70 :      RTS               ;IGEN
```

A BASIC betöltőprogram:

```
0 REM  ****  P117.  ****
1 :
2 :
10 FORY=0TO13:READA:POKE24576+Y,A:NEXTY:SYS24576
20 DATA162,0,32,207,255,157,60,3,232,201,13,208,245,96

READY.
```

A program bevitele és a RUN parancs után a képernyön megjelenik a kurzor. Bevihetjük a karaktereket. A RETURN lenyomásával a program félbeszakad. Ellenőrizzük, hogy jól tároltuk-e a füzért: L a bevitt füzér hossza. Ez az érték a 192-t nem haladhatja meg, mivel akkor a kazettapuffer megtelik.
Írjuk be a következő utasításokat:

```
FORY=0 TOL: PRINTCHR$(PEEK(828+Y));: NEXTY
```

Ha mindent jól csináltunk, akkor a bevitt karakterek (a kurzorvezérlők is) megjelennek a képernyőn. A rutin nemcsak a billentyűzet segítségével dolgozik. Ugyanilyen jól lehet az adatokat átvenni a lemezegységről, a kazettás egységről vagy az RS–232-esről. Ehhez azonban meg kell változtatnunk a beviteli egységszámot a következők szerint.
Tételezzük fel, hogy adatokat akarunk átvenni a már várakozó lemezegységről. Ez a folyamat több lépésből áll.

a) A paraméterek beállítása:

```
LDA #$01    logikai file-szám
LDX #$08    egységszám
LDY #$00    másodlagos cím
JSR $FFBA   a paraméterek beállítása
```

b) A file-név paraméterek beállítása:

```
LDA #$XX    a név hossza
LDX #$Y1    cím alsó byte
LDY #$Y2    cím felső byte
JSR $FFBD   a paraméterek beállítása
```

c) OPEN: JSR $FFCO OPEN rutin

d) A beviteli egység beállítása:

```
LDX #$01    logikai file-szám
JSR $FFC6   CHKIN
```

e) Egy adat bevitele a megnyitott file-ból:

```
BASIN
```

f) Nincs még minden adat behozva? Vissza az e) pontra

g) A standard értékek visszaállítása:

```
JSR $FFCC   CLRCH
```

h) A file lezárása:

```
LDA #$01    logikai file-szám
JSR $FFC3   CLOSE
```

Reméljük, ezek után nem okoz gondot a többi file-típus adatainak beolvasása sem.
A *Lemezegység trükkök* c. fejezetben erre is található egy példa.

KERNAL cím: $FFD2 (65490)
Funkció: egy karakter kiírása (BSOUT)
Valódi ugrási cím: a $0326–$0327 (806–807) vektor tartalma.
Alapállapotban: $F1CA
Ahogyan egy karaktert be tudunk hozni a BASIN rutinnal, úgy ki is tudjuk vinni a BSOUT rutinnal. A rutin működését egy példával világítjuk meg:

```
0 REM  **** P118.  ****
1 :
2 :
10 :       LDX #$00          ;A SZAMLALO NULLAZASA
20 :    * LDA $F0BE,X        ;A KARAKTER BEOLVASASA
30 :       JSR $FFD2         ;ES KIIRASA (BSOUT)
40 :       INX               ;A SZAMLALO NOVELESE
50 :       CPX #$09          ;MAR 9
60 :       BNE *             ;NEM
70 :       RTS               ;IGEN
```

Ez a rutin egyenként beolvassa a $FOBE–$FOC6 (61630–61638) tartomány tartalmát (az operációs rendszer I/O ERROR hibajelzése) az akkumulátorba, és a BSOUT rutin segítségével kiírja a képernyőre.
A BASIC betöltőprogram:

```
0 REM  ****  P119.  ****
1 :
2 :
10 FORX=0TO13:READA:POKE24576+X,A:NEXTX:SYS24576
20 DATA162,0,189,190,240,32,210,255,232,224,9,208,245,96

READY.
```

A RUN utasítás hatására a képernyőn megjelenik a I/O ERROR felirat. A példából is látható, hogy a BSOUT rutin könnyen kezelhető, valamint a BASIN rutinhoz hasonlóan más egységekhez is használható.
A programlépések megegyeznek a BASIN rutinéval, a különbség csak annyi, hogy

a) a másodlagos cím értéke 1 lehet;
b) a CHKIN rutin helyett a CKOUT rutint kell használni, mivel itt kimeneti egységgel dolgozunk;
c) a BASIN rutin helyett a BSOUT rutinra kell ugrani.

Mindkét rutin nagyon gyors és gyakran használható, mivel az összes I/O műveletnél működtethető.

KERNAL cím: $FFD5 (65493)
Funkció: LOAD
Valódi ugrási cím: $F49E
Ez a LOAD rutin. A rutin hívása előtt több paramétert be kell állítani.
A beállítandó paraméterek a következők:

a) $BA (186)	egységszám
b) $B7 (183)	a file-név hossza (a 8-as egységszám esetén feltétlenül meg kell adni, egyébként 0 is lehet)
c) $B9 (185)	másodlagos cím
d) $BB–$BC	a file-névre mutató alsó és felső byte ($B7 = 0 esetén nem kötelező)
e) X regiszter	a kezdőcím alsó byte-ja (a $C3-ba kerül)
f) Y regiszter	a kezdőcím felső byte-ja (a $C4-be kerül)
g) Akku	LOAD–VERIFY kapcsoló (0 = Load, 1 = Verify) a $93-ba kerül

A LOAD rutin $F4A2 címén egy indirekt ugrás valósul meg:

```
JMP ($0330)
```

Ez a vektor alapállapotban a $F4A5-ös címre mutat, de mutathat egy saját Load rutin címére is (l. a *Tippek és trükkök házi használatra* c. fejezet).

KERNAL cím: $FFD8 (65496)
Funkció: SAVE
Valódi ugrási cím: $F5DD
A SAVE rutin hívása előtt is be kell állítani a megfelelő paramétereket:

a) $AE–$AF	végcím
b) $C1–$C2	kezdőcím
c) $BA	egységszám
d) $B9	másodlagos cím (IEC-mentésnél értéke automatikusan 1)
e) $B7	a file-név hossza (az IEC mentésig lehet 0)
f) $BB–$BC	a file-név címe (ha $B7 = 0, szükségtelen)

A rutin a kezdőcím és a végcím között található byte-sorozatot kiviszi a megfelelő készülékre.

KERNAL-cím: $FFDB (65499)
Funkció: az idő beállítása
Valódi ugrási cím: $F6E4
Az akkumulátor, az X és az Y regiszter beállítja az órát. Az idő a $A0–$A2 (160–162) címeken található (éspedig a TI változó. PEEK(160) * 65525 + PEEK (161) * 256 + PEEK(162)). A rutin segítségével tetszőlegesen be tudjuk állítani az időt.

Akku	$A2-be (162, az óra alsó byte-ja)
X regiszter	$A1-be (161, az óra középső byte-ja)
Y regiszter	$A0-ba (160, az óra felső byte-ja)

KERNAL cím: $FFDE (65502)
Funkció: az idő beolvasása
Valódi ugrási cím: $F6DD
A rutin végrehajtása után a regiszterek tartalma a következő:

Regiszter	Cím	Paraméter
Akkumulátor	$A2 (162)	idő alsó byte
X regiszter	$A1 (161)	idő középső byte
Y regiszter	$A0 (160)	idő felső byte

KERNAL cím: $FFE1 (65505)
Funkció: a STOP billentyű lekérdezése
Valódi ugrási cím: a $0328–$0329 (808–809) vektor tartalma
Alapállapotban: $F6ED
A rutin csak az IRQ rutinnal működik. Az IRQ rutinból egy ugrás hajtódik végre a KERNAL alprogramra, amely lenyomott STOP billentyű esetén beállít egy kapcsolót ($91 (145)). Ez a KERNAL alprogram csak a kapcsolt értéket kérdezi le. Ha a kapcsoló értéke megfelelő ($7F(127)), akkor csak a CLRCH ($FFCC) rutinnal kérdezhetünk le. Közben a billentyűzetpuffer kiürül, a $C6 (198) cím tartalma pedig 0 lesz.
Ha az IRQ rutin STOP billentyűt lekérdező részét egy POKE 788,52 utasítással üzemen kívül helyezzük, akkor a rutin nem működik tovább.

KERNAL cím: $FFE4 (65508)
Funkció: GETIN
Valódi ugrási cím: a $032A–$032B vektor tartalma.
Alapállapotban: $F13E
A rutin behív egy karaktert.
A $99 (153) cím (aktív beviteli egység) tartalmának függvényében a számítógép különböző rutinokra ugrik.

$99 (153)	Beviteli egység	Ugrási cím
0	billentyűzet	$E5B4
2	RS–232-es illesztő	$F086
más	lemezegység, kazettás egység stb.	$F166

Ha a karaktert a billentyűzetről hozzuk be, akkor a rutin listája:

