Wilhelm Besenthal – Jens Muus

PLUS/4

Kézikönyv
az összes tudnivalóval

NOVOTRADE

Wilhelm Besenthal – Jens Muus

PLUS/4

Kézikönyv
az összes tudnivalóval

NOVOTRADE, 1989

A könyv eredeti címe: Alles über den PLUS 4 (1986)
Fordította: INOTAI LÁSZLÓ
Lektorálta: BRÁNYI LÁSZLÓ

A kiadásért felel RÉNYI GÁBOR, a NOVOTRADE RT. vezérigazgatója
Budapest, 1989
Műszaki szerkesztő: DÉVÉNYI ERIKA

Szedte a Nyomdaipari Fényszedő Üzem, Budapest (88.8627/10)
Készült a Somogy Megyei Nyomdaipari Vállalat kaposvári üzemében
Felelős vezető: MIKE FERENC igazgató

ISBN 963 02 5894 3

Hungarian translation © Inotai László, 1989
Copyright © 1986, Markt & Technik Verlag

TARTALOMJEGYZÉK

Előszó

Amikor a Plus/4-es néhány évvel ezelőtt piacra került, nem sok jövőt jósoltak neki. Az akkori árviszonyok mellett ez a pesszimizmus indokolt is volt.

Azóta jelentősen megváltozott a helyzet. A mai árat tekintve a gép képességeit akár szenzációsnak is lehetne nevezni. A tárkapacitása 64 kbyte RAM, kiemelkedő beépített szoftverrel rendelkezik, és számos külső egység csatlakoztatható hozzá. Könnyen kezelhető, ami a jól kialakított billentyűzetnek is köszönhető. A programozó munkáját a jelentősen kibővített BASIC 3.5 programozási nyelv támogatja.

Ez a könyv mindazokhoz szól, akik bővíteni szeretnék a Plus/4-ről eddig szerzett ismereteiket. Szándékosan foglalkoztunk olyan kérdésekkel, amelyeket a géppel szállított kézikönyv válasz nélkül hagy.

A könyv felépítéséről:

Először megismerkedünk néhány számítástechnikai alapfogalommal. Az első fejezetekben rövid áttekintést adunk a Plus/4-es felépítéséről és működéséről. Ezután részletesen ismertetjük a legfontosabb BASIC utasításokat. A könyv e részében tárgyaljuk a kazettás magnó és a lemezmeghajtó egység működését is.

A könyv második részében a gépi kódú nyelvvel foglalkozunk. Ismertetjük a beépített gépi kódú monitort, majd a gépi kódú parancsokat. Ezt követően kitérünk a tárkezelésre, a megszakítások programozására, és bemutatjuk, hogy hogyan lehet egy nyomtatókészüléket a Centronics-illesztőn keresztül a számítógéphez csatlakoztatni. Szólunk a DATA-előállításról, egy riasztóberendezés megépítéséről, és még egyebekről. Mindent nem akarunk itt elárulni, maradjon meglepetés is, hogy még mi mindenre képes ez a számítógép.

A könyv végén részletes függelék található, amelynek segítségével gyorsan eligazodhatunk a legfontosabb dolgokban, és újabb információkhoz is hozzájuthatunk. Egy kislexikonban összegyűjtöttük a számítástechnika legfontosabb alapfogalmait, és megadtuk ezek rövid magyarázatát.

Mindegyik fejezet önmagában zárt egység. Ebből következően bármelyik fejezet külön-külön is tanulmányozható és feldolgozható. A kissé ravaszabb fogások megértése azonban bizonyos előismereteket igényel, amelyeket viszont az első fejezetek tartalmaznak. Ezért célszerű a könyv tanulmányozását az elején kezdeni.

A könyvben levő programokról:

A könyvben levő programok közül jó néhányat igen részletesen tárgyalunk. A magyarázatokkal a programok megértését, és az esetlegesen kívánt változtatások elvég-

zését szándékoztuk megkönnyíteni. A szerzők tudják, hogy egy feladat megoldására nem mindig az ideális megoldást választották, de rövid, és áttekinthető programokat akartak írni. A könyv áttanulmányozása után az Olvasó bizonyára képes lesz arra, hogy a programokat a saját igényei szerint alakítsa. Valamennyi programot alaposan kipróbáltuk, úgyhogy ezeknek hibátlanul kell futniuk. Ha ennek ellenére problémák adódnának, akkor meg kell vizsgálni a gépbe beírt programokat, ill. a megadott programleírás segítségével meg kell keresni az esetleges hibát.

Ezen a helyen mondunk köszönetet mindazoknak, akik a könyv megírásához szükséges technikai eszközöket rendelkezésünkre bocsátották, továbbá azoknak, akik a kézirat legépelésében és javításában segédkeztek.

Sok sikert és örömöt a könyv olvasásához, és a Plus/4-es varázslatos világának feltárásához.

Jens Muus / Wilhelm Besenthal

1. A Commodore Plus/4-es

A Commodore Plus/4-es gép „személyében" mintegy két évvel ezelőtt olyan számítógép jelent meg a piacon, amely a házi számítógépek kategóriájában új dimenziókat nyitott. A gép már alapkiépítésében is nem kevesebb mint 64 kbyte RAM-mal, kimondottan jó BASIC-kel, és nem utolsósorban még négy, beépített programmal rendelkezik. Vele egyidőben jelentek meg a kisebb testvérei, a C 16-os és a C 116-os, amelyek a Plus/4-essel teljes mértékben kompatíbilisek.

A Plus/4-es igazán sokoldalúan használható házi számítógép. Igen jó és áttekinthető a billentyűzete, így a gépen végzendő huzamosabb munka sem fárasztó. A programok készítéséhez – a korábbi, VC 20-as és C 64-es minimális BASIC-jéhez képest – lényegesen bővített BASIC utasításkészlet áll rendelkezésre, amely – nem utolsósorban a hibakeresésre alkalmas speciális parancsoknak köszönhetően – lényegesen megkönnyíti a programozást.

Ugyancsak igen egyszerűen készíthetők rövidebb gépi kódú rutinok is. Ehhez a Plus/4-es operációs rendszerébe beépített, könnyen kezelhető gépi kódú monitorprogram áll rendelkezésünkre.

Annak, aki a Plus/4-est a külvilággal akarja összekapcsolni, a bőség zavarával kell szembenéznie. A gépen számos csatlakozó van, amelyeken keresztül gyakorlatilag minden probléma megoldható. A legfontosabb csatlakozó a Commodore típusú gépekre jellemző, úgynevezett felhasználói kapu (user port). E csatlakozó lábkiosztása és programozása azonban más mint a VC 20-asé vagy a C 64-esé. A gépen található továbbá a külső Commodore készülékek csatlakoztatására a soros busz-csatlakozó, a kazettabemenet, a bővítő csatlakoztatója, továbbá két bemenet a botkormányokhoz (joystick). A felhasználói kapu még egy RS232 bemenetet is tartalmaz.

Amint már említettük, a Plus/4-es kompatíbilis a C 16-os és a C 116-os számítógépekkel. Ezeknek a gépeknek közel azonos a hardverük, teljesen azonos az operációs rendszerük és a BASIC interpreterük. A VC 20-as és a C 64-es gépeken írt BASIC nyelvű programok ezeken a gépeken azonban csak akkor futtathatók, ha a programok nem tartalmaznak gépfüggő POKE, PEEK, SYS, USR vagy WAIT utasításokat.

Néhány szó a legfontosabb külső egységekről:

A legtöbb külső Commodore-készülék változtatás nélkül csatlakoztatható, így pl. a 1525, 1526, MPS 801, MPS 802, MPS 803 típusú nyomtatók. Sajnos ezek csak az angol karakterkészletet ismerik, ezért hiányoznak az ékezetek. (Megjegyzés: az eredeti könyv megjelenése óta már Magyarországon is kapható az MPS 1000-es.) Ezért a

könyvben ismertetjük a Centronics-illesztésű nyomtatók csatlakoztatását is. Ezeknek a nyomtatóknak az az előnyük, hogy más számítógépekkel is együtt tudnak működni, (és az írásképük közel grafikus minőségű). Annak, aki a Plus/4-eshez még nem szerzett be lemezmeghajtó egységet, a 1551-es típust ajánljuk, mivel ez lényegesen gyorsabb mint a 1541-es.

A most következő fejezetekben ismerkedjünk meg néhány alapfogalommal.

2. A számítógép általános felépítése

A számítógép működése a legtöbb ember számára sokkal misztikusabbnak tűnik, mint amilyen valójában. Nézzük hát meg, hogy mit is rejt ez a doboz.

Alapvetően mindenkinek, aki valamilyen formában foglalkozik a számítógéppel, illik tisztában lennie azzal, hogy mit jelentenek az olyan fogalmak, mint a RAM, ROM, vagy az operációs rendszer. Lehet, hogy ezek első hallásra ijesztőnek hangzanak, de mégsem olyan bonyolult ez az egész, mint sokan gondolnák. A számítógép alapvetően a következő elemekből épül fel: egy írható/olvasható tárból, egy csak olvasható tárból, egy bemeneti/kimeneti egységből és a számítógép szívéből, a mikroprocesszorból. Tulajdonképpen ennyi az egész. Azzal, hogy egy működésre képes számítógépben ezek az egységek kapcsolástechnikailag hogyan épülnek fel, nem is kell törődnünk. Az előzőekben említetteket azonban nézzük meg egy kicsit közelebbről, hiszen az olyan megnevezéssel, mint pl. csak olvasható tár, nem sokat tudunk kezdeni.

A könyvben használt megnevezések közül vizsgáljuk meg itt a következőket:

RAM Random Access Memory, írható és olvasható tár:
A RAM a (felhasználói) programok és az adatok tárolására szolgáló, úgynevezett munkatár.

ROM Read Only Memory, csak olvasható tár:
a ROM az operációs rendszert, a BASIC interpretert és a Plus/4-esbe beépített szoftvert tárolja.

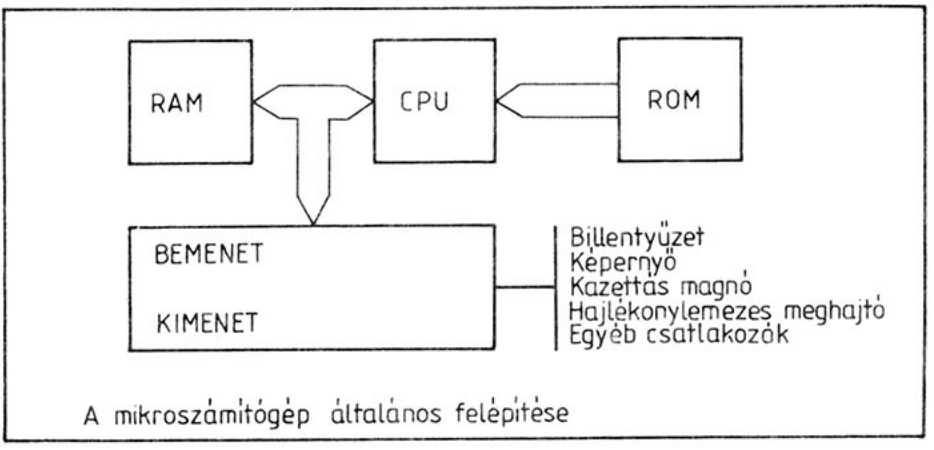
CPU Central Processing Unit, a mikroprocesszor:
a Plus/4-esben levő mikroprocesszor típusa 7501, amely kompatíbilis a 6502 típusú mikroprocesszorral.

Készítettünk egy rajzot, amely vázlatosan ábrázolja a számítógép általános felépítését (2.1. ábra).

A könyv további részében találkozunk még néhány speciális kifejezéssel, de a ROM, RAM és a CPU már elegendő a számítógép alapvető működésének megértéséhez.

A számítógép kétségkívül legfontosabb alkatrésze a CPU. A CPU a számítógépnek az az egysége, amelyik számokat tud összeadni, kivonni, összehasonlítani, és logikai műveleteket tud végezni. Ezen kívül el tudja olvasni a ROM-ban vagy a RAM-ban tárolt információkat, és ezeket el tudja helyezni a saját regisztereiben. Természetesen arra is képes, hogy információkat írjon a RAM-ba, ill. információkat továbbítson a bemeneti/kimeneti egységhez.

Talán meglepőnek tűnik, de a CPU nem tud pl. szorozni (természetesen csak a hétköznapi értelemben nem). Amit viszont csak a mikroprocesszor tud, az az a fantasztikus sebesség, amellyel a kapott utasításokat feldolgozza. A CPU megszakítás



2.1. ábra A Plus/4-es hardverfelépítése

nélkül, mindig feldolgoz valamilyen programot. Amikor bekapcsoljuk a Plus/4-et, a CPU a ROM egy pontosan meghatározott helyéről (a $FFFC és $FFFD címekről) beolvassa az ott tárolt információt, és elkezdi a gépben tárolt programok feldolgozását. Ezek gépi kódú programok, és nem tévesztendők össze a BASIC programokkal. Egy későbbi fejezetben a Plus/4-es gépi kódú programozásával is foglalkozunk.

Maradjunk azonban egyelőre a BASIC nyelvű programozásnál. A BASIC egyike az úgynevezett magas szintű programozási nyelveknek, amelyek lehetővé teszik, hogy a programutasításokat az ember számára érthetőbb formában írhassuk meg. A mikroprocesszor viszont csak a gépi kódban megírt utasításokat érti meg. Azért, hogy a gép mégis programozható legyen BASIC nyelven, a ROM tartalmaz egy úgynevezett interpreter programot, amely értelmezi a BASIC nyelvű utasításokat, és ezeket a gép számára közvetlenül érthető, gépi kódú utasításokra fordítja le. Ez egyébként az a program, amely a Plus/4-es bekapcsolásakor a képernyőn az alábbi formában jelentkezik be:

```
COMMODORE BASIC V3.5 60671 BYTES FREE
3-PLUS-1 ON KEY F1
```

A kiírás szerint a gépbe beépített BASIC interpreter (értelmező) típusjele a V3.5. Ez az interpreter értelmezi és fordítja le a BASIC utasításokat gépi kódú utasításokra.

Foglaljuk össze: a ROM olyan tár, amelynek a tartalmát csak olvasni lehet, és ez a tartalom nem változtatható meg. A programozó, ill. a felhasználó a saját programját és az adatait a RAM-ban és a bemeneti/kimeneti egységben helyezheti el. A RAM úgynevezett „felejtő" tár, ami azt jelenti, hogy a benne tárolt információk a tápfeszültség megszakadásakor (pl. a számítógép kikapcsolásakor) elvesznek.

A bemeneti/kimeneti egység – elterjedt angol nevén az I/O egység (Input, Output) – teszi egyáltalán lehetővé, hogy a számítógépen dolgozhassunk.