```
0 REM  ****  P120.  ****
1 :
2 :
3 OPEN1,8,1,"@0:P120G"
4 OPEN2,4
5 SYS36864
6 .OPT O1,P2
7 *= $E584
10 LDY $0277      ;AZ Y REGISZTER ELSO KARAKTERE
20 LDX #$00       ;SZAMLALO
30 LDA $0278,X    ;A KARAKTER BEHUZASA
40 STA $0277,X    ;ES TAROLASA EGY HELLYEL ELOBB
50 INX            ;A SZAMLALO NOVELESE
60 CPX #$06       ;A KARAKTEREK SZAMA
70 BNE $E589      ;MEG NINCS VEGE, TEHAT TOVABB
80 DEC $C6        ;A KARAKTERSZAM CSUKKENTESE EGGYEL
90 TYA            ;1. KARAKTER AZ AKKUBA
100 CLI           ;A MEGSZAKITAS ENGEDELYEZESE
110 CLC
120 RTS

READY.
```

A rutin behozza az első karaktert a billentyűzetpufferből (631–640), a többit eggyel előre hozza, a számukat pedig eggyel csökkenti.
A megszakítást kikapcsoltuk, mert a billentyűzetpuffer megváltoztatása hosszú ideig tart (az IRQ rutin pedig lekérdezi a billentyűzetet).

KERNAL cím: $FFE7 (65511)
Funkció: az összes nyitott file lezárása (CLALL)
Valódi ugrási cím: a $032C–$032D (812–813) vektor tartalma.
Alapállapotban: $F32F
Ez a rutin egy másik KERNAL rutint használ, a CLRCH rutint ($FFCC). Csak két utasítást igényel:

```
0 REM  ****  P121.  ****
1 :
2 :
3 OPEN1,8,1,"@0:P121G"
4 OPEN2,4
5 SYS36864
6 .OPT O1,P2
7 *= $F32F
10 LDA #$00
20 STA $98

READY.
```

Hatására lezáródik az összes nyitott file ($98 (158) a nyitott file-ok száma). A $F333-as címtől a CLRCH rutinnal folytatódik a végrehajtás.

KERNAL cím: $FFEA (65514)
Funkció: az idő növelése, a STOP billentyű lekérdezése
Valódi ugrási cím: $F69B
Ezt a rutint is az IRQ rutin használja, amely közvetlenül a $F69B címre ugrik. Ez a rutin növeli az időt. Ezen kívül megvizsgálja, hogy a mért idő elérte-e már a 24 órát. Ha igen, akkor az órát nullázza.
Ezután megvizsgálja, hogy a STOP billentyű le van-e nyomva. Ha igen, akkor beállítja a $91 (145) kapcsolót.
Mivel ezt a rutint az IRQ rutin használja, előfordulhat, hogy a rutin csak a STOP billentyűt kérdezi le, és nem ugrik el egy másik rutinra. Ennek van egy nagy előnye.
Aszerint, hogy mi hajtódott végre, elugorhatunk egy másik, a felhasználó által készített rutinra. Ha pl. a számítógép töltés-üzemmódban van, és lenyomjuk a STOP billentyűt, akkor egy másik rutinra ugrik, mintha a számítógép program-üzemmódban lenne.

KERNAL cím: $FFED (65517)
Funkció: a sorok és oszlopok számának beolvasása
Valódi ugrási cím: $E505
A rutin az oszlopok számát (40) az X regiszterbe, a sorok számát (25) az Y regiszterbe írja:

```
0 REM  **** P122.  ****
1 :
2 :
5 SYS36864
6 .OPT OO,P1
7 *= $E505
10 LDX #$28     ;OSZLOPSZAM
20 LDY #$19     ;SORSZAM
30 RTS

READY.
```

KERNAL cím: $FFF0 (65520)
Funkció: a kurzor beállítása, a kurzorpozíció beolvasása
Valódi ugrási cím: $E50A
Ha a CARRY kapcsoló be van állítva, akkor a rutin beolvassa a kurzorpozíciót, egyébként pedig beállítja a kurzort.

```
0 REM  **** P123.  ****
1 :
2 :
5 SYS36864
6 .OPT OO,P1
7 *= $E50H
10 BCS $E513    ;BEOLVASAS
20 STX $D6      ;SOR
```

```
30 STY $D3        ;OSZLOP
40 JSR $E56C      ;KURZOR POZICIONALAS
50 LDX $D6
60 LDY $D3
70 RTS

READY.
```

Amint látható,

a) a kurzorpozíció beolvasásakor a sorszám az X, az oszlopszám az Y regiszterbe kerül;
b) a kurzor beállításakor ugyanez a helyzet.

KERNAL cím: $FFF3 (65523)
Funkció: az I/O egység kezdőcímének beolvasása
Valódi ugrási cím: $E500
A rutin végrehajtása után az X regiszterben az I/O egység kezdőcímének alsó ($00), az Y regiszterben a felső byte-ja ($DC) helyezkedik el.
A rutin listája:

```
0 REM  ****  P124.  ****
1 :
2 :
5 SYS36864
6 .OPT OO,P1
7 *= $E500
10 LDX #$00      ;ALSO BYTE
20 LDY #$DC      ;FELSO BYTE
30 RTS

READY.
```

Az utolsó két rutin első pillantásra értelmetlennek tűnik. Ha azonban arra gondolunk, hogy a KERNAL-t a különböző gépek közti programozási különbségek áthidalására hozták létre, akkor már érthetőbb a dolog.
A programozáskor mindig gondot okoz a képernyőformátum megtervezése. E két utolsó rutin segítségével ez a feladat könnyen áttekinthetővé válik.
Reméljük, hogy ez a kis kirándulás a KERNAL területére hasznosnak bizonyult, és segítséget nyújtott a gépi kódú programozás rejtelmeinek megismeréséhez.

11. FEJEZET
A TÁR

11.1 Hogyan tárol a számítógép egy BASIC sort?

Ebben a részben a BASIC sorok tárolásának néhány részletével foglalkozunk. Ahhoz, hogy a számítógép a BASIC sorokat fel tudja dolgozni, szükséges feltétel, hogy a BASIC terület első byte-ja 0 legyen.
Normális esetben a BASIC terület a 2049-es címen kezdődik, így a 2048-as címen 0-nak kell lennie.
Állítsuk át a cím tartalmát egy POKE utasítással 1-re. Ekkor a számítógép a NEW és RUN parancsok hatására SYNTAX ERROR hibajelzést ír ki.
Írtunk egy rövid programot, amely szemléletessé és érthetőbbé teszi a BASIC sorok tárolását. A képernyő felső sorában folyamatosan megjelenik, hogy az utolsó BASIC sor hány byte-ot foglalt le a tárolóban.
A program bevitele és elindítása után a felső sorban a következő felirat jelenik meg:

```
00 HASZNÁLT BYTE
```

Ez azt jelenti, hogy az utolsó bevitel (az utolsó RETURN) óta a BASIC területet nem használtuk. A BASIC területet csak a programsorok foglalják le.
Írjuk be a következő sort:

```
100 PRINT
```

A program így válaszol:

```
06 HASZNALT BYTE
```

Ebből a hat byte-ból öt adminisztrációs feladatokat lát el. A hatodik byte a PRINT utasítás tokenje.
Az öt byte funkciója a következő:

a) az első két byte a következő BASIC sor kezdőcímét tartalmazza alsó és felső byte formájában. Az utolsó sorban az értékük 0;
b) a második két byte a sorszámot tartalmazza szintén alsó és felső byte formájában. Pl. az 1000-es sorszám 232/3 alakban tárolódik;
c) egy byte jelöli a BASIC sor végét. A byte értéke 0, és a sor végén áll.

Írjuk be a következő sort

```
0 REM  ****  P125.  ****
1 :
2 :
10 PRINT"ISTVHN"

READY.
```

Itt 14 byte-ra van szükség. Ebből 5 byte funkcióját már ismerjük. Ezen felül 1 byte a PRINT tokenje, a fennmaradó 8 byte pedig a szöveget tartalmazza (ebből hat az ISTVAN szót, kettő pedig az idézőjeleket).
Próbáljuk ki a következő sort is:

```
0 REM  ****  P126.  ****
1 :
2 :
30 PRINTCHR$(48)

READY.
```

Itt 11 byte szükséges a tároláshoz:

a) 5 byte funkciója ismert;
b) 1 byte a PRINT tokenje;
c) 1 byte a CHR$ tokenje;
d) 4 byte a szöveg (48).

A TAB és az SPC utasításoknál ügyeljünk arra, hogy ezek a kezdő zárójelet is tartalmazzák.
A BASIC sorok byte-jainak felépítése minden BASIC utasításnál megegyezik. Nézzünk erre még egy példát:

```
40 POKE 198,0
```

A felhasznált byte-ok listája pedig:
a) 5 byte funkciója ismert;
b) 1 byte a POKE tokenje;
c) 5 byte a szöveg (198,0).

Végül ismertetjük a BASIC betöltőprogramot:

```
0 REM  ****  P127.  ****
1 :
2 :
10 FORI=0TO87
20 READA
30 POKE40704+I,A
40 S=S+A
50 NEXTI
60 IFS<>7564THENPRINT"HIBA A DATA SORBAN..!!":END
70 PRINT"O.K...!!"
80 SYS40704
100 DATA120,169,15,141,20,3,169,159,141,21,3,133,56,88,96,169,48
```

```
110 DATA141,0,4,141,1,4,165,11,201,76,240,18,56,201,10,144,8,238
120 DATA0,4,233,10,76,30,159,105,48,141,1,4,162,2,189,69,159,157,0,4
130 DATA232,224,18,208,245,169,0,202,157,0,216,208,250,76,49,234
140 DATA32,8,1,19,26,14,1,12,20,32,2,25,20,5,32,32,32

READY.
```

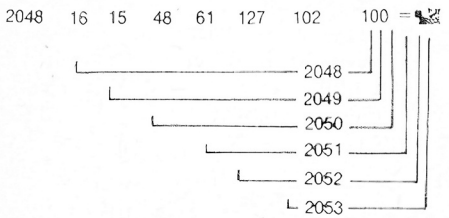
A gépi kódú lista:

```
0 REM  ****  F128.  ****
1 :
2 :
3 SYS49152
4 .OPT OO,P1
5 *= $9F00
10 SEI
20 LDA #0F
30 STA $0314
40 LDA #9F
50 STA $0315
60 STA $38
70 CLI
80 RTS
90 LDA #$30
100 STA $0400
110 STA $0401
120 LDA $0B
130 CMP #$4C
140 BEQ $9F2F
150 SEC
160 CMP #$0A
170 BCC $9F2A
180 INC $0400
190 SBC #$10
200 JMP $9F1E
210 ADC #$30
220 STA $0401
230 LDX #$02
240 LDA $9F45,X
250 STA $0400,X
260 INX
270 CPX #$12
280 BNE $9F31
290 LDA #$00
300 DEX
310 STA $D800,X
320 BNE $9F3E
330 JMP $EA31

READY.
```

Az utolsó adatok a HASZNALT BYTE felirat képernyőkódjai.

11.2 BASIC monitor

Az előzőekben megismertük a BASIC sorok tárolásának módját. Ebben a részben egy olyan rövid programot ismertetünk, amely a képernyőre írja a számítógép tárolótartalmát. Ezzel a módszerrel az egész (BASIC) programot átnézhetjük. Sőt, átvizsgálhatjuk a ROM tartalmát is, ami nagyon érdekes és tanulságos lehet.
A gépi kódú monitorok működését ismerők előtt nem ismeretlen az M utasítás (MEMORY DISPLAY), ami a memóriatartalom képernyőre írását jelenti. Programunk ezt az utasítást valósítja meg.
Először megkérdezi a kezdő- és a végcímet. A képernyőn megjelenik a kezdőcím és még öt cím tartalma, amiket PEEK utasítással olvastunk ki. A sor végén pedig megjelenik a hat érték képernyőkódja.
Példaképpen lássuk a következő (fiktív) sort:



A képernyőn a sorok elején mindig megjelenik az adott sor első byte-jának címe, esetünkben 2048. A címek és a karakterek egymáshoz tartozását az ábra jól szemlélteti. A számokat a program decimális formában jeleníti meg, mivel ezt a monitor BASIC nyelven fejlesztették ki. A programnak egy funkciója van, ezért egyszerűen kezelhető. A program betöltése és indítása után a számítógép a

```
BASIC MONITOR
KEZDOCIM ?
```

felirattal jelentkezik. Adjuk meg a kezdőcím értékét decimálisan. A következő kérdés:

```
VEGCIM?
```

Itt is adjuk meg a kívánt címet. A számítógép ezután a már leírt módon kilistázza a memóriatartalmakat. A próbálkozások során bizonyára fel fog tűnni, hogy az utolsó sort akkor is végigírja a program, ha a végcím a sor közepére esik.
Ha a programot egy hosszabb listázás közben meg akarjuk állítani, nyomjuk le az F1 billentyűt.

Néhány érdekes kezdőcím:

2048	A BASIC terület kezdete. Itt található a monitor (alapállapotban)
40964	BASIC utasítások

A program listája:

```
0 rem **** P129. ****
1 :
2 :
10 print"■■"chr$(14):poke53281,6:poke53280,6
20 print" ■          Basic-Monitor                ■":print:print
30 input" Kezdocim          ";a
40 input" Vegcim            ";e
50 fori=atoesteP5
60 printi;
70 foro=0to5
80 xP=peek(i+o):gosub200: rem print using
90 u=peek(214)*40+1024+33+o
100 poke u,peek(i+o):poke54272+u,1
110 next o
120 print
125 getb$:ifb$=chr$(133)thenstoP
130 nexti
140 print" ■       Meg egyszer (i/n)
150 a$="":geta$:ifa$=""" then 150
160 ifa$="i"then run
170 end
190 :
200 rem Print-forma
210 xP$=str$(xP)
220 iflen(xP$)<4thenxP$=" "+xP$:goto220
240 printxP$;:return