Az egyes egységek közötti összeköttetéseknek külön elnevezésük van. Nézzük először az úgynevezett címbuszt. A címbuszon keresztül a tár minden egyes rekeszéhez külön-külön lehet „fordulni". A tárban levő rekeszek folyamatosan sorszámozva vannak. A címbuszt címvezetékek alkotják, és az ezeken levő jelek állapotától füg-

gően a tár bármely, meghatározott rekeszét ki lehet választani. (A Plus/4-es címbusza 16 vezetékből áll.)

Ebből azonban a CPU még nem tudja, hogy a kiválasztott rekeszben milyen érték van. Ezt végzi az adatbusz, amely nyolc vezetékből áll. Ennek megfelelően beszélünk 8 bites mikroprocesszorról. Az előzőek szerint kiválasztott rekesz most a tartalmát az adatbuszra helyezi, és így olvassa el a CPU a tárban levő adatokat.

A számítógépben ezeken kívül még számos más vezeték is van (pl. a vezérlő busz), amelyekkel azonban e helyütt nem kívánunk foglalkozni. Az érdeklődőknek ajánljuk a szakirodalom tanulmányozását.

Már olvastuk a könyvben, hogy a mikroprocesszor „ismeri" az összeadást és a kivonást, de vajon hogyan tud szorozni és osztani? A mikroprocesszor ezeket a műveleteket valóban nem tudja végrehajtani, de az operációs rendszerben és a BASIC interpreterben vannak olyan programok, amelyekkel ez a probléma megoldható. Erről részletesebben majd a 20. és 21. fejezetben szólunk.

2.1. A LABDAJÁTÉK PÉLDAPROGRAM

Az otthoni számítógépeket – így minden bizonnyal a Plus/4-est is – az emberek többsége elsősorban játékszerként használja. Játsszunk hát mi is egy kicsit. Erre a célra írtuk a LABDAJÁTÉK programot. Meg kell vallanunk, hogy nem veszi fel a versenyt egy „profi" játékprogrammal, de azért el lehet vele szórakozni.

A képernyőn egy labda pattog, amelyet egy ütővel el kell találnunk. Az ütőt a „Z" billentyűvel balra, a „/" billentyűvel pedig jobbra mozgathatjuk. A játék a szóköz billentyűvel indítható, és egy játékban öt labda használható fel.

Írjuk be a gépbe a programot, és az első indítás előtt feltétlenül tároljuk lemezen vagy kazettán. Most már elindíthatjuk a

RUN

parancs beírásával. A program számos, még ismeretlen BASIC utasítást tartalmazhat. Sebaj, hiszen még előttünk az egész könyv!

```
5 REM LABDAJATEK
10 COLOR0,1:COLOR1,2
15 FORI=1630TO1724:READI$:POKEI,DEC(I$):NEXT:SYS1630
20 PRINTCHR$(147)
25 CHAR,5,0,"INDITAS: SZOKOZ BILLENTYUVEL"
30 CHAR,2,2,"UTO JOBBRA: /    UTO BALRA: Z"
35 FORI=1TO1000:NEXTI:PRINTCHR$(147)
40 CHAR,3,0,"KISERLETEK: 1":K=1
50 CHAR,20,0,"PONTOK: 0":P=0
60 FORI=0TO39:POKE3112+I,100:POKE3152+I,45:NEXT
65 IFK=6THEN170:ELSECHAR,14,0,STR$(K)
70 GETKEYA$:IFA$<>" "THEN70
80 XS=INT(RND(0)*40):YD=1:Y1=3:X1=XS:X=X1:Y=Y1
90 XD=INT(RND(0)*3-1)
100 POKE3072+X+Y*40,32:POKE3072+X1+Y1*40,81
105 X=X1:Y=Y1
110 X1=X1+XD:IFX1<00RX1>39THENXD=-XD:GOTO110
120 Y1=Y1+YD:IFY1<2THENYD=-YD:GOTO120
130 IFPEEK(3072+X1+Y1*40)=119THENYD=-YD:Y1=Y1+YD:GOTO90
140 IFPEEK(3072+X1+Y1*40)=45THENP=P+1:CHAR,28,0,STR$(P):YD=-YD:GOTO90
```

```
150 IFY1>24THENK=K+1:POKE3072+X+Y*40,32:GOTO65
160 GOTO100
170 CHAR,0,9,"ESBEN A JATEKBAN ";PRINTP;" PONTOT ERT EL.":PRINT
180 POKE239,0:INPUT"MEG EGY JATEK (I/N)";IN$:IFIN$="I"THEN20:
GOTOSYS65526
200 DATA 78,A9,77,8D,14,03,A9,06,8D,15,03
210 DATA A9,C0,85,D0,A9,0F,85,D1,A9,01,85
220 DATA D2,58,60,C6,D2,A5,D2,D0,0F,A9,04
230 DATA 85,D2,20,11,DB,C9,2F,F0,07,C9,5A
240 DATA F0,1A,4C,0E,CE,A5,D0,C9,E5,F0,F7
250 DATA A0,00,A9,20,91,D0,A0,03,A9,77,91
260 DATA D0,E6,D0,4C,8C,06,A5,D0,C9,C0,F0
270 DATA E0,A0,02,A9,20,91,D0,C6,D0,A0,00
280 DATA A9,77,91,D0,4C,0E,CE
```

3. Számrendszerek

A számítógépek fejlesztése során többek között arra a kérdésre is választ kellett adni, hogy ezek a gépek saját magukon belül hogyan is ábrázolják a számokat. A fejlesztők már a munka korai fázisában felismerték, hogy erre a célra a kettes (bináris) számrendszer lehet alkalmas. Már az első, programozható számítógépek – amelyek egyébként több ezer jelfogóból álltak – is ezen az elven működtek.

Ma a világon működő valamennyi mikroszámítógép a kettes számrendszert használja. Ebben a számrendszerben csak két számjegy van, a 0 és az 1. A mindennapi életben használatos tízes számrendszerben 10 különböző számjegy van. A számítógép esetében viszont igen körülményes lenne, hogy egy vezetéken 10 különböző állapotot hozzunk létre. Ennél sokkal egyszerűbb lehetőséget kínál a kettes számrendszer két állapota, nevezetesen az „1"-nek tekintett állapot, amikor van valamekkora feszültség a vezetéken, és a „0"-nak tekintett állapot, amikor nincs, vagy az előbbihez képest legalábbis jóval kisebb a feszültség.

Egy táblázatban egymás mellé írtuk a három legfontosabb számrendszert, a bináris, a decimális és a hexadecimális számrendszert. Ismét egy új fogalom bukkant fel, a hexadecimális számrendszer. A számítástechnikában a bináris számok áttekinthetőbb ábrázolására használják a hexadecimális, azaz a 16-os alapú számrendszert.

Bináris	Decimális	Hexadecimális
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	10	A
1011	11	B
1100	12	C
1101	13	D
1110	14	E
1111	15	F
10000	16	10
11110001	241	F1
1111111111111111	65 535	FFFF

Megnézhetjük, hogy a Plus/4-es hogyan ábrázolja a képernyőn a hexadecimális számokat. Írjuk be a gépbe a következő parancsot:

MONITOR

és nyomjuk le a RETURN billentyűt.

Most az úgynevezett monitorprogramban vagyunk, amely az operációs rendszer része (a monitorprogrammal a gépi kódú programozásról szóló fejezetben foglalkozunk). Írjuk be:

M FFOO

A képernyőn egy sereg szám és betű jelenik meg. A jobb szélen levő tíz oszloptól eltekintve a képen hexadecimális formában ábrázolt értékeket látunk. A hexadecimális számok ábrázolásához a decimális 10-15 közötti értékek helyett betűket (A, B, C, D, E, F) használunk. Míg a bináris számrendszerben a számolásnál a számjegyek már az 1-nél, a tízes számrendszerben a 9-nél „elfogynak", addig a hexadecimális számrendszerben csak 15-nél. A jobb érthetőség kedvéért a könyvben a bináris és a hexadecimális számokat megkülönböztető jelzéssel láttuk el.

A hexadecimális számokat

$FFOO (megkülönböztető jel a $ jel),

a bináris számokat

%01001110 (megkülönböztető jel a % jel)

formában írjuk.

A decimális számokat külön nem jelöljük.

A számítástechnikai szakirodalomban is ez az általánosan elterjedt jelölésmód.

Szokásos jelölés még:

$ helyett a szám után H-t,

% helyett a szám után B-t.

Már említettük, hogy a számítógép címbuszának „szélessége" 16, az adatbuszáé pedig 8 vezeték. A kettes számrendszer segítségével ily módon a legnagyobb ábrázolható szám a címbuszon $2^{16} = 65536$, az adatbuszon pedig $2^8 = 256$. Ez azt jelenti, hogy a címbuszon keresztül a tár 65536 darab, különböző rekeszéhez lehet külön-külön fordulni. Mellékesen megjegyezzük, hogy ez a Plus/4-esnél nem egészen van így: rafinált elektronikus kapcsolások segítségével a 16 címvezetéken keresztül még ennél is több tárrekesz érhető el.

A nyolc adatvezetéken keresztül mindegyik rekesz $2^8 = 256$ különböző értéket vehet fel (tehát a 0–255 közötti értékeket, hiszen a 0-t is be kell számítani).

Ezzel már majdnem a végére is értünk a számítógép felépítését és a számrendszereket ismertető, hosszú és bonyolult fejezetnek. Két, nagyon fontos fogalmat azonban még tisztáznunk kell.

3.1. A BIT ÉS A BYTE

A bit az angol BInary digiT (digitális szám) kifejezés rövidítése. A számítógépben a 8 adatvezeték mindegyike egy-egy bitet jelent. A bit az információ legkisebb, elemi egysége, amely két értéket vehet fel, az 1-et és a 0-át (igaz–hamis; folyik áram – nem folyik áram). Nyolc bit alkot egy byte-ot (kiejtése: bájt). A byte-okat általában hexadecimális formában ábrázolják. Nézzünk erre két példát:

Bináris 8 bit	Hexadecimális 1 byte	Decimális
%00101101	=$2D	= 45
%11110001	=$F1	=241

A fenti fogalmakat felhasználva azt mondhatjuk tehát, hogy a Plus/4-esben egy 16 bites címbusz és egy 8 bites adatbusz van.

Egy byte tehát 8 bitet jelent. Ennek alapján feltételezhetnénk, hogy a sokfelé – így a könyvünkben is – látható 1 kbyte (kilobájt) 1000 byte-ot jelent. Mivel azonban alapvetően kettes számrendszerben számolunk, az 2^{10}, azaz 1024 byte-tal egyenlő. Most már pontosan kiszámíthatjuk, hogy hány darab tárrekesz van a Plusz/4-es RAM-jában: 64 kbyte = 64*1024 byte = 65536 byte. A beépített ROM ugyancsak 64 kbyte-os, tehát 65536 byte-os.

Ennyit a bitekről és a byte-okról. Írjunk most egy programot, amely egy tetszőleges decimális számot bináris számmá alakít át. Végül is arra való a számítógép, hogy programokat futtassunk rajta. Az egyes programsorokat pontosan úgy írjuk be, ahogyan az a könyvben van. Mindegyik sor végén nyomjuk le a RETURN billentyűt.

```
10 PRINT CHR$(147)
20 PRINT "IRJON BE EGY DECIMALIS SZAMOT"
30 INPUT D : A = D
40 D = D / 2
50 IF D = INT (D) THEN H$ = "0" + H$ : GOTO 70
60 H$ = "1" + H$
70 D = INT (D) : IF D > 0 THEN GOTO 40
80 PRINT A; "=";H$
90 H$ = ""
100 PRINT
110 GOTO 20
```

Ha végeztünk a program beírásával, akkor indítsuk el a

RUN

paranccsal, és a RETURN lenyomásával. Ha nem követtünk el gépelési hibát, akkor a programnak most működnie kell. Ha viszont a képernyőn olyan szöveg jelenne meg, hogy „ERROR IN...", akkor a programot a

LIST

paranccsal újra ki kell listázni a képernyőre, és a könyvben levő programmal összehasonlítva ki kell javítani a hibát. Ebben egyébként a számítógép is a segítségünkre van. Hiba esetén az ERROR IN üzenet mögé kiír ugyanis egy számot. Ez a szám annak

a programsornak a sorszáma, amelyben hiba lehet, ill. amelyik sorhoz érkezve a gép a hibát felfedezte. De erről majd a 19. fejezetben. Most a program felszólít bennünket:

ÍRJON BE EGY DECIMÁLIS SZÁMOT
?

A szöveg utáni sorban egy kérdőjel áll. A számítógép ezzel jelzi, hogy adatbevitelre vár. Teljesítsük a kérését, és írjuk be a következő számot (ne felejtsük el utána a RETURN billentyű lenyomását):

241

A számítógép kiszámítja az ennek megfelelő bináris számot, és kiírja a képernyőre:

11110001

Most beírhatunk egy újabb számot. Próbáljuk ki néhányszor. Akár nagyobb számokat is beírhatunk. Meg kell mondani, hogy 9 számjegynél nagyobb számok esetén a gép már nem számol pontosan.

Elemezzük röviden a programot. Mivel még jóformán az egész könyv előttünk van, néhány dolog biztosan idegennek tűnik. Ennek ellenére érdemes elolvasni, hogy lássuk, hogyan épül fel egy program.

A 10-es programsorban levő utasítás a képernyőt törli.

A 20-as programsor kiíratja a szöveget a képernyőre.

A 30-as sorban a program egy szám beírására vár, és ezt a számot két változóhoz rendeli hozzá (A és D).

A program 40-es sora ezt a számot elosztja kettővel.

Az 50-es sorban megvizsgáljuk, hogy a kapott eredmény páros, vagy páratlan szám. Ha páros, akkora bináris számot ábrázoló változóba – esetünkben a H$-ba – jobbról kezdődően egy „0"-át írunk. Ha nem, akkor átugrunk a 70-es programsorra.

Ha az 50-es sorban páratlan számot kapunk, akkor a 60-as sor a H$-ba egy „1"-est ír be.

A 70-es sorban a decimális számot kerekítjük, majd ezután megállapítjuk, hogy az értéke 0, vagy nagyobb mint 0. Ha a szám nagyobb mint 0, akkor a program a 40-es sorra ugrik. Ellenkező esetben a program végrehajtása a 80-as sorban folytatódik.

A program 80-as sorában kiíratjuk a megadott decimális számot, és az ennek megfelelő bináris számot.

A 90-es sorban töröljük a bináris számot tartalmazó változót.

A 100-as sorral egy üres képernyősort íratunk ki. Ez áttekinthetőbbé teszi a képernyőn való ábrázolást.

Végül a 110-es sorban arra utasítjuk a számítógépet, hogy ugorjon vissza a 20-as sorra.

Hát ez csak egyszerű!? Vegyük észre, hogy a számítógép milyen gyorsan számítja ki az eredményt! Zsebszámológéppel, vagy papírral és ceruzával ez bizonyára sokkal tovább tartana.

Még egy tipp: Bizonyára unalmas lenne, ha mindannyiszor, ahányszor egy ilyen számátalakítást el akarunk végezni, ezt a programot be kellene írni a gépbe. Tároljuk ezért a programunkat kazettán vagy lemezen. Ehhez először is meg kell állítanunk a programot. Ezt azzal érhetjük el, ha lenyomjuk a RUN/STOP billentyűt. Próbáljuk ki, és ha nem megy...

Az ok a 30-as programsorban található. Ebben a sorban a BASIC nyelvű INPUT utasítás szerepel. Ez az utasítás figyelmen kívül hagyja a RUN/STOP billen-

tyű lenyomását. A programunkat tehát akkor kell megszakítani, amikor éppen nem az INPUT utasítás feldolgozása folyik, hanem, mondjuk a bináris szám kiszámítása. Írjunk be tehát egy számot, vagy csak nyomjuk le a RETURN billentyűt, majd ezután rögtön a RUN/STOP billentyűt. Lehetséges, hogy az első kísérletre nem áll meg a program, de ötödszörre biztosan menni fog, és a számítógép kiírja, hogy

BREAK IN 60 (az utolsó programsor, amelyet feldolgozott).

Ehhez kapcsolódóan még egy tipp: Ha olyan programmal dolgozunk, amely nem állítható meg, nyomjuk meg a RUN/STOP billentyűt, és tartsuk lenyomva. Ezután nyomjuk le a RESET billentyűt, majd engedjük el. Ha ezek után a monitorprogram jelentkezik be, akkor elengedhetjük a RUN/STOP billentyűt is. Most már csak egy „X"-et kell beírni, és a RETURN billentyűt lenyomni. A számítógép most újra készen áll a BASIC programozásra. Ha nem használunk lemezegységet, akkor a RUN/STOP billentyű nyomva tartása mellett nyomjuk le valamelyik shift gombot. Ha először a shift gombot engedjük el, akkor a program megáll.

Most a programot tárolhatjuk. Ha kazettás magnónk van, akkor írjuk be:

SAVE "DECIMAL/HEXA"

Ha a programot lemezen tároljuk, akkor ezt kell beírni:

DSAVE "DECIMAL/HEXA"

Ezzel a programunkat biztonságba helyeztük, és bármikor visszatölthetjük a számítógépbe.

3.2. A HEX$ ÉS A DEC UTASÍTÁS

A Plus/4-ben még további lehetőségek is vannak az egyes számrendszerek közötti átszámításokra. Nevezetesen a számítógép egy egyszerű BASIC utasítással átszámítja a hexadecimális számokat decimális számokká:

Ez az utasítás a

HEX$(n)

(az n egy 0 és 65535 közötti decimális szám).

Írjuk be pl ezt:

PRINT HEX$ (241)

A képernyőn most ez jelenik meg:

00F1

Ez egy olyan hasznos, és hatékony utasítás, amelyet számos más mikroszámítógép nem ismer. A Plus/4-es BASIC-je ezt a műveletet fordítva is el tudja végezni:

DEC ("hex")

(a hex egy 0 és FFFF közötti hexadecimális szám)

Próbáljuk ki ezt például ezzel:

```
PRINT DEC ("00F1")
```

és látni fogjuk, hogy a számítógép a helyes eredményt írja ki:

241

Ennyit a számrendszerekről. Nézzünk meg most egy újabb mintaprogramot.

3.3. LOTTÓSZÁMOK PÉLDAPROGRAM

A következő program a véletlenszámokat előállító generátor alkalmazásával a lottójáték 90 száma közül 5 számot „húz ki". Kívánunk minden héten sok-sok találatot!

```
10 REM LOTTO
20 DIMA(90)
30 PRINTCHR$(147)
40 PRINTSPC(12)"**LOTTOSZAMOK**"
50 FORI=1TO5
60 LI=INT(RND(0)*90+1)
70 IFA(LI)=1THEN60
80 A(LI)=1
90 L(I)=LI
100 NEXT
110 FORI=1TO5:A(L(I))=0:NEXT
120 FORJ=1TO5
130 FORI=1TO4
140 IFL(I)>L(I+1)THENX=L(I+1):L(I+1)=L(I):L(I)=X
150 NEXTI,J
160 CHAR,3,15,""
170 FORI=1TO5:PRINTL(I);"  ";:NEXTI
180 CHAR,5,24,"MEG EGY TIPPET (I/N)?"
190 GETKEYA$:IFA$="I"THEN30
200 END
```

4. Számolás a számítógéppel

Joggal kérdezheti az Olvasó, hogy minek kell ez a fejezet. Végül is az csak természetes, hogy egy számítógéppel lehet számolni. Ez persze igaz, de azért érdemes ezt egy kicsit közelebbről is megvizsgálni. A Plus/4 természetesen ismeri a számtani alapműveleteket, így az

- összeadást (+),
- kivonást (−),
- szorzást (*),
- osztást (/),
- hatványozást (↑),

Az bizonyára világos, hogy az alábbi feladatok megoldása a műveletek végrehajtási sorrendjétől függően nem azonos:

2+10*5 = 52

vagy

2+10*5 = 60

Természetesen az első megoldás a helyes. Bizonyára ismerik a szabályt, miszerint a szorzás és az osztás megelőzi az összeadást és a kivonást. E szabály szerint számol a számítógép is.

A műveletek végrehajtásánál vannak még további sorrendiségi szabályok is, így a hatványozás mind a négy alapműveletet megelőzi.

Ha a példánkban először a 2-t és a 10-et össze akartuk volna adni, majd az eredményt ez után 5-tel szorozni, akkor ezt kellett volna írnunk:

(2+10)*5 = 60

Először a zárójelben levő műveletet kell elvégezni. Több zárójel is használható:

((2+10)*5)↑3 = 21600

Ha nem tennénk ki a második zárójelpárt, akkor a számítógép először a 2+10 és az 5↑3 műveleteket végezné el, majd a kapott eredményeket összeszorozná (12*125). Így a végeredmény 1500 lenne. A könyvben előfordul, hogy meglehetősen bonyolult számításokat végzünk. Itt nagyon kell ügyelni arra, hogy milyen sorrendben végezze el a műveleteket a számítógép.

A számtani műveleteket a PRINT paranccsal közvetlenül végrehajthatjuk. E műveletek mellett vannak még összehasonlító műveletek, melyeket a 6. fejezetben tárgyalunk. E műveletek eredményei is közvetlenül kiírathatók, vagy változókba átvehetők.

4.1. AZ SGN, ABS, SQR, EXP, LOG FÜGGVÉNYEK

Egy szám előjelének megállapítására alkalmas a SGN (szignum) függvény.
Szintaxisa:

SGN (x)

ahol az x lehet változó, vagy akár egy bonyolult képlet is. A függvény értéke −1; 0 vagy 1 aszerint, hogy x negatív, 0 vagy pozitív.

Egy szám abszolút értékének képzésére az ABS függvény alkalmas. A parancs a negatív számok előjelét pozitívra változtatja.
Szintaxisa:

ABS (x)

Az x itt is lehet konkrét érték, változó, vagy összetett kifejezés. Az ABS eredménye mindig pozitív vagy 0.

A Plus/4 négyzetgyököt is tud vonni. Ezt a műveletet végzi el a BASIC nyelv SQR parancsa.
Szintaxisa:

SQR (x)

Itt az x nem lehet negatív szám, (hiszen olyan valós szám nincs, amit négyzetre emelve eredményül negatív számot kapnánk.)

Az e (természetes logaritmusrendszer alapszáma) szám (e = 2,7182818...) hatványait számítja ki az EXP művelet.
Szintaxisa:

EXP (x)

Az x azt adja meg, hogy hányszor kell az e számot önmagával megszorozni. Például:

EXP (5) = e * e * e * e * e = 148,413159

A LOG (x) függvény egy szám természetes alapú logaritmusát számítja ki. A logaritmus az a kitevő, amelyre az e számot (2,7182818...) kell emelni ahhoz, hogy az x-et megkapjuk.
Szintaxisa:

LOG (x)

Itt is behelyettesíthető az x helyére egy szám vagy egy összetett kifejezés, csak arra kell vigyázni, hogy a zárójelbe kerülő szám ne legyen negatív vagy 0. Ebben az esetben ugyanis a számítógép hibát jelezne.

Vannak olyan logaritmusok, amelyeknek az alapja nem az e szám. Ilyen például a 10-es alapú logaritmus. A Plus/4 a 10-es alapú logaritmust közvetlenül nem tudja kezelni. Megfelelő összefüggések alapján ezzel is végezhetünk számításokat. Így:

lg x = 0,434294 * ln x (a tízes alapú logaritmus számítása)
ln x = 2,302585 * lg x (tízes alapú logartimus átszámítása természetes alapú logaritmussá)

ahol

ln = az e alapú logaritmus jelölése
lg = a tízes alapú logaritmus jelölése

4.2. A SZÖGFÜGGVÉNYEK

A Plus/4 természetesen szögfüggvényeket is tud számítani. A következő BASIC parancsokkal:

SIN(x)
COS(x)
TAN(x)
ATN(x)

Az x helyére a kívánt szögértéket kell beírni radiánban.
Ehhez van néhány megjegyzésünk: Az közismert, hogy egy szög értéke 0 és 360° közé eshet. Az azonban előfordulhat, hogy egy programban 360°-nál nagyobb szöggel kell számolni, pl. ha egy kört másfélszer kell megrajzolni. Pl. A CIRCLE parancsban 360°-nál nagyobb szöget is meg kell tudnunk adni anélkül, hogy ezt a számítógép hibának tekintené. A számítógép a 360°-nál nagyobb szögekből automatikusan levon 360°-ot vagy ennek többszörösét.

A szöget nem fokokban kell megadni, hanem először át kell számítani ívmértékre. Ezt a következő képlettel végezzük el:

szög * PI/180

Az átszámítás eredménye 0 és 360° közötti szögek esetén 0 és 2 PI közé esik. A képletet a SIN, COS és a TAN függvényeknél a zárójelben közvetlenül argumentumként is megadhatjuk.

Az ívmérték (egysége a radián) az egységsugarú körben a 0 és 360° közötti szögeknek megfelelő körív hossza.

Az ATN (arkusztangens) a tangens inverz függvénye. A függvény argumentumaként (y) tehát egy szög tangensének értékét kell megadni. A függvény eredménye a szög ívmértéke.

Szintaxisa:

ATN(y)

Ezzel be is fejezzük a matematika világába tett kirándulásunkat.

Ne feledkezzünk meg arról, hogy a Plus/4 nem számol olyan pontosan, mint egy zsebszámológép, mivel a BASIC értelmező (interpreter) közelítő képletekkel számol, amiből elkerülhetetlenül pontatlanságok származnak.

4.3. FÜGGVÉNYEK MEGHATÁROZÁSA A DEF FN SEGÍTSÉGÉVEL

Ezzel a paranccsal komplett képleteknek nevet adhatunk, és a későbbiekben ezekre a képletekre a nevükön keresztül hivatkozhatunk. Az elvet a változók használatánál megismerhettük. Ha egy olyan programunk van, amelyben egy hosszú képlettel sokszor kell számolni, akkor e parancs alkalmazásával begépelési időt, és nem utolsósorban tárhelyet takaríthatunk meg. Egy programban több függvényt is definiálhatunk.

Szintaxisa:

DEF FN név (változó) = képlet

név = a függvény nevére ugyanaz vonatkozik, mint a változók nevére: legalább egy karakter hosszúságúnak kell lennie, és az első karakternek betűnek kell lennie. A név intarziaként nem tartalmazhat BASIC utasításszót. A változó nevét zárójelbe kell tenni. Annak nincs jelentősége, hogy a változó a függvényben szerepel-e vagy nem. Példa:

```
DEF FN SZOG (SZ) = ABS(SIN(SZ * 3,1415/180))
```

A DEF FN csak programon belül használható. Parancs módban kiadva hibajelzést kapunk.

A példában szereplő SZOG nevű függvényt most a programban bárhol behívhatjuk az FN SZOG (SZ) beírásával. Általánosan az

```
FN SZOG (x)
```

beírásával, az X lehet konkrét érték, egy változó vagy akár egy összetett kifejezés is. A képlet több változót is tartalmazhat, de a fent leírt módon csak egy változóhoz lehet egy értéket automatikusan hozzárendelni.

4.4. AZ RND PARANCS

Az RND paranccsal véletlen számokat állíthatunk elő. Az előállított számok 0-nál nagyobbak és 1-nél kisebbek, az értékük pedig „véletlen". A véletlen számok többek között játékprogramokban használhatók pl. a kockadobás szimulálására. A statisztikai számításokban is használnak véletlen számokat. Másik alkalmazási terület lehet pl. adatok rejtjelezése.

Szintaxisa:

RND(x)

ahol az x tetszőleges szám vagy változó.

Az x értékére három fontos számtartományt érdemes megvizsgálni: a pozitív számokat, a negatív számokat és a 0-t. Nézzük akkor egy kicsit közelebbről a véletlen számokat. Írjuk be:

```
10 PRINT RND(1)
20 GOTO 10
```

Most egy hosszú számsor fut előttünk a képernyőn. Első pillantásra úgy tűnhet, hogy ezek valóban „véletlen" számok. De vajon hogyan lehetséges, hogy egy olyan, digitális elven működő, előre meghatározott programokat feldolgozó készülék, mint a számítógép, „véletlen számokat" állítson elő?

Az igaz, hogy a gép a „véletlen számokat" pontosan meghatározott képletekkel számítja ki. Ezen kívül a véletlen számok egy bizonyos idő után ismétlődnek. Ennek bizonyítására kapcsoljuk a számítógépet egy pillanatra ki, majd kapcsoljuk be újra, és írjuk be:

```
PRINT RND(1)
```

A képernyőn most a következő szám jelenik meg:

```
1.07870447E-03
```

A RND (pozitív szám) mindig ugyanazzal az értékkel indul, és a folytatás is mindig azonos. Ennek az az oka, hogy a gép a véletlen számokat mindig az előre megadott képletek szerint számítja ki. Nézzük meg most a RND függényt negatív argumentumok esetén. Írjuk be:

```
PRINT RND(-1)
```

A képernyőn a következő szám jelenik meg:

```
2.99196472E-08
```

Most valami különleges következik: írjuk be ismét az előző parancsot, és látjuk, hogy ugyanazt az eredményt kapjuk.

A gép mindegyik negatív számra egy egészen pontosan meghatározott véletlen számot állít elő. Ezt bármely negatív számra kipróbálhatjuk. Így pl. a RND(-220.999) eredménye 0.910092327.

Véletlen számok előállítására a harmadik lehetőség az, ha a RND argumentuma zérus. RND(0) esetén a Plus/4 a véletlen számokat a belső órája segítségével állítja elő. A

```
PRINT RND(0)
```

parancs mindig különböző véletlen számot eredményez, bár pontosan véve természetesen ezek a véletlen számok is mindig ugyanazzal az értékkel kezdődnek. A véletlenszerűséget itt az okozza, hogy a parancsot aligha lehet egy bekapcsolást vagy RESET-et követő, mindig pontosan azonos időpillanatban kiadni.