ready.
```

Bár a monitort BASIC nyelven írtuk, néhány cím tartalmának megváltoztatásával vizsgálhatunk vele más BASIC programot is.
Tegyük a következőket:

– hozzuk a számítógépet a bekapcsolási állapotba;
– írjuk be a

POKE 44,150

utasítást. A 150 helyett más számot is választhatunk 8 és 159 között. Az utasítással áthelyeztük a BASIC terület kezdőcímét, és így a két program nem fog ütközni egymással;
– írjuk be a

POKE 256*PEEK(44),0

utasítást, amely a BASIC terület kezdetére egy nullát helyez el. Ez a lépés feltétlenül szükséges;

– töltsük be a BASIC-monitort;
– írjuk be a

POKE 44,8

utasítást. Ez a BASIC-terület kezdetét eredeti helyére állítja vissza;
– töltsük be az átvizsgálandó programot;
– írjuk be ismét a

POKE 44,(általunk választott érték)

utasítást, hogy ismét a monitorhoz jussunk;
– indítsuk el RUN-nal a monitort. Most már könnyen átvizsgálhatjuk a programunkat (a 2048-as címtől).

Mivel a monitor nem hosszú és egyszerűen megérthető, más utasításokkal is kiegészíthetjük. A monitorral pl. megoldhatjuk a BASIC programok átírását.

11.3 A nulláslap listája magyarázatokkal

A következőkben a nulláslap teljes listáját közöljük kiegészítő magyarázatokkal. A lista segítségünkre lehet az ismeretlen címek értelmezésében, de általában a gép működésének jobb megértésében is.

Jó szórakozást!

0000	0	a processzor port adatirány-regisztere
0001	1 055	processzor port 0. bit: ROM vagy RAM a $A000–$BFFF (BASIC értelmező) * POKE 1,54 1. bit: ROM vagy RAM a $E000–$FFFF (operációs rendszer, KERNAL) * POKE 1,53 2. bit: I/0 vagy ROM a $D000–$DFFF (karaktergenerátor) * POKE 1,51
0002	2	használaton kívül
0003	3	alsó byte (lebegőpontos/fix átalakítás)
0004	4	felső byte
0005	5	alsó byte (fix/lebegőpontos átalakítás)
0006	6	felső byte
0007	7	a kereső karakter
0008	8	idézőjel kapcsoló
0009	9	az oszlop a TAB utasításnál
000A	10	értelmező kapcsoló (LOAD = 0, VERIFY = 1)
000B	11	mutató az input pufferben, a dimenziók száma
000C	12	a DIM utasítás kapcsolója
000D	13	típus ($00 = numerikus, $FF = szöveges)
000E	14	kapcsoló, $80 = egész, $00 = valós
000F	15	idézőjel kapcsoló LIST utasításnál
0010	16	FN kapcsoló
0011	17	INPUT ($00), GET($40), READ($98) kapcsoló
0012	18	az ATN előjele (összehasonlításnál az egyenlőség kapcsolója)
0013	19	aktív I/0 egység POKE 19,1: INPUT"KERDOJEL NELKUL";A$: POKE 19,0 (INPUT esetén a kérdőjel kikapcsolása)
0014	20	alsó byte, egész szám
0015	21	felső byte
0016	22	mutató a füzérveremben POKE 22,35: LIST (listázás sorszámok nélkül)
0017	23	alsó byte, mutató az utolsó füzérre
0018	24	felső byte
0019	25	füzérverem
001A	26	füzérverem
001B	27	füzérverem
001C	28	füzérverem
001D	29	füzérverem

001E	30	füzérverem
001F	31	füzérverem
0020	32	füzérverem
0021	33	füzérverem
0022	34	segédmutatók
0023	35	segédmutatók
0024	36	segédmutatók
0025	37	segédmutatók
0026	38	a szorzás eredménye
0027	39	a szorzás eredménye
0028	40	a szorzás eredménye
0029	41	a szorzás eredménye
002A	42	a szorszás eredménye
002B	43 01	alsó byte, BASIC-kezdet új BASIC-kezdet: POKE (új cím)–1,0: NEW!!
002C	44 08	felső byte
002D	45	alsó byte, változóterület kezdete
002E	46	felső byte
002F	47	alsó byte, tömbkezdet
0030	48	felső byte
0031	49	alsó byte, tömbök vége
0032	50	felső byte
0033	51	alsó byte, füzérek kezdete
0034	52	felső byte
0035	53	alsó byte, füzérek segédmutatója
0036	54	felső byte
0037	55	alsó byte, mutató a BASIC RAM végére, segítségével lecsökkenthetjük a BASIC területet az ott tárolt assembler program védelmére
0038	56	felső byte
0039	57	alsó byte, aktuális BASIC sorszám PRINT PEEK(57) + 256*PEEK(58) előállítja az aktuális BASIC sorszámot. (A lekérdezés csak program-üzemmódban érvényes.)
003A	58	felső byte
003B	59	alsó byte, előző BASIC sorszám PRINT PEEK(59) + 256*PEEK(60) előállítja az előző BASIC sorszámot (parancs-üzemmódban is működik)
003C	60	felső byte
003D	61	alsó byte, a következő utasítás mutatója
– – – –	– –	az ugrási cím mutatója CONT esetén
003E	62	felső byte
003F	63	alsó byte, a DATA utasítás aktuális sorszáma, megadja a DATA pillanatnyi sorszámát (READ-del összekötve, saját hibajelzéseknél a sorszám kiírására használhatjuk)

0040	64	felső byte
0041	65	alsó byte, mutató a következő DATA elemre (a BASIC tároló következő DATA elemének címére mutat)
0042	66	felső byte
0043	67	alsó byte, a bevitel forrásának mutatója
0044	68	felső byte
0045	69	1-es regiszter, a változók neve
0046	70	2-es regiszter, a változók neve a tároláshoz
0047	71	alsó byte, a változók címe
0048	72	felső byte
0049	73	alsó byte, a változók értékének mutatója
004A	74	felső byte
004B	75	alsó byte, a programmutató átmeneti tára
004C	76	felső byte
004D	77	az összehasonlító műveletek maszkja
004E	78	alsó byte, mutató az FN-re
004F	79	felső byte
0050	80	füzér-leíró (a különböző felhasznált munkaterületek)
0051	81	füzér-leíró (a különböző felhasznált munkaterületek)
0052	82	füzér-leíró (a különböző felhasznált munkaterületek)
0053	83	füzér-leíró (a különböző felhasznált munkaterületek)
0054	84 76	konstans $4C, ugrás a függvényekre
0055	85	alsó byte, a függvények ugrási vektora
0056	86	felső byte
0057	87	aritmetikai regiszter, 3-as akkumulátor
0058	88	aritmetikai regiszter, 3-as akkumulátor
0059	89	aritmetikai regiszter, 3-as akkumulátor
005A	90	aritmetikai regiszter, 3-as akkumulátor
005B	91	aritmetikai regiszter, 3-as akkumulátor
005C	92	aritmetikai regiszter, 2-es akkumulátor
005D	93	aritmetikai regiszter, 2-es akkumulátor
005E	94	aritmetikai regiszter, 2-es akkumulátor
005F	95	aritmetikai regiszter, 2-es akkumulátor
0060	96	aritmetikai regiszter, 2-es akkumulátor
0061	97	1-es lebegőpontos akkumulátor, FAC
0062	98	1-es lebegőpontos akkumulátor, FAC
0063	99	1-es lebegőpontos akkumulátor, FAC
0064	100	1-es lebegőpontos akkumulátor, FAC
0065	101	1-es lebegőpontos akkumulátor, FAC
0066	102	a FAC előjele
0067	103	számláló a polinom kiértékeléséhez (pl. $E059-től)
0068	104	a FAC kerekítő byte-ja
0069	105	2-es lebegőpontos akkumulátor, ARG
006A	106	2-es lebegőpontos akkumulátor, ARG

006B	107	2-es lebegőpontos akkumulátor, ARG
006C	108	2-es lebegőpontos akkumulátor, ARG
006D	109	2-es lebegőpontos akkumulátor, ARG
006E	110	az ARG előjele
006F	111	FAC/ARG összehasonlító byte
0070	112	a FAC kerekítő byte-ja

Néhány megjegyzés

„A "képletkiértékelő rutin" a $AD9E címtől található a BASIC értelmezőben. A rutin behoz egy tetszőleges kifejezést és kiszámítja az értékét. A kiértékelést valós számokkal végzi, az egész számokat is lebegőpontos számmá alakítja, nemcsak számokat, hanem füzérparamétereket is ki tud értékelni. A füzérparaméterek a számoktól egy megfelelő kapcsoló beállításával különböztethetők meg (13-as cím). A numerikus értékek közbenső tárolója a FAC 1 ($61-től). Mivel azonban ez a tároló nem az aritmetikai műveletek (kivonás, összeadás stb.) célját szolgálja, van egy másik lebegőpontos akkumulátor is (a $69-től), az ARG.
A rutin az eredményt a FAC argumentumába teszi.