Véletlen számok előállítására első pillantásra a RND(0) parancs tűnik a legalkalmasabbnak.

Vizsgáljuk most meg, hogy a véletlen számok egyenletes eloszlásúak-e, vagy vannak-e bizonyos kivételezett számok. Írjunk be ehhez egy rövid programot:

```
10 REM VELETLEN SZAMOK ELOSZLASA
20 PRINTCHR$(147)
30 X=INT(RND(1)*10)
40 Z(X)=Z(X)+1
50 PRINTCHR$(19)
60 FORT=0TO9
70 PRINTT;Z(T)
80 NEXT
90 GOTO30
```

Ha futtatjuk a programot, akkor látni fogjuk, hogy a véletlen számok eloszlása igencsak egyenletes. A program többszöri futtatása után is erre az eredményre jutottunk.

Ebben a programban a 0 és a 9 közötti véletlen számok előfordulásának gyakoriságát írtuk ki. Változtassuk meg most a 30-as sort a következőre:

```
30 X = INT(RND(0) * 10)
```

Az RND(0)-nak az az előnye, hogy mindig más-más az indulási értéke.

4.5. AZ INT PARANCS

Az INT parancs lefelé kerekít. Az eredmény egy egész szám. Például:

```
A = 10.34
PRINT INT(A)
```

az eredmény 10. De:

```
A = -10.34
PRINT INT(A)
```

az eredmény -11 lesz. Arra is lehetőség van, hogy a következő egész számra kerekítsünk (a szokásos kerekítés: 0,5-től feljebb felfelé, egyébként lefelé kerekítünk). Ehhez azonban már egy kis programot kell írnunk:

```
10 INPUT A
20 PRINT INT(A+.5)
30 GOTO 10
```

Ebben a mintaprogramban a bekért számhoz hozzáadunk 0,5-et, majd lefelé kerekítünk. Így elérjük a szokásos fel- vagy lefelé kerekítést.

Írjuk be pl. az 1,4 értéket (a beírás formája: 1.4). A számítógép ehhez hozzáad 0,5-et, az eredmény most 1,9. Az INT parancs lefelé kerekít, így az eredmény 1 lesz. Ha viszont 1,5-et írunk be, akkor a számítógép a 2,0 értéket számítja ki (1,5+0,5 = 2,0), így a végeredmény 2 lesz.

Az INT paranccsal azokat a számokat, amelyekben a tizedespont után több érték is van, a tizedespontot követően, pontosan meghatározott hosszban írathatjuk ki. Lássunk erre egy példát: írassunk ki véletlen számokat úgy, hogy ezek a tizedespont után csak két számjegyet tartalmazzanak. A véletlen számok a 0–10 intervallumba essenek.

```
10 X = RND(0)*10
20 PRINTX,,INT(X*100)/100
30 GOTO 10
```

A 10-es sorban állítjuk elő a véletlen számokat, és ezeket 10-zel megszorozzuk. Így a 0–10 intervallumba eső véletlen számokat kapunk. Az előző fejezetben láttuk, hogy a véletlen számok többnyire 9 számjegyből állnak.

A 20-as sorban ezeket a számokat 100-zal szorozzuk, az INT-tel kerekítjük, majd 100-zal osztjuk. Az eredeti véletlen számot a képernyőn balra, a kerekített számot pedig jobbra kiírjuk.

A 100-zal való szorzás, majd az ezt követő, 100-zal való osztás azt eredményezi, hogy a tizedespont után két számjegy áll. Ha csak egyet akarnánk kiíratni, akkor 10-zel kellene szorozni, majd ezután 10-zel kellene osztani. Három jegy esetén 1000-rel stb.

Olvassuk el a kerekítés témaköréhez a 11. fejezetet is. Ott tárgyaljuk a PRINT USING parancsot, amely szintén használható számok kerekítésére.

Ha például a és b közötti egész számokat akarunk előállítani, akkor azt a következőképpen végezhetjük;

INT(RND(0)*(b − a + 1 + a).

Példa: dobókocka esetén

INT(RND(0)*6 + 1)

− 1,0,1 esetén

INT(RND(0)*3 − 1)

4.6. PÉLDAPROGRAM: SZÁMTOTÓ

Nézzünk meg most egy példaprogramot, amely véletlen számokkal dolgozik. Talán ismerős a SZÁMTOTÓ nevű játék: az egyik játékos gondol egy számot, a másik pedig találgatással és kombinálással megpróbálja ezt a számot kitalálni.

A programunk először megkérdezi, hogy hány jegyű legyen a kitalálandó szám. A kérdésre egy 2–6 közötti szám megadásával válaszolhatunk. Ezután a program azt kérdezi, hogy a számban hányféle számjegy fordulhat elő. A válaszunk 2–8 között lehet. Most kezdődhet a találgatás. A program összesen tíz kísérletet engedélyez. Egy-egy szám beírását a RETURN billentyű lenyomásával kell a gép tudomására hozni. A számítógép az így beírt számot összehasonlítja az általa „ gondolt" számmal. Ha a beírt szám valamelyik számjegye a véletlen szám egyik számjegyével helyre és értékre is megegyezik, akkor a gép egy fekete kört, ha csak értékre, de helyre nem, akkor egy üres karikát rajzol ki.

A KS változó a kísérletek számát tartalmazza. Ha a játékban már némi gyakorlatra tettünk szert, akkor csökkenthetjük a kísérletek számát. Ehhez az 55-ös programsort kell megváltoztatni.

Jó szórakozást a SZÁMTOTÓ-hoz.

```
10 REM SZAMTOTO
15 PRINT CHR$(147);"              ***SZAMTOTO***"
20 INPUT "HANY JEGYU LEGYEN A SZAM(2-6)";JS
30 IF JS<2 OR JS>6 THEN 20
40 INPUT"HANY FELE SZAMJEGY LEHET(2-8)";JV
50 IF JV<2 OR JV>8 THEN 40
55 K=0:R=0:KS=10
60 GOSUB 200:REM VELETLEN SZAMOK
70 GOSUB 300:REM KEPERNYO
80 DO UNTIL K=KS
90 GOSUB 500:REM SZAM BEIRASA
95 X=0:R=0
100 GOSUB 800:REM VIZSGALAT
110 IF R=JS THEN CHAR,0,KS+10,"":PRINT"  JOL GONDOLKODOTT!":
```

```
PRINT:GOTO 160
120 K=K+1
130 LOOP
135 CHAR,1,KS+10,"":PRINT"SAJNOS NEM TALALTA EL..."
140 CHAR,0,KS+12,"":PRINT"A HELYES SZAM  ";
145 FORI=1TOJS:SZ$(I)=STR$(SZ(I))
150 PRINTRIGHT$(SZ$(I),1);:NEXT
155 PRINT"  VOLT.":PRINT
160 INPUT"MEG EGY JATEK   (I/N)";V$
170 IF V$="I" THEN 55:ELSE END
195 :
200 REM VELETLEN SZAMOK
205 :
210 FOR I=1 TO JS:SZ(I)=INT(RND(0)*JV)+1:NEXT I
220 RETURN
295 :
300 REM A KEPERNYO
305 :
310 PRINTCHR$(147);"              ***SZAMTOTO***":PRINT
320 PRINT" 10 KISERLET",JS;" JEGYU SZAM":PRINT
330 PRINT" 1 ES ";JV;" KOZOTTI SZAMOK":PRINT
340 PRINT"KISERLET        EREDMENY"
350 PRINT
360 FORI=6TO15
370 PRINT"NR.";I-5;TAB(7);":       :"
380 NEXTI
390 CHAR,1,KS+10,"":PRINT"KEREM A SZAMOT !"
400 RETURN
495 :
500 REM  A SZAM BEIRASA
505 :
510 DOUNTILX>JS
520 GETKEYSZ$
525 IFSZ$=CHR$(20)ANDX=0THEN520
530 IFSZ$=CHR$(20)THENSZ$=SZA$(X-1):SZA$(X)=" ":X=X-1:GOTO565
540 IFSZ$=CHR$(13)ANDX=JSTHEN560
545 IFSZ$=CHR$(13)ANDX<JSTHEN520
550 IFSZ$<"1" OR SZ$>CHR$(48+JV)THEN520
555 IFX=JS THEN520
560 X=X+1
565 IF X<1 THENX=0
570 SZA$(X)=SZ$:SZA$(X+1)=" "
580 GOSUB 700:REM KIIRAS
590 LOOP
600 RETURN
695 :
700 REM KIIRAS
705 :
710 FOR I=1 TO JS
720 CHAR,7+I,8+K,"":PRINT SZA$(I)
730 NEXT I
740 RETURN
795 :
800 REM VIZSGALAT
802 :
805 FORJ1=1TOJS:S1(J1)=SZ(J1):NEXTJ1
810 CHAR,15,8+K,""
820 FORJ1=1TOJS
830 IFS1(J1)=VAL(SZA$(J1))THENPRINTCHR$(113);
:R=R+1:S1(J1)=0:SZA$(J1)="9"
```

```
835 NEXTJ1
836 FORJ1=1TOJS
840 FORJ2=1TOJS
850 IFSI(J1)=VAL(SZA$(J2))THENPRINTCHR$(119);
:SI(J1)=0:SZA$(J2)="9":GOTO870
860 NEXTJ2
870 NEXTJ1
880 FORJ1=1TOJS:SZA$(J1)="":NEXTJ1
890 RETURN
```

5. Logikai műveletek

Foglalkozzunk most a logikai műveletekkel. A logikai műveletek a kifejezések között logikai kapcsolatokat hoznak létre. Ezek megértéséhez persze nem kell matematikusoknak lennünk. A fejezetben megismerkedünk a logikai műveletekkel, hatásukkal, és alkalmazásukkal.

Az ide tartozó BASIC parancsok:

- AND
- OR
- NOT

5.1. AND (KONJUNKCIÓ)

Kezdjük az ÉS művelettel. Definíciója: egy ÉS művelet eredménye akkor 1, ha mindkét összekapcsolandó kifejezés értéke 1.

Talán egyszerűbben érthető így: az autó motorja akkor indul el, ha a tankban van üzemanyag, és az indítókulcsot elfordítjuk.

A logikai műveletek eredményét igazságtáblákban szokás ábrázolni:

A	ÉS	B	eredmény
0		0	0
0		1	0
1		0	0
1		1	1

Ilyen kapcsolatok létrehozásával kiválóan lehet bináris számok között műveleteket végezni. Nézzünk egy példát a decimális 107 és a 210 számokkal (bináris alakban %0110 1011 és %1101 0010).

ÉS

A	%0110 1011	107	$6B
B	%1101 0010	210	$D2
eredmény:	%0100 0010	66	$42

Amint látjuk, a decimális és a hexadecimális számoknál ezzel a művelettel nem sokra mennénk. Ugyan ki mondaná, hogy a 107 ÉS 210 eredménye 66 lenne? Kissé furcsának tűnik, ugyanakkor logikus. Ne cseréljük össze az ÉS (angolul AND) műveletet az összeadás (+) művelettel!

A bináris ábrázolásmódnál viszont rögtön látjuk, hogy mi lesz az eredmény. Írjuk egyszerűen a számokat jobbról kezdődően egymás alá, és végezzük el a műveletet jegyről jegyre.

Ellenőrizzük számításunkat a számítógéppel. Írjuk be a fenti példánkat:

```
PRINT 107 AND 210
```

a képernyőn tényleg az általunk számított eredmény jelenik meg:

66.

Beírhatjuk a műveletet hexadecimális számokkal is:

```
PRINT HEX$(DEC("6B") AND DEC("D2"))
```

Megint a helyes eredményt kapjuk:

$0042.

5.2. OR (ADJUNKCIÓ)

Az OR, magyarul a VAGY művelet definíciója: Két érték közötti VAGY művelet eredménye akkor 1, ha a két érték közül legalább az egyik értéke 1. (Az eredmény akkor is 1, ha mindkét érték 1.)

Nézzünk meg egy példát:

```
PRINT 107 OR 210
```

A számítógép eredményként 251-et ír ki. Vizsgáljuk meg ezt az igazságtáblát a bináris számokon is:

A	OR	B	eredmény
0		0	0
0		1	1
1		0	1
1		1	1

A bináris számokkal:

		VAGY	
A	%0110 1011	107	$6B
B	%1101 0010	210	$D2
eredmény	%1111 1011	251	$FB

A hexadecimális számokkal nézzük meg újra a számítógépen:

```
PRINT HEX$(DEC("6B") OR DEC("D2"))
```

A számítógép eredményül $00FB-t ír ki.

5.3. NOT (NEGÁLÁS)

Az 1 az nem 0. Logikus, mondanánk, és a 0 sem 1! Hát ilyen egyszerű ez a művelet. Azért nézzük meg az igazságtáblát:

A	NOT A
0	1
1	0

Meg kell azonban jegyezni, hogy ez a BASIC NOT parancs nem így működik. A PRINT NOT 1 eredménye nem 0, hanem −1. A számítógép a parancsot mégis helyesen hajtja végre, csak a belső számábrázolás miatt írja ki ezt a látszólag helytelen értéket.

```
PRINT NOT 107
```

NOT

A	%0110 1011	107	$6B
eredmény	%1001 0100	148	$94

5.4. A KIZÁRÓ VAGY MŰVELET

Erre a műveletre ugyan nincs BASIC parancs, azonban a Plus/4-esnél van egy grafikai parancs, amely egy KIZÁRÓ VAGY műveletet hajt végre. Ez a GSHAPE parancs, amellyel a grafikáról szóló fejezetben foglalkozunk. Ezen kívül a 20. és 21. fejezetben is találkozunk ezzel a művelettel. A kizáró vagy, röviden EXOR művelet eredménye akkor 1, ha a két, összekapcsolandó kijelentés közül vagy csak az egyiknek, vagy csak a másiknak az értéke 1. Az eredmény azonban nem 1, ha mindkét kifejezés értéke 1, vagy ha mindkét kifejezés értéke 0. A művelet igazságtáblája:

A	**EXOR**	B	eredmény
0		0	0
0		1	1
1		0	1
1		1	0

Lássuk ezt a példánkon is:

EXOR

A	%0110 1011	107	$6B
B	%1101 0010	210	$D2
eredmény	%1011 1001	185	$B9

5.5. A LOGIKAI MŰVELETEK ALKALMAZÁSA

Egy alkalmazási területet már említettünk: a GSHAPE parancs logikai műveleteket végez. Ennek során a grafikus képernyő egy részét egy másik bitmintával kapcsolja össze.

Nézzünk egy újabb példát, a Plus/4-esben vannak vezérlőregiszterek, és a vezérlőregiszterekben mindegyik bitnek valamilyen speciális feladata van. Ilyen regiszterek vannak a TED-chipben is. Tekintsük most a 65286-os címen levő regisztert. Ennek a 3. bitje azt mutatja meg, hogy a képernyőn 24 vagy 25 sor jelenjen meg. Ha ennek a 3-as bitnek az értéke 0, akkor a kép 24, ha az értéke 1, akkor 25 sorból áll.

Alapesetben a képernyő 25 soros, tehát a 3-as bittel átkapcsolhatjuk 24 sorosra. Ehhez a 3-as bitet törölni kell. A 65286-os regiszterben (akárcsak más regiszterekben) még további 7 bit van, amelyeknek szintén megvan a speciális szerepük. Ezek értékét tehát nem szabad megváltoztatnunk. Ilyen esetekben használhatók a logikai műveletek:

Az ÉS művelet lehetővé teszi, hogy akárcsak egyetlen bitet is törölhessünk, azaz a bit értékét nullára állítsuk. A VAGY művelettel viszont akár egyetlen bitet is „magasra" állíthatunk, azaz a bit értékét 0-ról 1-re változtathatjuk.

Írjuk be a következőt:

```
POKE 65286 ,PEEK(65286)AND 247
```

A POKE paranccsal a tár rekeszeibe 0-255 közötti, tetszőleges értéket írhatunk.

A PEEK paranccsal egy rekesz tartalmát olvashatjuk el.

Először elolvassuk a 65286-os címen levő rekesz tartalmát, a tartalom és a 247 között elvégezzük az ÉS logikai műveletet, majd az így kapott értéket visszaírjuk a 65286-os rekeszbe.

Ezzel annak a bizonyos 3-as bitnek az értékét 0-ra állítottuk, és a képernyő tetejéről és az aljáról egy-egy fél sort „levágtunk". Vajon miért éppen a 247 kellett?

Van egy bináris számunk, amelynek az értéke %0001 1011

Az egyes biteket jobbról balra, 0-val kezdődően, az elfoglalt pozíciójuk száma szerint jelöljük. Ennek megfelelően a 3-as bit értéke ebben a bináris számban 1.

Végezzünk el most egy ÉS műveletet:

```
régi érték   %0001 1011
ÉS           %1111 0111
-----------------------
eredmény     %0001 0011
```

A bináris %1111 0111 szám decimális megfelelője a 247.

Most vissza akarjuk állítani a régi állapotot. A 3-as bitet tehát ismét 1-re kell állítani. Erre alkalmas a VAGY művelet:

```
régi érték   %0001 0011
VAGY         %0000 1000
-----------------------
eredmény     %0001 1011
```

A bináris %0000 1000 szám decimális megfelelője a 8. Írjuk be:

```
POKE 65286,PEEK(65286)OR 8
```

Láthatjuk, hogy amit akartunk, elértük: a képernyő ismét alapállapotba került.

A logikai műveletek egy további alkalmazási területe programok titkosítása. A programokat vagy programneveket ily módon annyira titkosítani lehet, hogy a kódok megfejtése még a szakembereknek is komoly munkát okozhat. Ebbe a témakörbe azonban nem akarunk részletesen belemenni. A BASIC nyelvű programozáshoz elegendő az, amit a logikai műveletekről ebben a fejezetben leírtunk.

5.6. PÉLDAPROGRAM: GYUFASZÁLAK

Ebben a játékban az ellenfél a számítógép. A képernyőn 10 gyufaszál látható. A feladat, hogy mi és a számítógép felváltva egy-egy gyufaszálat elvegyen vagy visszategyen. Az veszít, akinek az utolsó gyufaszálat kell elvennie.

Írjuk be a programot, majd tároljuk. Ezután elindíthatjuk a

```
RUN
```

paranccsal.

A számítógép felszólít bennünket, hogy egy gyufaszálat elvegyünk (a „ − " billentyű), vagy visszategyünk (a „ + " billentyű). Maga a játék tulajdonképpen nagyon egyszerű, és hamar kiismerhető, hogy mitől függ, hogy nyerünk vagy vesztünk. Alakítsuk át a játékot úgy, hogy 1, 2 vagy 3 gyufaszálat is el lehessen venni, ill. vissza lehessen tenni.

```
10 REM GYUFASZALAK
20 COLOR0,1:COLOR1,2:COLOR2,3:COLOR3,8:GRAPHIC3,1
30 S=10
40 CHAR1,0,24,"NALAM 0 GYUFASZAL, NALAD 0 GYUFASZAL VAN"
50 GOSUB 500
60 CHAR,0,1,"EGY PILLANAT ! LASSUK CSAK,"
70 CHAR,0,2,"KI IS KEZD ?"
80 GOSUB490:GOSUB490
90 X=INT(RND(0)*2)
100 GOSUB480
110 IFX=0THENCHAR,0,1,"EN KEZDEK !":GOTO130
120 CHAR,0,1,"TE KEZDESZ !":GOTO250
130 GOSUB490
140 GOSUB480
150 X=INT(RND(0)*2):Y=INT(RND(0)*6):IFY=1THEN260
160 IFS=1ORS=3THEN260
170 IFS=4ORS=2THEN210
180 IFX=1THENIFH0=0THENGOTO210:ELSEGOTO260
190 IFX=1ANDS/2<>INT(S/2)THENGOTO260:ELSE210
200 GOTO260
210 GOSUB480
220 CHAR,0,1,"ELVESZEK EGY SZALAT"
230 H0=H0+1:S=S-1:GOSUB510
240 IFS=0THEN470
250 GOTO310
260 GOSUB490
270 GOSUB480
```

```
280 IFH0=0THEN220
290 CHAR,0,1,"VISSZATESZEK EGY SZALAT"
300 H0=H0-1:S=S+1:GOSUB510
310 GOSUB490
320 GOSUB480
330 CHAR,0,1,"MOST TE KOVETKEZEL  (-/+)"
340 POKE239,0:GETKEYA$:IFA$="-"ORA$="+"THEN350:ELSE340
350 GOSUB480
360 IFA$="+"THEN400
370 IFS=1THEN430
380 CHAR,0,1,"TEHAT ELVESZEL EGYET"
390 H1=H1+1:S=S-1:GOSUB510:GOTO130
400 IFH1=0THENCHAR,0,1,"EZT NEM LEHET!!":GOTO310
410 CHAR,0,1,"EGY SZALAT VISSZA KELL TENNED!"
420 H1=H1-1:S=S+1:GOSUB510:GOTO130
430 CHAR,0,1,"NEM VOLT SZERENCSED, MOST VESZTETTEL!"
440 CHAR,0,3,"MEG EGY JATEK (I/N) ?"
450 POKE239,0:GETKEYA$:IFA$="I"THENRUN
460 GRAPHIC0,1:END
470 GOSUB480:CHAR,0,1,"GRATULALOK, TE NYERTEL!":GOTO440
480 FORI=0TO39:CHAR,I,1," ":CHAR,I,2," ":NEXT:RETURN
490 FORI=1TO1200:NEXT:RETURN
500 FORI=1TOS:GOTO550
510 IFS<2THENSA=1:ELSESA=S-1
520 CHAR,5,24," ":CHAR,24,24," "
530 CHAR,5,24,STR$(H0):CHAR,24,24,STR$(H1)
540 FORI=SATOS
550 CIRCLE2,I*10+20,50,3,6
560 PAINT2,I*10+20,50
570 FORJ=I*10+20-1TOI*10+20+1
580 DRAW3,J,55TOJ,155
590 NEXTJ:NEXTI
600 PAINT0,(S+1)*10+20,50
610 CIRCLE0,(S+1)*10+20,50,3,6
620 FORJ=SATOS+1
630 DRAW0,J,55TOJ,155
640 NEXTJ:RETURN
```

6. Összehasonlító utasítások

6.1. AZ IF...THEN UTASÍTÁS

Már az utasítás írásmódjából – IF...THEN (HA...AKKOR) látszik, hogy ez az utasítás valamiféle összehasonlítást végez: ha egy feltétel teljesül, akkor a számítógép valami mást csinál, mint akkor, ha a feltétel nem teljesül. Lássunk erre egy rövid példát:

```
10 A=INT(RND(1)*100+1)
20 INPUT"IRJON BE EGY SZAMOT 1 ES 100 KOZOTT";X
30 IF X = A THEN PRINT"HELYES!":GOTO 10
40 IF X > A THEN PRINT"AZ EN SZAMOM KISEBB":GOTO 20
50 PRINT"AZ EN SZAMOM NAGYOBB"
60 GOTO20
```

A 10-es sorban a program előállít egy véletlen számot 1 és 100 között. Az ide vonatkozó RND paranccsal a 4.4. fejezetben foglalkoztunk.

A 20-as sor egy adat beírására vár.

A 30-as és 40-es sorban a program összehasonlítja a beírt számot a véletlen számmal. Ha a két szám megegyezik, akkor a számítógép kiírja a „HELYES" szöveget, és visszaugrik a 10-es sorra.

A 40-es sorban a program megállapítja, hogy az általunk beírt szám nagyobb, mint a véletlen szám.

Az 50-es sorban nincs IF...THEN parancs. Ezt itt megtakaríthatjuk, hiszen a 30-as és a 40-es sorokban elvégzett összehasonlítás után már biztos, hogy a véletlen szám a nagyobb.

A következő összehasonlítások végezhetők el:

=	egyenlő
>	nagyobb
<	kisebb
<=	kisebb, egyenlő
>=	nagyobb, egyenlő
<> vagy ><	nem egyenlő
AND	ÉS művelet
OR	VAGY művelet
NOT	nem

Példák az összehasonlításokra:

```
10 IF A = B AND B > C THEN...
```

A feltétel akkor teljesül, ha A ugyanakkora mint B, ÉS ha B nagyobb mint C.

```
10 IF A OR B THEN...
```

A feltétel akkor teljesül, ha mindkét érték 0. Hasonlítsuk össze ezt az állítást az OR műveletre felírt igazságtáblával.

```
10 IF A AND B THEN...
```

Ha a logikai AND művelet eredménye nagyobb mint 0, akkor a feltétel teljesül.

```
10 IF NOT A AND B THEN...
10 IF A < C OR B = C THEN...
```

A feltétel akkor teljesül, ha A kisebb mint C, VAGY ha B egyenlő C-vel.

```
10 IF A THEN...
```

Ennél az „összehasonlítás"-nál a program csak azt állapítja meg, hogy az A értéke 0, vagy nem 0. Ha nem 0, akkor a feltétel teljesül.

Az összehasonlító műveletek eredményét a PRINT parancs segítségével közvetlenül kiírathatjuk a képernyőre. Nézzünk ezekre is példákat:

```
PRINT A = B
```

Ha az A és a B értéke azonos, akkor a képernyőn −1 jelenik meg. Ha az A és a B nem azonos, akkor 0-t ír ki.

```
PRINT A > B
```

Ha A nagyobb mint B, akkor a kiírás −1, egyébként 0.

```
PRINT A < B
```

Ha A kisebb mint B, akkor a kiírás −1, egyébként 0.

```
PRINT A <= B
```

Ha A kisebb, egyenlő B, a kiírás −1, egyébként 0.

```
PRINT A >= B
```

Ha A nagyobb, egyenlő B, a kiírás −1 egyébként 0.

```
PRINT A <> B
```

vagy

```
PRINT A >< B
```

Ha A kisebb vagy nagyobb, mint B, akkor a kiírás −1, egyébként 0, próbáljuk ki:

```
PRINT A = B <> C
PRINT A > B > C
PRINT 9 * 5 > 40
```

Az összes eddig bemutatott összehasonlítást karakterláncokkal (string), ill. karakter típusú változókkal is el lehet végezni. Ilyenkor a művelet az ASCII-kódtáblázatban levő, megfelelő értékeket hasonlítja össze.

```
PRINT "ALMA" > "AUTO";
PRINT "1" > "9"
```

Ezt a két műveletet csak számokkal, ill. numerikus vagy egész számú változókkal lehet elvégezni:

PRINT A AND B

Ezt a példát már a logikai műveleteknél is láttuk. A parancs ÉS kapcsolatot hoz létre az A és a B között, és az eredményt decimális alakban kiírja.

PRINT A OR B

A parancs VAGY kapcsolatot hoz létre az A és a B között, és az eredményt decimális alakban kiírja.

6.2. IF...THEN...ELSE

Magyarul: HA...AKKOR...EGYÉBKÉNT. Ha a feltétel teljesül, akkor a számítógépnek valamit el kell végeznie. Ha nem teljesül, akkor azt kell végrehajtania, ami az ELSE utasítás után következik. Gondoljunk csak a normál IF...THEN parancsra. Ha a feltétel nem teljesül, akkor a program végrehajtása az IF utáni sorban folytatódik.

Nézzük meg még egyszer az előző, számkitalálós programpéldánkat:

```
10 A=INT(RND(1)*10)+1
20 INPUT"IRJON BE EGY SZAMOT 1 ES 100 KOZOTT";X
30 IFX=ATHENPRINT"HELYES!":GOTO10
40 IFX>ATHENPRINT"AZ EN SZAMOM KISEBB"
ELSEPRINT"AZ EN SZAMOM NAGYOBB"
50 GOTO20
```

Megjegyzés: A 40-es sor már elég hosszúra sikerült. Egy programsorba összesen 89 karakter írható (a sort lezáró RETURN-t is beszámítva). Ennél hosszabb sort a program nem fogad el. Viszont majdnem valamennyi BASIC parancs rövidítve is írható. Erre itt a legjobb példa a PRINT parancs. Ennek a rövidítése a kérdőjel. Ha tehát a 40-es sorban a PRINT helyett ?-et írunk, akkor ennél hosszabb sor is beírható a programba. A rövidítésekkel a későbbiekben foglalkozunk.

A példaprogramból látjuk, hogy egy programsort megtakarítottunk. Ne feledjük azonban, hogy az IF...THEN...ELSE utasításssornak mindig egyetlen programsorban kell lennie. Az ELSE nem állhat külön sorban.

Az összehasonlítási parancsokban eddig csak számokat ill. numerikus változókat használtunk. Karakterláncokat is össze lehet azonban egymással hasonlítani. Itt a megszorítás mindössze annyi, hogy tisztán logikai kapcsolatok nem hozhatók létre. Természetesen egy összehasonlítás bővítéseként az AND és az OR használható.

Egy példa:

```
10 INPUT "BEFEJEZI A PROGRAMOT (I/N)";A$
20 IF A$="I" THEN END:ELSE 10
```

Befejezésül utalunk a Programciklusok című fejezetre, ahol ugyancsak foglalkozunk összehasonlításokkal.

6.3. A WAIT UTASÍTÁS

A WAIT utasítás segítségével a tár tetszőleges rekeszének tartalma és egy előre megadott érték között logikai művelet végezhető. Ha a művelet eredménye 0, akkor a parancs ismét megvizsgálja a rekesz tartalmát. Eközben a számítógép várakozik (ezért az angol WAIT = várj), azaz a program futását mindaddig felfüggeszti, míg az eredmény 0-tól eltérő.

Vigyázat: Ezzel az utasítással nagyon óvatosan kell bánni. Valamilyen módon biztosítani kell azt, hogy az illető tárrekesszel végzendő művelet eredménye valamikor 0 legyen. Ellenkező esetben a program a WAIT utasításnál állva maradna. A programot ebből az állapotából már csak a számítógép kikapcsolásával vagy a RESET gomb megnyomásával lehetne kimozdítani.