A füzérek kiértékelésére a $64–$65 címeket használja, amelyek a füzér-leíró címét tartalmazzák. Itt találhatók a feldolgozandó füzér hosszára és címére vonatkozó információkat. Az értelmező alábbi rutinjait a gépi kódú programozók tetszőleges kifejezések kiértékelésére használhatják:

FAC = ARG + FAC (összeadás)	$B86A (47210)
FAC = ARG − FAC (kivonás)	$B853 (47187)
FAC = ARG * FAC (szorzás)	$BA28 (47656)
FAC = ARG/FAC (osztás)	$BB12 (47890)
FAC = ARG↑FAC (hatványozás)	$BF7B (49019)

A $B475 (46197) rutin a füzér hosszát átviszi az akkumulátorba. A cím alsó és felső byte formájában az X és Y regiszterekben található.

0071	113	alsó byte, mutató a polinom kiértékelésére
0072	114	felső byte, a polinom együtthatóinak ($E043) és fokának ($E059) mutatója
0073	115	CHRGET rutin (assembler ROM lista):
0074	116	
0075	117	0073 INC $7A
0076	118	0075 BNE $0079 BASIC szöveg növelésének mutatója
0077	119	0077 INC $7B
0078	120	0079 LDA $HHLL
0079	121	
007A	122	programszámláló alsó byte
007B	123	programszámláló felső byte
007C	124	

007D	125	007C CMP "$3A ":"
007E	126	007E BCS $008A
007F	127	0080 CMP "$20" " a szóköz átlépése
0080	128	0082 BEQ $0073
0081	129	0084 SEC
0082	130	0085 SBC "$30
0083	131	0087 SEC
0084	132	0088 SBC "$DO
0085	133	008A RTS
0086	134	
0087	135	a CHRGET rutin a ROM-ban hajtódik végre
0088	136	SE3A2-től, de a RESET alatt a RAM-ba
0089	137	másolódik. A rutin csak a RAM-ban működik.
008A	138	Az X és Y regisztert nem használja. CARRY = 0 : a beolvasandó byte tartalma a $30–$39 tartományba van ASCII-kódban. Zero = 1 : karakter = $00 vagy $3A (":")
008B	139	az utolsó RND érték
008C	140	az utolsó RND érték
008D	141	az utolsó RND érték
008E	142	az utolsó RND érték
008F	143	az utolsó RND érték
0090	144	az ST státuszszó (pl. az IEC-busz rutinoknál)
0091	145 10	a STOP billentyű kapcsolója
0092	146	időállandó a szalaghoz ($F92C-től állítja be)
0093	147	LOAD ($00) vagy VERIFY ($01) kapcsoló. A LOAD rutin állítja be, a BOA (10) címet a LOAD parancsban adjuk meg
0094	148	soros (IEC) output kapcsoló
0095	149	az IEC-busz output puffere
0096	150	EOT (End Of Tape) vétele szalagról, kapcsoló
0097	151	a regiszterek közbenső tára
0098	152	a megnyitott file-ok száma POKE 152,0, lezárja az összes file-t POKE 152,12, megakadályozza a file-megnyitást
0099	153 01	aktív input egység (1 = billentyűzet) POKE 153,2, letiltja a billentyűzetet
009A	154 03	aktív output egység (3 = képernyő) POKE 154,1 jelentése ua. mint a 19-es cím az INPUT-nál
009B	155 00	a szalag paritásbyte-ja
009C	156	"a byte vétele megtörtént" kapcsoló
009D	157	direkt-üzemmód ellenőrzés ($00 = program, $80 = RUN) GOTO-val indított program $80!
009E	158	szalag 1. menet, ellenőrzőösszeg
009F	159	szalag 2. menet, hibajavítás
00A0	160	idő 3 (idő 2 átvétele)

00A1	161	idő 2 (idő 1 átvétele)
00A2	162	idő 1 (TI és TI$ számlálója)
00A3	163	a soros olvasás bitszámlálója, $FB97-en számolja a byte 8 bitjét
00A4	164	ciklusszámláló ($A3 = 8, akkor $A4 = 0)
00A5	165	számláló szalagra írás közben
00A6	166	mutató a szalagpufferben (ha beviszünk egy karaktert a szalagpufferbe, akkor ez a cím eggyel nő, és a következő szabad helyre mutat. Ha a puffer megtelik (192 karakter), a zero kapcsoló beáll ($F80D).
00A7	167	LOAD/SAVE közbenső tároló kazettás egységnél
00A8	168	LOAD/SAVE közbenső tároló kazettás egységnél
00A9	169	LOAD/SAVE közbenső tároló kazettás egységnél
00AA	170	LOAD/SAVE közbenső tároló kazettás egységnél
00AB	171	LOAD/SAVE közbenső tároló kazettás egységnél
00AC	172	a szalagpuffer és a görgetés (SCROLLING) mutatója
00AD	173	felső byte a szalagpuffer és a görgetés (SCROLLING) mutatója
00AE	174	alsó byte, LOAD/SAVE program vége mutató
00AF	175	felső byte
00B0	176 00	a szalagműveletek állandói
00B1	177	a szalagműveletek állandói
00B2	178	alsó byte (szalagpuffer eltolása mutató)
00B3	179	felső byte
00B4	180	bitszámláló a szalagon
00B5	181	az RS–232-es következő bitje
00B6	182	a kihozandó byte puffere
00B7	183	a programnév hossza
00B8	184	az utoljára használt file-szám
00B9	185	az utoljára használt másodlagos cím
00BA	186	az utoljára használt egységszám
00BB	187	alsó byte, a file-név mutatója
00BC	188	felső byte
00BD	189	a soros I/0 munkaterülete
00BE	190	a még írandó (olvasandó) blokkok száma ($FBCD rutin)
00BF	191	a soros kihozatal puffere (189-es cím)
00C0	192	kazettás egység motorkapcsolója
00C1	193	alsó byte, I/0 kezdőcím
00C2	194	felső byte ($A000-ra mutat: SAVE-rutin)
00C3	195	a képernyő I/0 végcíme
00C4	196	felső byte, I/0 végcíme
00C5	197	a lenyomott billentyű sorszáma (64 = nincs lenyomva) 10 IF PEEK (1974) = 64 THEN 10 egy billentyű lenyomására vár (lásd függelék)
00C6	198	a lenyomott billentyűk száma

00C7	199	REVERS (RVS) mód kapcsoló 1=REVERS 0=normál POKE 199,1: PRINT "DEMO"
00C8	200	a sor vége bevitelkor
00C9	201	a kurzor sora bevitelkor
00CA	202	a kurzor oszlopa bevitelkor
00CB	203	a lenyomott billentyű száma (64=nincs lenyomva) Hasonló a 197-es címhez. A kettő között a végrehajtás idejében van különbség
00CC	204	kurzorkapcsoló, 0=be, 1=ki, a kurzort programban lehet bekapcsolni
00CD	205	a kurzorvillogás számlálója
00CE	206	a kurzor alatti karakter
00CF	207	kurzorkapcsoló, 1=be, 0=ki 10 IF PEEK (207) THEN 10 vár, amíg a kurzor kikapcsol
00D0	208	beviteli kapcsoló (pl. $E65F, $F16A)
00D1	209	alsó byte, aktuális video-RAM mutató
00D2	210	felső byte
00D3	211	a kurzor oszlopa (a kurzor sora a 214-es cícm, a SET rutin hívása SYS 58732)
00D4	212	idézőjel-üzemmód kapcsoló. Pl. az ESCAPE rutinhoz használható. POKE 212,1: PRINT ... szintén kiírja a vezérlőjeleket
00D5	213	a képernyősor hossza (39–79)
00D6	214	a kurzor sora (l. 211)
00D7	215	különböző célokra
00D8	216	a beszúrások száma
00D9	217	a képernyősor kezdetének MSB-je
-től		
00F0	240	a képernyősor kezdetének MSB-je
00F1	241	ál képernyősor
00F2	242	képernyősor címke
00F3	243	alsó byte, aktuális szín-RAM mutató
00F4	244	felső byte
00F5	245	alsó byte, billentyűzet-dekódolási táblázat mutató
00F6	246	felső byte
00F7	247	alsó byte, RS–232-es input puffer mutató
00F8	248	felső byte
00F9	249	alsó byte, RS 232-es output puffer mutató
00FA	250	felső byte
00FB	251	nulláslap terület saját felhasználásra
00FC	252	nulláslap terület saját felhasználásra
OOFD	253	nulláslap terület saját felhasználásra
00FE	254	nulláslap terület saját felhasználásra
00FF	255	lebegőpontos (ASCII) átalakítás puffer

A nulláslap vége

11.4 A következő lapok fontosabb címei

0277–0280 631–640 Billentyűzetpuffer
Ezt a területet közbenső tárolóként használhatjuk. Akkor van pl. jelentősége, amikor a karakterek bevitele közben a számítógép műveletekkel van elfoglalva. A billentyűzetpuffert saját céljainkra is felhasználhatjuk. A benne tárolt adatok megjelennek, mihelyt a 198-as címnek a megfelelő értéket adjuk.
A billentyűzetpuffer sajátossága, hogy a benne tárolt karaktereket csak a program befejeztével, direkt üzemmódban írja ki a számítógép.
A billentyűzetpuffer segítségével új BASIC sorokat fűzhetünk a már futó programunkhoz (l. a *DATA generátor* c. fejezetben a programot).

0286 646 Karakterszín-tároló

Ez a cím tárolja a pillanatnyi karakterszínt. Ha a cím tartalmát megváltoztatjuk akkor a karakterszín módosul.

```
 0 REM **** P130. ****
 1 :
 2 :
 10 A=INT(RND(1)*15)
 20 POKE646,A:REM VELETLEN SZIN
 30 PRINT"*";
 40 GOTO10

 READY.
```

0 = fekete	1 = fehér	2 = vörös
3 = türkiz	4 = lila	5 = zöld
6 = kék	7 = sárga	8 = narancs
9 = barna	10 = rózsaszín	11 = 1. szürke
12 = 2. szürke	13 = világoszöld	14 = világoskék
15 = 3. szürke		

0288 648 A video-RAM felső byte-ja
A

```
PRINT PEEK (648)*256
```

utasítás megadja a képernyőtároló aktuális kezdőcímét. Értéke alapállapotban: 1024.
A képernyőtárolót az 53272-es cím 4.–7. és az 56576-os cím 0. és 1. bitje segítségével tolhatjuk el (l. még a *Grafika* c. fejezetben!)

028A 650 A billentyűk ismétlő funkciója

A cím által nyújtott vezérlési lehetőségek:

POKE 650,0	csak a vezérlőbillentyűk ismétlése
POKE 650,64	az ismétlés kikapcsolása
POKE 650,128	az összes billentyű ismétlése

028C	652	Az ismétlés-késleltetés számlálója

A 650-es cím tartalmának megváltoztatásával elérhetjük, hogy a lenyomva tartott billentyű megismétlődjék a képernyőn. A folyamat a következő: először csak egy, majd kis idő elteltével további karakterek jelennek meg.
A 652-es cím tartalma határozza meg, hogy az első és a többi karakter megjelenése között mennyi idő teljen el.
A számunkra megfelelő értéket próbálgatással érdemes beállítani!

028D; 028E	653:654	A SHIFT, a C= és a CTRL kapcsolója

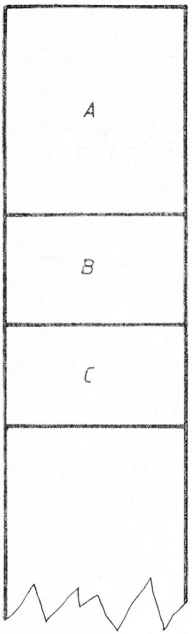
A címek 0., 1. és 2. bitjével állíthatjuk be ezeket a billentyűket

0291	657	A SHIFT/C= letiltva kapcsoló

A cím tartalmának megváltoztatásával letilthatjuk a másik karakterkészletre való áttérést a SHIFT/C= billentyűk hatására.

11.5 A változók tárolása

Az alábbiakban áttekintést adunk arról, mely területeken tárolja a számítógép a változókat:



– A BASIC tároló kezdete ($2B,$2C) $0800

– A-terület: BASIC program

– A változók kezdete ($2D, $2E)

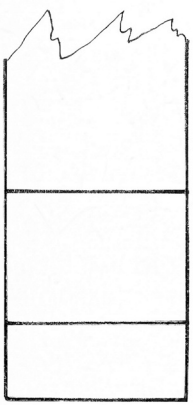
– B-terület: a változók tárolása

– A tömbök kezdete ($2F, $30)

– C-terület: a tömbváltozók tárolása

– A tömbök vége ($31, $32)

– Szabad BASIC tároló



– Szabad BASIC tároló

– A füzérek kezdete (lefelé mozog, $33, $34)

– Tárolt füzérek

– A BASIC terület vége ($37, $38)

– $9FFF

A változók tárolásának leírása

Az A-területen található a BASIC program. Ezt követi a valós, az egész számok és a függvényeljárások (DEF FN) tárolása (B-terület).
A C-terület tárolja a tömböket, vagyis az indexes változókat. A füzérek tárolási helye a BASIC terület vége.

Az összes területet a mutatók állítják. A mutatókat alapállapotban az értelmező automatikusan meghatározza. Ezeket a mutatókat mi magunk is megváltoztathatjuk (különösen a BASIC-kezdet és -vég mutatókat).

43–44 A BASIC kezdete
45–46 A változók kezdete
47–48 A tömbök kezdete
49–50 A tömbök vége
51–52 A füzérek kezdete
53–54 Segédmutató
55–56 A BASIC vége

A változók kezdetének megváltoztatásakor gondolni kell arra, hogy a program a tároláskor hosszabb lesz. A SAVE rutin ui. vakon kiment mindent, ami a 43–44 és a 45–46 címek között elhelyezkedik!

A tároláshoz

Mivel a tömbök külön tárolóterületet foglalnak le maguknak, összesen négy különböző változóformát különböztetünk meg:

a) valós számok (pl. A, B, CD stb.);
b) füzérek ($ jellel kezdődő változók);
c) egész számok (% jellel kezdődő változók);
d) függvények (FN).

A változótípusok megkülönböztetéséhez 2 bitre (4 lehetséges variáció) van szükségünk.
A két bit a változónévben található.
Mivel a változónevekben nem szerepelhet inverz karakter, ezért a 7. bit máris rendelkezésünkre áll. A változónevek mindig két karakteresek (akkor is, ha egyet adunk meg), tehát itt is van egy szabad bit.
A B-területen az értelmező minden használt változónak lefoglal 7 byte-ot. Ebből kettő tartalmazza a változónevet. Mindegy tehát, hogy A vagy DR változónevet használunk. A nevekben található 2 bittel a következő módon különböztetjük meg a változótípusokat.

Változónév 1. byte	Változónév 2. byte	Változótípus
0	0	valós
0	1	füzér
1	0	függvény
1	1	egész

A 7 byte felépítése:

BYTE	1	2	3	4	5	6	7	Változó
	VN1	VN2	binárisan tárolódik					valós
	VN1	VN2	hossz	LB	HB	0	0	füzér
	VN1	VN2	HB	LB	0	0	0	egész

VN1 Első változónév
VN2 Második változónév
LB és HB A változótartalom alsó és felső byte-ja

A függvényváltozó kivételt képez, mivel a tárolásához 14 byte-ra van szükség.

BYTE	1	2	3	4	5	6	7
	VN1F	VN2F	LB	HB	LBA	HBA	ASCD
BYTE	8	9	10	11	12	13	14
	VN1V	VN2V	az argumentum értéke				

VN1F–VN2F A függvény változónevének első és második karaktere
LB–HB A függvény alsó és felső byte-ja
LBA–HBA Az argumentum alsó és felső byte-ja
VN1V–VN2V Az argumentum változónevének első és második karaktere

11.6 Az érdekesebb mutatók listája

A következőkben ismertetjük a C 64-es fontosabb mutatóinak listáját. Ezen mutatók megváltoztatásával saját rutinra is ugorhatunk anélkül, hogy a ROM-ot a Ram-ba másolnánk.
Végül a lista:

Alsó byte	Felső byte	Ugrási cím	Rutin
3	4		lebegőpontos/fix átalakítás
5	6		fix/lebegőpontos átalakítás
23	24		az utoljára használt füzér
34	35		a mutató szabad felhasználása
36	37		a mutató szabad felhasználása
43	44		BASIC start (1. byte = 0)
45	46		változókezdet
47	48		tömbkezdet
49	50		a tömbök vége
51	52		a füzérek kezdete (lefelé mozog)
53	54		a füzérek segédmutatója
55	56		a BASIC vége
61	62		a CONT utáni utasítás mutatója
65	66		a következő DATA elem
85	86		a függvények ugrási vektora
122	123		programmutató (CHRGET rutin)
178	179		a kazettapuffer kezdőcíme
187	188		a programnév mutatója
209	210		az aktuális BASIC sor mutatója
243	244		az aktuális szín-RAM mutatója
245	246		billentyűzet-dekódolási táblázat
247	248		RS–232-es input puffer
248	250		RS–232-es output puffer
655	656		billentyűzet-dekódolás, mutató
768	769	$E38B	BASIC melegstart
770	771	$A483	sorbevitel
772	773	$A57C	értelmező kóddá alakítás
774	775	$A71A	szöveggé alakítás (LIST)
776	777	$A7E4	a BASIC utasításcím beolvasása
778	779	$AE86	a kifejezés kiértékelése
785	786	$B248	USR mutató
788	789	$EA31	IRQ vektor
790	791	$FE66	BREAK vektor
792	793	$FE47	NMI vektor
794	795	$F34A	OPEN

796	797	$F291	CLOSE
798	799	$F20E	CHKIN
800	801	$F250	CKOUT
802	803	$F333	CLRCH
804	805	$F157	INPUT
806	807	$F1CA	OUTPUT
808	809	$F6ED	STOP
810	811	$F13E	GET
812	813	$F32F	CLALL
814	815	$FE66	melegstart
816	817	$F45A	LOAD
818	819	$F5ED	SAVE