Az utasítás szintaxisa:

WAIT a,x [,y]

ahol

a = a tárbeli rekesz címe (0-65535)
x = az az érték, amellyel a parancs ÉS műveletet végez
y = az az érték, amellyel a parancs KIZÁRÓ VAGY műveletet végez

Egy példa: (ezt ne írjuk be!)

WAIT 192,1,15

A parancs először a 192-es rekesz és a 15 között elvégzi a KIZÁRÓ VAGY műveletet, majd az így kapott eredmény és az 1 között az ÉS műveletet. Az EXOR és az AND műveletek mindaddig ismétlődnek, míg az eredmény 0-tól eltérő.

Vigyázat: az utasítás kiadása előtt pontosan tudnunk kell, hogy milyen eredményt kapunk!

Ha a második értéket nem adjuk meg (tehát csak: WAIT 192,1), akkor a számítógép azt 0-nak tekinti, amellyel az EXOR művelet elvégzése a kiinduló értéket nem változtatja meg. Ilyenkor a parancs a 192-es rekesz tartalma és az 1 között elvégzi az ÉS műveletet. A WAIT utasítás is összehasonlító utasítás azzal a korlátozással, hogy – az IF...THEN-től eltérően – nem ad választási lehetőséget. Egyszerűen mindaddig várakozik, amíg az a bizonyos érték el nem éri a programban beállított értéket.

A WAIT nagyon jól használható a billentyűzet vagy a botkormány lekérdezésére. Arra is alkalmas, hogy bizonyos be- és kiviteli műveletek befejeződését megvárjuk (kazettás vagy lemezes egység). A használatához azonban egészen pontosan tudnunk kell, hogy mely rekeszek, mikor és milyen értékeket vesznek fel.

Nézzünk egy példát, ami programokban is jól használható:

```
WAIT 239,8
```

A 239-es tárrekeszben az operációs rendszer a lenyomott billentyűk számát tárolja. A WAIT utasítással most addig várunk, míg a billentyűzeten nyolc gombot le nem nyomunk. Ki is próbálhatjuk! Ha lenyomtunk nyolc billentyűt, akkor a WAIT utasításban megadott feltétel teljesül, és a nyolc billentyűhöz tartozó karaktereket kiírathatjuk a képernyőre.

A példából látjuk, hogy nem adtunk meg második értéket. Ez azt jelenti, hogy a parancs a 239-es rekesz és a 0 érték között végzi el a KIZÁRÓ VAGY műveletet. Ennek az eredménye megegyezik a rekesz tartalmával. Most ezen érték (tehát az eredeti érték) és a 8 között az ÉS műveletre kerül sor. Ha még csak 7 gombot nyomtunk meg, akkor a parancs a 7 ÉS 8 műveletet végzi el, aminek az eredménye 0. A WAIT utasítás mindaddig vár, amíg egy nyolcadik billentyűt is le nem nyomtunk.

A WAIT utasítással a Plus/4 belső óráját is lekérdezhetjük, ill. várakozhatunk, míg az óra egy bizonyos értéket el nem ér. A belső óra aktuális értékét a 163-165-ös rekeszek tartalmazzák. A

```
WAIT 163,1
```

addig várakozik, míg a 163-as tartalma ÉS az 1 közötti művelet eredménye 0-tól különböző. Másként fogalmazva ez azt jelenti, hogy ha a 163-as rekeszben levő érték nem egyenlő 1-gyel, akkor a program várakozik.

6.4. PÉLDAPROGRAM: REAKCIÓTESZT

A következő programmal a reakcióidőnket mérhetjük. A program indításakor a képernyő színe zöldre vált. Meghatározatlan idő után a képernyő piros lesz. Ekkor le kell nyomni a RETURN billentyűt. A számítógép most kiírja a reakcióidőt.

```
10 REM REAKCIOIDO TESZT
20 SCNCLR:COLOR0,1:COLOR1,2
30 PRINTSPC(13)"REAKCIOTESZT"
40 PRINT:PRINT"INDITAS A  G  BILLENTYUVEL"
50 PRINT:PRINT"AMIKOR A KEPERNYO PIROS LESZ,"
60 PRINT:PRINT"NYOMJA LE A RETURN BILLENTYUT"
70 GETA$:IFA$<>"G"THEN70
80 SCNCLR
90 COLOR0,6
100 X=INT(RND(0)*5000+1000)
110 FORI=1TOX:NEXT
120 COLOR0,3:TI$="000000"
130 INPUTA
140 T=TI
150 COLOR0,1
160 PRINT T/60;"MASODPERCRE VOLT SZUKSEGE"
170 FORI=1TO1000:NEXT
180 PRINT:PRINT"MEG EGYSZER (I/N) ?"
190 POKE239,0:GETKEYA$:IFA$="I"THEN20
200 END
```

7. A Plus/4-es billentyűzete

Mielőtt ezt a könyvet az Olvasó megvásárolta, már bizonyára némi tapasztalatra tett szert a gép billentyűzetét illetően. Ezért ebben a fejezetben feltételezzük a kurzormozgató és más billentyűk alapvető funkcióinak ismeretét. A billentyűk elrendezése az amerikai szabványnak felel meg, így az Y és a Z billentyű az európaihoz képest fel van cserélve. A billentyűzeten nincsenek továbbá ékezetes karakterek sem. (Magyarországon ékezetes változat került forgalomba, amely a második betűkészletben a C= jellel érhető el. Ebben a fejezetben közelebbről megismerkedünk a billentyűzettel, és sajátosságaikkal. Az egyes billentyűk funkcióinak pontos ismerete sokkal kényelmesebbé teszi mind a gép, mind a programok kezelését.

7.1. A KÖZVETLEN ÜZEMMÓD

Amikor a Plus/4-est bekapcsoljuk, a gép az úgynevezett közvetlen (vagy parancs) üzemmódban van. Ez azt jelenti, hogy minden tevékenységet, amire a gépet most utasítjuk, a gép azonnal végrehajt.

Egy példa:

```
PRINT "EZ A KOZVETLEN MOD"
```

Ha ezt beírtuk, és utána megnyomjuk a RETURN gombot, akkor a képernyőn ez jelenik meg:

```
EZ A KOZVETLEN MOD
```

Amint látjuk, a számítógép a parancsot, nevezetesen azt, hogy írja ki az "EZ A KOZVETLEN MOD" szöveget, azonnal végrehajtotta. A beírt parancsot azonban nem tárolja. A számítási műveletek is végrehajthatók közvetlen módban. Írjuk be a következőt, és nyomjuk meg a RETURN billentyűt:

```
PRINT 5 * 3
```

A képernyőn megjelenik az eredmény: 15.

A BASIC parancsok többsége használható közvetlen módban. Vannak azonban olyanok is, amelyeket csak programon belül lehet használni. Közvetlen módban a számítógép valamennyi billentyűje aktív állapotban van, ami azt jelenti, hogy a gép minden egyes billentyű, vagy billentyűkombináció lenyomásakor az annak megfelelő műveletet azonnal végrehajtja.

7.2. A RETURN BILLENTYŰ

Ezzel a gép jobb oldalán található billentyűvel kapcsolatban már a korábbiakban szereztünk tapasztalatokat. Ez a számítógép egyik legfontosabb billentyűje. E billentyű lenyomásával közöljük a számítógéppel, hogy egy adat beírásával végeztünk. A számítógép ekkor megállapítja, hogy melyik sorban található a kurzor, és megkísérli a sorban levő szöveg értelmezését, hogy a benne levő parancsokat, utasításokat végrehajtsa. Ha a sorban valami olyat talált, amit nem ért meg, ezt a körülményt hibaüzenet formájában kiírja a képernyőre.

Mindegyik parancs- vagy utasítássor két képernyősor hosszúságú +9 betű lehet. Ha a sor ennél hosszabb, akkor a számítógép szintén hibát jelez. Ha a kurzor előtt a sorban egyetlen karakter sincs, akkor RETURN billentyű lenyomása soremelést jelent. Nyomjuk meg néhányszor a szóköz billentyűt, majd a RETURN-t. A kurzor most a következő képernyősor első oszlopába ugrik. Ismételjük ezt meg úgy, hogy ezúttal a SHIFT/szóköz billentyűt nyomjuk le. A számítógép most a READY üzenettel jelentkezik vissza, a kurzor pedig a beviteli sor alatti harmadik sorba ugrik. Amint látjuk, a második esetben a számítógép megkísérelte a „beírt" karaktereket értelmezni.

A kurzort a SHIFT/RETURN billentyűkombinációval is a következő sorba vihetjük. Ilyenkor azonban a gép a sorban levő utasításokat nem hajtja végre, és ha ez egy programsor volt, akkor nem kerül be a programba.

Jegyezzük meg:

Minden parancs vagy utasítás beírását a RETURN billentyű lenyomásával kell befejezni!

Látszólag a SHIFT/RETURN-nek is azonos a hatása, de a gép ekkor a sorban található utasításokat nem hajtja végre, és a sor nem kerül be a programba.

7.3. A CTRL BILLENTYŰ

Ezzel a billentyűvel a billentyűzet speciális funkciói hajthatók végre. A CTRL billentyűt mindig valamelyik más billentyűvel együtt kell használni. Ha csak a CTRL billentyűt nyomjuk le, akkor ennek nincs semmiféle hatása. A billentyűzet felső sorában találhatók a számjegybillentyűk. A billentyűk elülső oldalán is vannak feliratok. Az első nyolc billentyűvel a színeket állíthatjuk. Így pl. a CTRL és a 3-as billentyű egyidejű lenyomására az előbbi fekete kurzor piros színű lesz. Az ezután beírásra kerülő összes karakter ilyen színű lesz. A színeket tetszés szerinti gyakorisággal váltogathatjuk; a számítógép egy utasítássornak csak a tartalmát értelmezi, a színét nem. Azt, hogy hogyan lehet a háttér és a keret színét változtatni, majd a grafikáról szóló fejezetben, a COLOR parancs ismertetésekor vizsgáljuk meg.

A CTRL és a 9-es billentyű egyidejű lenyomására a számítógép az úgynevezett inverz üzemmódba kerül. Ilyenkor a beírt karakterek színe a háttér színe lesz, míg a háttér színe a karakter színére változik. Ezt az üzemmódot a CTRL/0, ESC/0 vagy a RETURN billentyű lenyomásával lehet kikapcsolni.

Tekintsük át a CTRL billentyű egyes funkcióit:

CTRL	Az elért hatás
1	Fekete
2	Fehér
3	Piros
4	Türkiz
5	Lila
6	Zöld
7	Kék
8	Sárga
9	Inverz mód bekapcsolva, RVS ON
0	Inverz mód kikapcsolva, RVS OFF
S	Megállítja egy program futását vagy egy program LIST-tel való kilistázását üzenet nélkül. Folytatás valamelyik karaktert író billentyűvel.
FLASH ON	Az utoljára beírt karakter villog
FLASH OFF	Kikapcsolja a villogást

Néhány további, eddig esetleg nem ismert kombináció:

CTRL	Az elért hatás
H	Kikapcsolja a SHIFT/C= billentyűkombinációt
I	Bekapcsolja a SHIFT/C= billentyűkombinációt
M	Mint a RETURN billentyű
Q	Mint a kurzor-fel billentyű
;	Mint a kurzor-jobbra billentyű
N	Bekapcsolja a nagybetű/kisbetű billentyűzetet
:	Az ESC billentyű lenyomását szimulálja
T	Mint a DEL billentyű (karaktert töröl)
E	A karakter színe fehér
£ (font)	A karakter színe piros

A CTRL/H és a CTRL/I kivételével a többi kombináció hatása más billentyűkkel egyszerűbben és logikusabban érhető el; ezeket itt csak a teljesség kedvéért tüntettük fel.

A Plus/4-es 16 alapszínt képes előállítni. Nézzük meg, hogy hogyan lehet a további nyolc színt megjeleníteni.

7.4. A C= (COMMODORE) BILLENTYŰ

Ennek a billentyűnek nem véletlenül ez a neve: ilyen billentyű csak a Commodore gépeken van. A billentyű a bal alsó sarokban, a SHIFT gomb mellett található. E billentyű segítségével a karakterbillentyűk előoldalán a balra látható grafikus karakterek írhatók ki. Ebben az esetben annak nincs jelentősége, hogy a nagybetű/grafikus vagy a nagybetű/kisbetű üzemmódban vagyunk. Magyar gépeken a nagybe-

tű/kisbetű üzemmódban ékezetes betűket ír. E két üzemmód között egyébként a SHIFT és a COMMODORE billentyűk egyidejű lenyomásával lehet átkapcsolni. Lássuk akkor a kombinációkat:

C= billentyű	Az elért hatás
1	Narancs
2	Barna
3	Sárgászöld
4	Rózsaszín
5	Kékeszöld
6	Világoskék
7	Sötétkék
8	Világoszöld
SHIFT 1-szer	Nagybetű/kisbetű mód bekapcsolása
SHIFT 2-szer	Nagybetű/grafikus mód bekapcsolása

Ennek a billentyűnek még további funkciói is vannak. Ha a Plus/4-eshez egy kazettás-egységet csatlakoztatunk, és egy programot szeretnénk a gépbe tölteni, akkor a gép a LOAD parancs hatására elkezd a kezettán egy programot keresni. Amikor megtalálja az első betölthető, vagy a nevével behívott programot, akkor kiírja a képernyőre a FOUND programnév üzenetet. Ha most megnyomjuk a COMMODORE billentyűt, akkor a gép elkezdi a program betöltését.

Például egy program listázásakor a C= billentyű lenyomásakor a listázás sebessége lényegesen csökken. Egy program végrehajtási sebessége is csökkenthető ezzel a billentyűvel.

Az összes, itt nem említett kombinációnál a C= billentyű úgy viselkedik, mint a SHIFT billentyű.

7.5. A SHIFT BILLENTYŰK

Két azonos értékű ilyen billentyű van. Ezekkel a billentyűkkel azokat a jeleket írhatjuk ki, amelyek a billentyűk elülső oldalán, jobbra láthatók. Ha a nagybetű/kisbetű üzemmód van bekapcsolva, akkor ezeknek a billentyűknek ugyanaz a szerepe, mint az írógépeken a váltóbillentyűnek, azaz nagybetűre kapcsol át.

A billentyűzeten van még egy SHIFT LOCK feliratú billentyű is. Ha ezt lenyomjuk, akkor beugrik az alsó helyzetébe, és elengedve ott is marad. A hatása az, mintha a SHIFT billentyűt állandóan lenyomva tartanánk. Újabb lenyomásra a SHIFT LOCK billentyű visszaugrik az eredeti helyzetébe.

Ha egyidejűleg lenyomjuk a SHIFT és a RUN/STOP billentyűket – és ha egy lemezmeghajtó egység boldog tulajdonosai vagyunk –, akkor a meghajtóban levő lemezről az első program betöltődik a gépbe, és automatikusan elindul, de csak akkor, ha a bekapcsolást követően még nem töltöttünk be programot. Ha igen, akkor az utoljára betöltött program még egyszer betöltődik. Erre vonatkozó részleteket a lemezmeghajtó egységet ismertető fejezet tartalmaz.

7.6. A PLUS/4-ES EGYÉB BILLENTYŰI

Lássunk ezekről a billentyűkről egy áttekintést:

A táblázatban az Sh betű a SHIFT billentyűket jelöli.

Billentyű	A billentyű hatása
INST/DEL	A kurzortól balra álló karakter törlődik
ShINST/DEL	A kurzor pozíciójára egy üres hely kerül, ahová újabb karakter írható be, ilyenkor nem a megszokott módon működik néhány billentyű, pl. DEL, RUN/STOP
CLEAR/HOME	A kurzort a képernyő első oszlopának első sorába viszi (bal felső sarok). A képernyő tartalma nem változik.
ShCLEAR/HOME	A képernyő törlődik, a kurzor az első sor első oszlopába ugrik.
RUN/STOP	Megszakítja a programokat vagy a listázást.
ShRUN/STOP	Betölti és elindítja azt a programot, amelyre a lemezmeghajtó éppen pozicionálva van. Lásd még DLOAD.

Bizonyára feltűnt, hogy ha egy billentyűt hosszabb ideig lenyomva tartunk, akkor a karakter kiírása ismétlődik. Akinek ez esetleg nem tetszene, ezt az ismétlődést könnyedén kikapcsolhatja.

POKE 1344,0	Csak a kurzor, a DEL és a szóköz billentyűk ismételnek.
POKE 1344,64	Egyik billentyű sem ismétel.
POKE 1344,128	Valamennyi billentyű ismétel.

7.7. A FUNKCIÓBILLENTYŰK

A Plus/4-es egyik különlegessége, hogy az F1-F8 billentyűk szabadon programozhatók. Lehetőség szerint használjuk is ki ezt az előnyt, mert ezzel jelentős beírási munkát takaríthatunk meg. A számítógép bekapcsolásakor ill. egy RESET után az operációs rendszer az F-billentyűkhöz BASIC parancsokat rendel. Ezt a „szereposztást" azonban bármikor, tehát akár közvetlen, akár program módban is megváltoztathatjuk.

A funkcióbillentyűk programozásának **szintaxisa**:

KEY n,"xyz"

az n helyére annak a billentyűnek a számát kell írnunk, amelyiket programozni akarjuk.

Az xyz helyére azokat a betűket, számokat vagy parancsokat írjuk, amelyeknek a billentyű lenyomására a képernyőn meg kell jelenniük. Írjuk be most a számítógépbe azt, hogy KEY, majd nyomjuk meg a RETURN billentyűt. A képernyőn megjelenik a funkcióbillentyűk „tartalma".

7.7.1. A funkcióbillentyűk programozása

Nézzük meg, hogyan változtathatjuk meg az F1 billentyű tartalmát úgy, hogy a billentyűhöz a PRINT parancs tartozzon. Írjuk be:

```
KEY1,"PRINT"
```

Ha most – természetesen a RETURN lenyomása után – lenyomjuk az F1 gombot, akkor a képernyőn a PRINT szöveg jelenik meg.

Ha egy program hibaüzenettel szakad meg, akkor a hibakereséshez gyakran szükség van a legfontosabb változók nevére és tartalmára. Nevezzük ezeket a változókat A, B, C változóknak, és rendeljük ezeket az F2 billentyűhöz:

```
KEY2,"PRINTA,B,C" + CHR$(13)
```

Az F2 billentyűt lenyomva a gép a benne tárolt parancsot azonnal végrehajtja. Az azonnali végrehajtást a CHR$(13) beírása váltja ki: a 13 ugyanis a „kocsi vissza" (elterjedt angol nevén Carriage Return) ASCII-kódja, és ennek a parancsba való beírásával a RETURN billentyű lenyomását imitáltuk. Az egyes ASCII-kódok jelentését a Függelék tartalmazza. Így például a PRINT CHR$(68) parancs egy D betűt ír a képernyőre. Egy kicsit nehezebb az, ha pl. az F3 billentyűvel a nevünket akarnánk a képernyőre íratni. Írjuk be mondjuk ezt:

```
KEY3,"PRINT" + CHR$(34) + "KATINKA" + CHR$(13)
```

Az F3 lenyomására most a képernyőn a KATINKA szöveg jelenik meg. A PRINT parancsszó utáni idézőjelet külön kellett előállítanunk. Az idézőjel ASCII-kódja 34, az idézőjel kiírását tehát CHR$(34) végzi.

Az F billentyűkbe akár még rövidebb, önmagukban zárt rutinok is elhelyezhetők:

```
KEY4,"FOR I=0 TO 100:PRINT I;:NEXT" + CHR$(13)
```

Ezt azonban még jobban is csinálhatjuk. Mint bizonyára tudjuk, a Plus/4-es BASIC nyelvének legtöbb parancsát és műveletét rövidítve is lehet írni. Így pl. a PRINT parancs rövidítése a kérdőjel. A funkcióbillentyűk programozásánál ezeket a rövidítéseket használva tárhelyet takaríthatunk meg, ami néha nagyon fontos lehet, mivel a 8 billentyű összesen maximum 128 karaktert tartalmazhat.

A funkcióbillentyűk tartalma a tárban a $055F – $05E6 közötti címeken található. Ezeket a MONITOR-programmal nézhetjük meg.

Az eddigi kísérletezgetéseinkkel a funkcióbillentyűk tartalmát bizonyára sikerült alaposan megkeverni. Az eredeti állapot visszaállítására a gép kezelési útmutatója két lehetőséget említ. Az egyik a gép kikapcsolása, a másik pedig a RESET billentyű megnyomása. Mi ezeket nem ajánljuk, mert mindkettő törli az esetlegesen a gépben levő programot. A mi javaslatunk:

```
SYS 62359
```

Ha beírjuk ezt a parancsot, majd megnyomjuk a RETURN billentyűt, akkor a funkcióbillentyűk ismét az eredeti tartalmukat veszik fel, és más változás nem lesz.

Egy kivétel azonban van, mégpedig, hogy az F1 billentyű most nem a szövegfeldolgozó program bekapcsolását végző SYS parancsot, hanem a GRAPHIC parancsot tartalmazza. Ennek a következő az oka: a gép a bekapcsolásakor, ill. a RESET lenyomásakor – többek között – az itt beírt rutint is meghívja, és az F1 billentyűhöz

a GRAPHIC parancsot rendeli. Ezt követően azonban még egy úgynevezett modulreset-et is végrehajt. Ez a modulreset a beépített szoftver ROM-ból meghív egy rutint, amely az F1 billentyűt átprogramozza, úgy hogy az ezután már a SYS1525:3-PLUS-1" parancssort tartalmazza.

Itt jegyezzük meg, hogy számunkra ésszerűbbnek tűnik, ha az F1 billentyű a GRAPHIC parancsot tartalmazza. A másik esetben ugyanis könnyen előfordulhat, hogy a beépített szoftver véletlen elindításával a tárban levő program visszahozhatatlanul törlődik!

A fejezet lezárásaként következzen egy olyan program, amellyel könnyen és gyorsan programozhatjuk a funkcióbillentyűket. Először sorról sorra ismertetjük a programot, hogy azt mindenki a saját igényei szerint alakíthassa.

10-es sor:	Megjegyzés sor, beíráskor elhagyható.
20-as sor:	Törli a képernyőt és a változókat, kurzor a bal felső sarokba ugrik.
30-as sor:	Kiíratja az F billentyűk pillanatnyi tartalmát.
40-es sor:	Megkérdezi az átprogramozandó billentyű számát.
50-es sor:	Egy billentyű lenyomására vár.
60-as sor:	Ha a lenyomott billentyű egy betű, vagy egy 8-nál nagyobb vagy 1-nél kisebb szám, akkor ugrás vissza a 40-es sorba.
70-es sor:	A beírt számot a gép szöveges változóként tárolta; ezt most numerikus változóvá alakítjuk.
80-as sor:	Megkérdezzük, hogy mit tartalmazzon a kiválasztott billentyű.
90-es sor:	A POKE parancs hatására a számítógép idézőjel üzemmódba kerül. A Z változó az A$ aktuális hosszát tartalmazza, majd a program egy billentyű lenyomására vár.
100-as sor:	Ha lenyomtuk a DEL billentyűt, és az A$-ban még nem volt karakter, akkor a program a DELete-t figyelmen kívül hagyja, és visszaugrik a 90-es sorba.
110-es sor:	Az ESC kivételével minden lenyomott karakter kiíródik a képernyőre. Az ESC billentyű lenyomását azonban „beírjuk" az F$-ba.
120-as sor:	Ha a lenyomott billentyű a RETURN volt, akkor az adatbevitelt befejezettnek tekintjük, és elágazunk a 160-as sorra.
130-as sor:	Az A$-ba beírjuk a „lenyomott billentyűt".
140-es sor:	Ha a DEL billentyűt nyomtuk le, akkor az A$-ban levő utolsó karaktert töröljük.
150-es sor:	Visszaugrás a 90-es sorra, hogy újabb karaktereket lehessen fogadni.
160-as sor:	Ha az A$-ban nincs karakter, akkor a kiválasztott billentyű tartalma nem változik, ugrás a 190-es sorra.
170-es sor:	A billentyűbe kerülő program RETURN-nel záródjon?
180-as sor:	Egy billentyű lenyomására vár.
190-es sor:	Ha a kérdésre „I"-vel válaszolunk, akkor a program az A$ végére még beír egy CHR$(13)-at, azaz a RETURN parancsot.
200-as sor:	A kiválasztott funkcióbillentyűbe a most meghatározott program kerül (ez utóbbit az A$ tartalmazza).
210-es sor:	Képernyő törlése, a billentyűkben levő új programok kiírása.
220-as sor:	Kell másik billentyű tartalmát is változtatni?
230-as sor:	Várakozás egy billentyű lenyomására. Ha a válasz „I", akkor ugrás a 20-as sorra, és kezdődhet az egész elölről.
240-es sor:	Képernyő törlése, program vége.

```
10 REM F-BILLENTYUK
20 PRINTCHR$(147);:CLR
30 KEY
40 PRINT:PRINT"MELYIK F TARTALMA VALTOZZON ? (1-8)"
50 GETKEY B$
60 IF ASC(B$)<49 OR ASC(B$)>56 THEN 40
70 B=VAL(B$)
80 PRINT:PRINT"MI LEGYEN AZ F"B" BILLENTYU TARTALMA ?":PRINT
90 POKE203,15:Z=LEN(A$):GETKEY F$
100 IF F$=CHR$(20) AND Z=0 THEN 90
110 IF F$<>CHR$(27) THEN PRINT F$;
120 IF F$=CHR$(13) THEN 160
130 A$=A$+F$
140 IF F$=CHR$(20) THEN A$=MID$(A$,1,Z-1)
150 GOTO 90
160 IF A$="" THEN PRINT :
PRINT"A BILLENTYU TARTALMA NEM VALTOZOTT":GOTO220
170 PRINT:PRINT"LEGYEN A VEGEN  RETURN  ? (I)"
180 GETKEY B$
190 IF B$="I" THEN A$=A$+CHR$(13)
200 KEY B,A$
210 PRINT CHR$(147);:KEY
220 PRINT:PRINT"MASIK F-BILLENTYU TARTALMA IS VALTOZZON ? (I)"
230 GETKEY B$:IFB$="I" THEN 20
240 PRINT CHR$(147);:END
```

Megjegyzések a program működéséhez:

A billentyűkön látható valamennyi betű, szám és jel beírható, tehát olyan jelek is, mint pont, pontosvessző, kettőspont és idézőjel. Ezt az első látásra talán egy kicsit érthetetlennek tűnő 90-140-es programsorok teszik lehetővé. Nincs szükség tehát CHR$-kódokra. A számítógép a karaktereket bekérő ciklus során állandóan az úgynevezett idézőjel módban van. Így lehetséges az is, hogy a kurzormozgató billentyűket közvetlenül, tehát PRINT nélkül beírhassuk a funkcióbillentyűkbe.

Idézőjeleket csak akkor kell beírni, ha ezt maga a parancs szintaxisa követeli meg, mint pl. a PRINT"PLUS/4". Az idézőjel használata a következő formában hibás lenne: "PRINT"PLUS/4".

Valamennyi ESCAPE művelet is beírható a billentyűkbe. Ezt ugyanúgy kell végezni, mint a közvetlen módban. Egy példa: ESC/A. Először tehát le kell nyomni az ESC billentyűt, és miközben ezt lenyomva tartjuk, az A billentyűt. Ugyanígy használhatók a CTRL és kombinációi.

Hosszú programok beírása előtt egyszerűen töltsük be ezt a programot és helyezzük a majd beírandó program végére úgy, hogy ezt a rövid segédprogramot pl. a RENUMBER60000 paranccsal átsorszámozzuk. Ezután a segédprogramunk a RUN60000 paranccsal bármikor hívható. Ha a programozási munkánkat félbeszakítjuk, akkor a megírt programmal együtt ezt a segédprogramot is tároljuk. Ha a főprogramunk már teljesen kész és működőképes, akkor a segédprogramunkat a DELETE60000- paranccsal egyszerűen kitöröljük.

7.8. AZ ESCAPE BILLENTYŰ

A billentyűzet bal felső sarkában található az ESC billentyű. Ez lehetővé teszi a programozó számára, hogy a számítógép olyan üzemmódjaihoz vagy műveleteihez férhessen hozzá, amelyeket más módon nem használhatna. Mivel a gép kézikönyve ennek a billentyűnek a szerepével csak nagyon röviden foglalkozik, és helyenként hibás is, ebben a fejezetben erre részletesebben kitérünk. Ha közvetlen módban lenyomjuk ezt a gombot, akkor egyelőre nem történik semmi. A gép egy másik billentyű lenyomására is vár. Mivel azonban a képernyőn sem az ESC, sem az ezt követő billentyű lenyomását nem jelzi ki, az egész könnyen áttekinthetetlenné válik. Ezért azt javasoljuk, hogy a továbbolvasás előtt írjuk be a gépbe az e fejezet végén található, az ESC billentyűkre vonatkozó gépi kódú programot. Ez a program megkönnyíti az ESC billentyűk hatásának tanulmányozását.

Az ESC billentyű és egy további, meghatározott betűt tartalmazó billentyű lenyomására a számítógép vagy egy üzemmódot kapcsol be/ki, vagy azonnal végrehajt egy műveletet. Ha az ESC billentyű lenyomása után egy olyan billentyűt nyomunk le, amelyek kombinációját a gép nem értelmezi, akkor a számítógép visszatér a billentyűzet „normál" lekérdezésére anélkül, hogy bármit is végrehajtana.

A műveletet itt nem matematikai értelemben használjuk, hanem azt értjük alatta, hogy a számítógép valamit csak egyszer hajt végre. Ezzel szemben ha a számítógép egy üzemmódot kapcsol be, akkor a billentyűk lenyomására mindaddig másképp reagál, míg az üzemmódot ki nem kapcsolja.