A mutatók lekérdezése:

PRINT PEEK(alsó + 256*PEEK(felső)

Néhány mutatót csak a megszakítás letiltása után változtathatunk meg:

POKE 56334,0 megszakítás ki
POKE 56334,1 megszakítás be

12. FEJEZET
FÜGGELÉK

12.1 A táblázatokról általában

Ezen függelékünk a C 64-es Felhasználói kézikönyv táblázatait kívánja kiegészíteni. Az a véleményünk, hogy abból a könyvből néhány táblázatot kifelejtettek. Mi a sok kis táblázatot egy naggyá foglaltuk össze, amelyben az egyes számok mögött megtaláljuk azok jelentéseit is. Csak a joystick lekérdezést nem tartalmazza a táblázatunk.
Az első táblázatban megtaláljuk az egyes számok három számrendszerbeli alakját, sorrendben a decimális, a hexadecimális és a bináris alakokat. Ezenkívül a táblázat tartalmazza a gépi kódú utasításokat (mnemonikokat), a BASIC utasításokat (tokeneket), valamint a billentyűzet-dekódoló segédletet. A táblázat első három oszlopához nincs hozzáfűznivalónk. A BASIC-ben csak decimális számokat használunk. A hexadecimális számokra a gépi kodú programozásban van szükség. A bináris számok ismerete a számítógép működésének jobb megértéséhez szükséges.
A MNEMONIK oszlopban a gépi kódú utasítások vannak. A Z mindenütt hexadecimális számot jelent. Mint már említettük, a BASIC utasítások a memóriában egy kódként tárolódnak, amelynek TOKEN a neve. A táblázat ötödik oszlopa ezeket tartalmazza.
Az utolsó oszlopban a billentyűzet-dekódoló segédletet láthatjuk. Ezt az értéket pl. a 197-es vagy a 203-as címen találhatjuk.

12.2 Átszámítási táblázat

Dec.	Hex.	Bináris	Mnemonik	Token	Billentyűzet-dekódoló
0	$00	%00000000	BRK		Inst/Del
1	$01	%00000001	ORA (ZZ,X)		RETURN
2	$02	%00000010			Kurzor jobbra
3	$03	%00000011			F7
4	$04	%00000100			F1
5	$05	%00000101	ORA ZZ		F3
6	$06	%00000110	ASL ZZ		F5
7	$07	%00000111			Kurzor le
8	$08	%00001000	PHP		3
9	$09	%00001001	ORA #ZZ		W
10	$0A	%00001010	ASL		A
11	$0B	%00001011			4
12	$0C	%00001100			Z
13	$0D	%00001101	ORA ZZZZ		S
14	$0E	%00001110	ASL ZZZZ		E
15	$0F	%00001111			SHIFT balra
16	$10	%00010000	BPL ZZZZ		5
17	$11	%00010001	ORA (ZZ), Y		R
18	$12	%00010010			D
19	$13	%00010011			6
20	$14	%00010100			C
21	$15	%00010101	ORA ZZ, X		F
22	$16	%00010110	ASL ZZ,X		T
23	$17	%00010111			X
24	$18	%00011000	CLC		7
25	$19	%00011001	ORA ZZZZ,Y		Y
26	$1A	%00011010			G
27	$1B	%00011011			8
28	$1C	%00011100			B
29	S1D	%00011101	ORA ZZZZ,X		H
30	$1E	%00011110	ASL ZZZZ,X		U
31	$1F	%00011111			V
32	$20	%00100000	JSR ZZZZ		9
33	$21	%00100001	AND (ZZ,X)		I
34	$22	%00100010			J

35	$23	%00100011		0
36	$24	%00100100	BIT ZZ	M
37	$25	%00100101	AND ZZ	K
38	$26	%00100110	ROL ZZ	0
39	$27	%00100111		N
40	$28	%00101000	PLP	+
41	$29	%00101001	AND #ZZ	P
42	$2A	%00101010	ROL	L
43	$2B	%00101011		–
44	$2C	%00101100	BIT ZZZZ	.
45	$2D	%00101101	AND ZZZZ	:
46	$2E	%00101110	ROL ZZZZ	Ü
47	$2F	%00101111		,
48	$30	%00110000	BMI ZZZZ	£
49	$31	%00110001	AND (ZZ), Y	*
50	$32	%00110010		;
51	$33	%00110011		CLR HOME
52	$34	%00110100		SHIFT jobbra
53	$35	%00110101	AND ZZ, X	=
54	$36	%00110110	ROL ZZ, X	↑
55	$37	%00110111		–
56	$38	%00111000	SEC	1
57	$39	%00111001	AND ZZZZ,Y	nyil
58	$3A	%00111010		CTRL
59	$3B	%00111011		2
60	$3C	%00111100		SPACE
61	$3D	%00111101	AND ZZZZ,X	COMMODORE
62	$3E	%00111110	ROL ZZZZ,X	Q
63	$3F	%00111111		RUN-STOP
64	$40	%01000000	RTI	nincs bill.
65	S41	%01000001	EOR (ZZ,X)	
66	$42	%01000010		
67	$43	%01000011		
68	$44	%01000100		
69	$45	%01000101		
70	$46	%01000110	LSR ZZ	
71	$47	%01000111		
72	$48	%01001000	PHA	
73	$49	%01001001	EOR #ZZ	
74	$4A	%01001010	LSR	

75	$4B	%01001011	
76	$4C	%01001100	JMP ZZZZ
77	$4D	%01001101	EOR ZZZZ
78	$4E	%01001110	LSR ZZZZ
79	S4F	%01001111	
80	$50	%01010000	BVC ZZZZ
81	$51	%01010001	EOR (ZZ), Y
82	$52	%01010010	
83	$53	%01010011	
84	$54	%01010100	
85	$55	%01010101	EOR ZZ,X
86	$56	%01010110	LSR ZZ,X
87	$57	%01010111	
88	$58	%01011000	CLI
89	$59	%01011001	EOR ZZZZ,Y
90	$5A	%01011010	
91	$5B	%01011011	
92	$5C	%01011100	
93	$5D	%01011101	EOR ZZZZ,X
94	$5E	%01011110	LSR ZZZZ,X
95	$5F	%01011111	
96	$60	%01100000	RTS
97	$61	%01100001	ADC (ZZ,X)
98	$62	%01100010	
99	$63	%01100011	
100	$64	%01100100	
101	$65	%01100101	ADC ZZ
102	$66	%01100110	ROR ZZ
103	$67	%01100111	
104	$68	%01101000	PLA
105	$69	%01101001	ADC #ZZ
106	$6A	%01101010	ROR
107	$6B	%01101011	
108	$6C	%01101100	JMP (ZZZZ)
109	$6D	%01101101	ADC ZZZZ
110	$6E	%01101110	ROR ZZZZ
111	$6F	%01101111	
112	$70	%01110000	BVS ZZZZ
113	$71	%01110001	ADC (ZZ),Y
114	$72	%01110010	

115	$73	%01110011		
116	$74	%01110100		
117	$75	%01110101	ADC ZZ,X	
118	$76	%01110110	ROR ZZ,X	
118	$77	%01110111		
120	$78	%01111000	SEI	
121	$79	%01111001	ADC ZZZZ,Y	
122	$7A	%01111010		
123	$7B	%01111011		
124	$7C	%01111100		
125	$7D	%01111101	ADC ZZZZ,X	
126	$7E	%01111110	ROR ZZZZ,X	
127	$7F	%01111111		
128	$80	%10000000		END
129	$81	%10000001	STA (ZZ,X)	FOR
130	$82	%10000010		NEXT
131	$83	%10000011		DATA
132	$84	%10000100	STY ZZ	INPUT
133	$85	%10000101	STA ZZ	INPUT#
134	$86	%10000110	STX	DIM
135	$87	%10000111		READ
136	$88	%10001000	DEY	LET
137	$89	%10001001		GOTO
138	$8A	%10001010	TAX	RUN
139	$8B	%10001011		IF
140	$8C	%10001100	STY ZZZZ	RESTORE
141	$8D	%10001101	STA ZZZZ	GOSUB
142	$8E	%10001110	STX ZZZZ	RETURN
143	$8F	%10001111		REM
144	$90	%10010000	BCC ZZZZ	STOP
145	$91	%10010001	STA (ZZ),Y	ON
146	$92	%10010010		WAIT
147	$93	%10010011		LOAD
148	$94	%10010100	STY ZZ,X	SAVE
149	$95	%10010101	STAY ZZ,X	VERIFY
150	$96	%10010110	STX ZZ,Y	DEF
151	$97	%10010111		POKE
152	$98	%10011000	TYA	PRINT#
153	$99	%10011001	STA ZZZZ,Y	PRINT
154	$9A	%10011010	TXS	CONT

155	$9B	%10011011		LIST
156	$9C	%10011100		CLR
157	$9D	%10011101	STA ZZZZ,X	CMD
158	$9E	%10011110		SYS
159	$9F	%10011111		OPEN
160	$AO	%10100000	LDY #ZZ	CLOSE
161	$A1	%10100001	LDA (ZZ,X)	GET
162	$A2	%10100010	LDX #ZZ	NEW
163	$A3	%10100011		TAB
164	$A4	%10100100	LDY ZZ	TO
165	$A5	%10100101	LDA ZZ	FN
166	$A6	%10100110	LDX ZZ	SPC(
167	$A7	%10100111		THEN
168	$A8	%10101000	TAY	NOT
169	$A9	%10101001	LDA #ZZ	STEP
170	$AA	%10101010	TAX	+
171	$AB	%10101011		–
172	$AC	%10101100	LDY ZZZZ	*
173	$AD	%10101101	LDA ZZZZ	–
174	$AE	%10101110	LDY ZZZZ	
175	$AF	%10101111		AND
176	$BO	%10110000	BCS ZZZZ	OR
177	$B1	%10110001	LDA (ZZ),Y	:
178	$B2	%10110010		=
178	$B3	%10110011		;
180	$B4	%10110100	LDY ZZ,X	SGN
181	$B5	%10110101	LDA ZZ,X	INT
182	$B6	%10110110	LDY ZZ,Y	ABS
183	$B7	%10110111		USR
184	$B8	%10111000	CLV	FRE
185	$B9	%10111001	LDA ZZZZ,Y	POS
186	$BA	%10111010	TSX	SQR
187	$BB	%10111011		RND
188	$BC	%10111100	LDY ZZZZ,X	LOG
189	$BD	%10111101	LDA ZZZZ,X	EXP
190	$BE	%10111110	LDY ZZZZ,Y	COS
191	$BF	%10111111		SIN
192	$CO	%11000000	CPY #ZZ	TAN
193	$C1	%11000001	CMP (ZZ,X)	ATN
194	$C2	%11000010		PEEK

195	$C3	%11000011		LEN
196	$C4	%11000100	CPY ZZ	STR$
197	$C5	%11000101	CMP ZZ	VAL
198	$C6	%11000110	DEC ZZ	ASC
199	$C7	%11000111		CHR$
200	$C8	%11001000	INY	LEFT$
201	$C9	%11001001	CMP #ZZ	RIGHT$
202	$CA	%11001010	DEX	MID$
203	$CB	%11001011		GO
204	$CC	%11001100	CPY ZZZZ	
205	$CD	%11001101	CMP ZZZZ	
206	$CE	%11001110	DEC ZZZZ	
207	$CF	%11001111		
208	$DO	%11010000	BNE ZZZZ	
209	$D1	%11010001	CMP (ZZ),Y	
210	$D2	%11010010		
211	$D3	%11010011		
212	$D4	%11010100		
213	$D5	%11010101	CMP ZZ,X	
214	$D6	%11010110	DEC ZZ,X	
215	$D7	%11010111		
216	$D8	%11011000	CLC	
217	$D9	%11011001	CMP ZZZZ,Y	
218	$DA	%11011010		
219	$DB	%11011011		
220	$DC	%11011100		
221	$DD	%11011101	CMP ZZZZ,X	
222	$DE	%11011110	DEC ZZZZ,X	
223	$DF	%11011111		
224	$EO	%11100000	CPX #ZZ	
225	$E1	%11100001	SBC (ZZ,X)	
226	$E2	%11100010		
227	$E3	%11100011		
228	$E4	%11100100	CPX ZZ	
229	$E5	%11100101	SBC ZZ	
230	$E6	%11100110	INC ZZ	
231	$E7	%11100111		
232	$E8	%11101000	INX	
233	$E9	%11101001	SBC #ZZ	
234	$EA	%11101010	NOP	

235	$EB	%11101011		
236	$EC	%11101100	CPX ZZZZ	
237	$ED	%11101101	SBC ZZZZ	
238	$EE	%11101110	DEC ZZZZ	
239	$EF	%11101111		
240	$FO	%11110000	BEQ ZZZZ	
241	$F1	%11110001	SBC (ZZ),Y	
242	$F2	%11110010		
243	$F3	%11110011		
244	$F4	%11110100		
245	$F5	%11110101	SBC ZZ,X	
246	$F6	%11110110	INC ZZ,X	
247	$F7	%11110111		
248	$F8	%11111000	SED	
249	$F9	%11111001	SBC ZZZZ,Y	
250	$FA	%11111010		
251	$FB	%11111011		
252	$FC	%11111100		
253	$FD	%11111101	SBC ZZZZ,X	
254	$FE	%11111110	INC ZZZZ,X	
255	$FF	%11111111		PI

12.3 Az egységszámok táblázata

A következő táblázat megadja, hogy melyik készülékhez milyen egységszámot rendelhetünk:

Egységszám	Készülék
0	billentyűzet
1	kazettás egység
2	RS–232-es illesztő
3	képernyő
4	nyomtató
5	további nyomtatók
6	további nyomtatók
7	további nyomtatók
8	lemezegység
9	további lemezegységek
10	további lemezegységek
11	további lemezegységek
12	további lemezegységek
13	további lemezegységek
14	további lemezegységek
15	további lemezegységek

13. FEJEZET
EGY HARDVER TIPP

A számítógép üzemének félbeszakítása

Mindnyájan átéltünk már hasonló helyzetet: éppen belemerültünk egy jó játékba, amikor csöngött a telefon, vagy kopogtattak az ajtón. Abba kellett hagyni a játékot. Ezzel elveszítettük a korábban szerzett pontjainkat, mivel nagyon kevés számítógépnek van olyan billentyűje, amellyel félbeszakíthatjuk a programot.

Szerencsére van egy hardveres lehetőség a számítógép üzemének félbeszakítására. Kössük össze a bővítőegység (expansionsport) IRQ-vezetékét egy kapcsolón keresztül a földvezetékkel (GND).

Ha a kapcsolót zárjuk, akkor a számítógép leáll, és csak abban az esetben indul újra, ha a kapcsolót lekapcsoljuk.

Ezt a kapcsolót nemcsak a programok megállítására használhatjuk. Gondoljunk pl. a programok listázására.

Ha a program listája a CTRL billentyűvel is túl gyorsan halad ahhoz, hogy elolvashassuk, akkor tegyünk egy nyomógombot az előbb említett helyre. Ha meg akarjuk állítani a listázást, csak nyomjuk meg a nyomógombot.

Néhány megjegyzés: a kontaktus zárásával az IRQ rutin a megszokottnál többször hajtódik végre, tehát a kurzor gyorsabban villog, az óra még pontatlanabb, mint eddig. De a listázásnál és a játékprogramoknál ennek semmi jelentősége sincs.

Zenekönyv a Commodore 64-hez

A könyv a C–64 felhasználási lehetőségeinek újabb területeivel ismerteti meg az olvasót.
Az első részben ismert Beatles-slágerek leírásán keresztül mélyedhet el az érdeklődő a BASIC zeneprogramozás alapjaiban, a programszerkesztés menetében.
A második rész a hangképzés folyamatával ismerteti meg az olvasót. Részletesen magyarázza az ADSR érték, a frekvenciabeállítás, a szűrők használatát.
A harmadik rész a halódó zeneprogramozók csemegéje. Ez tartalmazza az assembler szintű zeneprogramozás leírását. Itt található a Syntimat 64 rövid leírása is.
A függelékben a hardver iránt érdeklődők találnak hasznos tanácsokat a számítógép és a sztereoberendezések kapcsolatáról és egyéb felhasználási lehetőségekről.

Simon's Basic gyakorlatok

Ez a könyv hiánypótló a hazai C 64 irodalomban. Mindazok haszonnal forgathatják, akik a Simon's Basic-kel már birtokosai egy nagyszerű Basic-bővítésnek, de azoknak is ajánljuk, akik még nem rendelkeznek vele.
Az előbbiek számára egy szerkezetileg logikusan felépített, mintaprogramokkal (szám szerint 100!), részletes magyarázatokkal és ellenőrző kérdésekkel megszerkesztett „tankönyv", amely kitér, de nem bonyolódik bele a szükséges alapismeretek részletezésébe.
Az utóbbiak számára ismertetőként, kedvcsinálóként szolgálhat. Szerkezete lehetővé teszi, hogy bárki rövid belelapozással is képet kapjon a Simon's Basic nyújtotta lehetőségekről.
Az eredeti Simon's Basic kézikönyvnél pontosabb, igényesebb, felépítése folytán könnyebben kezelhető és sokkal jobban érthető. Minőségét a szerzők neve szavatolja, akik már eddig is bizonyították szakértelmüket.

Gépi kódú programozás a Commodore 64-esen haladók számára

A könyv elsősorban azok számára készült, akik már rendelkeznek némi gépi kódú programozási gyakorlattal.
A szerző a feladat elemzésétől kezdve minden lépést részletesen kidolgozva vezeti el az Olvasót a magasszintű gépi kódú program elkészítéséig.
A könyv tartalmából: a megszakítástechnika és annak saját célú felhasználása, az operációs rendszer rutinjai; célszerű módosításuk, és beépítésük saját készítésű gépi kódú programokba stb.
Számos izgalmas témakör mindazoknak, akiknek szívügyük a számítógép.

Megnyílt a NOVOTRADE Rt. COMMODORE User's klubja!

SZOLGÁLTATÁSAINK:

- klubtagoknak szabad gépidőt és szakirodalmat biztosítunk.
- közületeknek és magánszemélyek részére C 16-os, C 64-es programozói, valamint speciális tanfolyamokat szervezünk.

RÉSZLETES FELVILÁGOSÍTÁS:

2C

Számítástechnikai szaküzlet
1136 Budapest, Balzac u. 35.
Telefon: 402-954
Gábriel László klubvezetőnél

Ára: 239,– Ft