Nézzük meg az ESC-műveleteket, ill. üzemmódokat először a közvetlen módban.

Háromféle üzemmód létezik:

1-es üzemmód: **ESC A** betűk *automatikus beszúrása*

Tegyük fel, hogy programozás közben egy szöveget hibásan írtunk be. A helyes szöveg: "ETTEM KENYERET MEGGYEL", de a beíráskor kifelejtettük a "KENYERET" szót, és most szeretnénk beírni. Erre két lehetőségünk van:

Az egyik, hogy a kurzorral az ETTEM szó utáni helyre állunk, a SHIFT/DEL billentyű lenyomásával beszúrunk nyolc üres karaktert, és az így előállított üres helyre beírjuk az előbb kihagyott szót.

A második lehetőség a következő: nyomjuk le az ESC billentyűt, majd ezután az A billentyűt. Menjünk a kurzorral az ETTEM szó utáni helyre, és írjuk be a KENYERET szót. Az egyes billentyűk lenyomásakor a megfelelő karakterek rendre a helyükre kerülnek. Ha beszúrtuk a szót, nyomjuk le ismét az ESC, majd a C= billentyűt. Ezzel kikapcsoltuk a beszúrási üzemmódot.

Nem kell bizonygatni, hogy a második módszer kényelmesebb, hiszen ennél nem kell foglalkozni a beszúrandó karakterek leszámolásával. Az első megoldás akkor lehet a jobb, ha csak egyetlen karaktert kell beszúrni: ekkor egy billentyű lenyomását megtakaríthatjuk.

2-es üzemmód: **ESC M** a képernyő görgetésének leállítása

Ennek a billentyűkombinációnak a lenyomása után a következő történik: ha a kurzor pl. a képernyő közepén van, és beírjuk a PRINT A parancsot, akkor a RETURN lenyomása után a változó értéke a parancssor alatt jelenik meg. Ha viszont a kurzor az utolsó sorban állt, akkor a kiírás a képernyő első sorába kerül.

Ezt az üzemmódot az **ESC L** lenyomásával lehet kikapcsolni, ez egyben a képernyő görgetésének az indítását is jelenti. Ezt az üzemmódot elsősorban programon belül lehet ésszerűen használni. Ugyanez vonatkozik a harmadik, egyben utolsó üzemmódra is.

3-as üzemmód: **ESC R** a képernyő méretének csökkentése, a képernyő törlése

Az ESC és az R billentyűk lenyomására a képernyő törlődik, és a képernyő mérete a bal és a jobb oldalon egy-egy oszloppal, a felső és az alsó szélén pedig egy-egy sorral csökken.

Ezt az üzemmódot az **ESC N** vagy a HOME billentyű kétszeri megnyomása kapcsolja ki, és egyúttal a képernyőt is törli. Valószínűleg ez az üzemmód is csak programon belül használható értelmesen (vagy ha a tv-n nem látszik a kép szélein levő betű). Térjünk rá most a műveletekre. Ismétlésképpen: az üzemmódot és a műveletet aszerint különböztettük meg, hogy az üzemmódból való kilépéshez az üzemmódot ki kell kapcsolni, míg egy műveletet a gép azonnal, de csak egyszer hajt végre.

(Megjegyzés: a későbbiekben ismertetésre kerülő ESC B és ESC T felfogható az ESC R egy speciális esetének.)

1-es művelet: **ESC D**

Törli a képernyőnek azt a sorát, amelyben a kurzor éppen van. A törölt sor alatti képernyőrész egy sorral feljebb gördül, és a kurzor a sor elejére áll. Ha egy logikailag összetartozó karaktersor 40-nél több karaktert tartalmaz – azaz a sor átnyúlik a következő képernyősorba, akkor mindkét sor törlődik. Ez a művelet tehát nagyon jól használható programsorok szerkesztésére, ha a programsort még nem zártuk le egy RETURN-nel. Ha viszont lezártuk, akkor a programsor a billentyűkombináció hatására a képernyőről ugyan eltűnik, de a program továbbra is tartalmazza.

2-es művelet: **ESC I**

Egy sort szúr be a képernyőre: a kurzor sora, és valamennyi alatta levő sor egy sorral lejjebb gördül, és a kurzor az így létrejövő üres sor elejére áll. Az ESC I mind közvetlen módban, mind szerkesztésre használható.

3-as művelet: **ESC J**

A kurzor a sor elejére ugrik. Itt is érvényes, hogy a sor tartalmazhat 40-nél több karaktert is, tehát átnyúlhat a következő képernyősorba. Ez a művelet szintén programok javításánál használható jól, különösen akkor, ha a kurzor már a javítandó sor végén áll, a javítanivaló pedig a sor elején van. Az ESC J gyorsabban állítja a kurzort a sor elejére, mintha a kurzormozgató billentyűvel kellene visszalépkednünk.

4-es művelet: **ESC K**

Ugyanaz, mint az ESC J, azzal a különbséggel, hogy itt a kurzor annak a sornak a végére ugrik, amelyben állt.

5-ös művelet: **ESC P**

Törli a sort a kurzor helyéig. Ha pl. a kurzor a sor 26. oszlopában van, akkor ez a művelet a sorban a 0–26 közötti oszlopokban levő karaktereket törli, és helyettük 26 üres karaktert ír be. A kurzor pozíciója nem változik. Ez a művelet is felhasználható a javításhoz.

6-os művelet: **ESC Q**

Álljon a kurzor ismét a 26. oszlopban. E művelet hatására a sorban a kurzortól jobbra levő karakterek törlődnek. Szintén javításhoz használható.

7-es művelet: **ESC V**

A képernyő egy sorral felfelé gördül, és a képernyő alján egy üres sor jelenik meg. Ezt a gördítést egyébként csak akkor lehetne elérni, ha a kurzor a legalsó sorban állna (új sorba lépünk pl. ha megnyomnánk a KURZOR LE billentyűt). Ennél a műveletnél tehát mindegy, hogy a kurzor a képernyő melyik sorában áll.

8-as művelet: **ESC W**

A képernyő egy sorral lefelé gördül, és a képernyő tetején egy üres sor jelenik meg. Ez a művelet sem érhető el más módon.

9-es művelet: **ESC X**

Ha lenyomjuk az ESC és az X billentyűt, akkor a számítógép billentyűzete ismét a „normál" módon működik. Ez a billentyűkombináció nem kapcsol be ESC-üzemmódot, és nem hajt végre ESC-műveletet. Az X billentyű helyett bármely más, az ESC billentyűvel egyébként nem együtt használatos szám vagy betű billentyű lenyomása is ugyanezt a hatást fejti ki.

10-es művelet: **ESC O**

Tegyük fel, hogy közvetlen módban vagyunk, és a RVS ON és a FLASH ON billentyűkkel bekapcsoltuk az inverz és a villogtató üzemmódot. Ez a két üzemmód egyszerre kikapcsolható az ESC O billentyűkombinációval. Ezzel a művelettel az idézőjel üzemmód is kikapcsolható: tegyük fel, hogy a programozás közben idézőjel üzemmódba kerültünk – ezt az üzemmódot a nyitó idézőjel beírása kapcsolja be, és csak a lezáró idézőjel beírása kapcsolja ki – és javítani akarjuk az utoljára beírt karaktert. Ekkor az idézőjel üzemmódot az ESC O billentyűkkel kikapcsoljuk, a kurzor balra billentyűvel egy helyet balra lépünk, és kijavítjuk a hibás karaktert. Ilyen – idézőjelek közötti helyeken végzendő – javítások esetén előbb mindig ki kell lépni az idézőjel üzemmódból, mert egyébként a kurzormozgató gomb lenyomásának hatására egy KURZOR BALRA karakter jelenne meg a képernyőn.

Még egy helyreigazítás a kézikönyvben:
Az ESC O nem szünteti meg az automatikus beszúrás módot! Ezt csak az ESC C kapcsolja ki. (De megszünteti az INST/DEL gombbal létrehozott beszúrást.)

A legtöbb üzemmód és művelet főként a szövegfeldolgozási munkában hasznosítható. Az olyasmit mint pl. teljes sorok törlése vagy automatikus beszúrása, sok más számítógépnél csak körülményes programozással lehet megvalósítani. A Plus/4-es esetén ehhez elegendő két billentyű lenyomása. De vajon hogyan lehet a számítógép tudtára adni, hogy pl. be kell kapcsolnia az automatikus beszúrás üzemmódot? Írjuk be közvetlen módban a következőt:

```
PRINT CHR$(27)"A"
```

Menjünk a kurzorral a képernyőn levő valamilyen karaktersorra, és nyomjuk le valamelyik betű billentyűjét. Amint látjuk, a számítógép bekapcsolta az automatikus beszúrás üzemmódot. Ez a parancs programokban is használható, természetesen az

elején valamilyen programsorszám áll. Ily módon valamennyi ESCAPE üzemmód és művelet használható programokban is. Javasoljuk, hogy a programozási munka során amikor csak lehet, használjuk ki az ESCAPE által nyújtott lehetőségeket, mert ezáltal sok munkát takaríthatunk meg.

7.8.1. A „képernyőablak" programozása

Bizonyára feltűnt, hogy két ESC műveletről még nem esett szó. Ezek az ESC T és az ESC B műveletek, amelyek a Plus/4-esnél a „képernyőablak" meghatározására alkalmasak. Képernyőablak alatt a képernyőnek egy meghatározott részét értjük. Ennek segítségével elérhetjük, hogy például egy parancs eredményét megjelenítsük a képernyő egy körülhatárolt részén úgy, hogy eközben a képernyő többi része nem változik. Nézzük meg ezt közelebbről:

ESC T

A Plus/4-esnél egy-egy ablakot két paraméter határoz meg: az ablak bal felső és a jobb alsó sarka. A bal felső sarkot az ESC T billentyűkombinációval határozzuk meg: lenyomásával az ablak bal felső sarkát a kurzor aktuális pozíciójába rögzítjük. Hasonlóképpen rögzíthetjük a jobb alsó sarkot az

ESC B

billentyűkkel.

Lássunk erre egy példát: A HOME billentyű lenyomásával állítsuk a kurzort a képernyő bal felső sarkába. Ezután nyomjuk le az ESC T billentyűket. Ezzel rögzítettük az ablak bal felső sarkát. Menjünk a kurzorral a harmadik sor tizedik oszlopába, nyomjuk le az ESC B billentyűt. Ezzel definiáltunk egy ablakot. A billentyűzetről beírt valamennyi adat, ill. egy program kiírás a képernyőnek ebben, az általunk definiált részében jelenik meg. Vigyük a kurzort a HOME billentyűvel a képernyő bal felső sarkába. Ha ezt a billentyűt kétszer egymás után lenyomjuk, akkor ezzel töröljük az ablakot (az ablak tartalma természetesen megmarad).

A képernyő ilyen módon való felosztását programból is elvégezhetjük. Az ehhez szükséges programsorok:

```
100 PRINT CHR$(27) "T"
```

Ez a programsor az ablak bal felső sarkát definiálja.

A

```
200 PRINT CHR$(27) "B"
```

programsor az ablak jobb alsó sarkát definiálja.

A

```
300 PRINT CHR$(19) CHR$(19)
```

programsor a definiált ablakot törli, és a kurzort a képernyő bal felső sarkába állítja. Mivel a Plus/4-es egy programban csak egy ablak definiálását engedi meg, ezért ha egy ablakot át akarunk definiálni, akkor ezt külön kell programozni. Van azonban egy ennél elegánsabb megoldás is.

7.8.2. Ablakprogramozás POKE paranccsal

A Plus/4-es a képernyő méreteit négy tárcímen tárolja. Mivel ezek a tárcímek a RAM-ban vannak, ezeket POKE parancsokkal könnyűszerrel megváltoztathatjuk. A tárcímek és tartalmuk:

Tárcím	Tartalom	Max.érték	Min.érték
2021	alsó szél	24	0
2022	felső szél	24	0
2023	bal oldali szél	39	0
2024	jobb oldali szél	39	0

Ezekkel a POKE parancsokkal tetszőleges méretű ablak definiálható, csak arra kell vigyázni, hogy a megadott értékeken belül maradjunk. Az egyes értékeket a képernyő kezdőpontjától kell számítani, ami a nulladik sor nulladik oszlopában van. Nézzünk egy példát:

```
10 POKE 2021,19:POKE 2022,15
20 POKE 2023,19:POKE 2024,39:PRINT CHR$(19)
```

Ezzel a két sorral egy ablakot definiáltunk, amelynek a bal felső sarka a képernyő 16. sorában, a bal alsó sarka pedig a képernyő 20. sorában van. Az ablak bal oldali széle a képernyő 20., a jobb oldali a 40-ik oszlopban van. A PRINT CHR$(19) parancs az ablak definiálását követően a kurzort az ablak bal felső sarkába állítja. Érdemes kipróbálni!

Az ESCAPE billentyűk használatának megkönnyítésére rövid kis gépi kódú programot írtunk.

Az ilyen programokat BASIC nyelvű betöltő programoknak nevezik. Ezek a programok a gépi kódú programot „bePOKE-olják" a tárba. A programot indítás előtt feltétlenül tároljuk, mert a program a végrehajtása után saját magát törli.

A BASIC programot a RUN paranccsal kell indítani. Ha ekkor a "HIBA A DATA SOROKBAN" üzenet jelenik meg a képernyőn, akkor az adatok összege nem egyezik meg a 70-es sorban levő vizsgáló végösszeggel. Nézzük át a DATA sorokban levő adatokat, és javítsuk ki a hibát. Ezután a programot ismét tárolni kell.

Ha a program beírása rendben megtörtént, akkor a képernyő első sorában megjelenik egy fekete csík.

Nyomjuk le az ESC billentyűt. A képernyőkeret színének meg kell változnia. Ha ekkor lenyomjuk az M billentyűt, akkor ez a betű inverz módban jelenik meg a képernyő első sorában, és a képernyő keretének színe újra az eredeti lesz. Ezzel kikapcsoltuk a képernyő görgetését. Az ismételt bekapcsoláshoz újra meg kell nyomni az ESC billentyűt, majd az L betűt. Most az inverz M is eltűnik.

A program, a beszúrás üzemmódot is be tudja mutatni. Ekkor egy inverz A jelenik meg.

Az ESC billentyűhöz még egy műveletet rendeltünk hozzá. A keret, a háttér és a karakterek színének megváltoztatása után nyomjuk le először az ESC, majd az INST/DEL billentyűt. A képernyő – az első sor kivételével – ugyanolyan színekben jelenik meg, mint a bekapcsolás vagy egy RESET után, de a tárban levő program

érintetlen marad. Az egész program az ESC és a CLEAR/HOME lenyomásával ki is kapcsolható.

Ha a programra újra szükségünk lenne, akkor a SYS 1630 paranccsal ismét indítható. Sok sikert kívánunk az ESC üzemmódok és műveletek alkalmazásához.

```
10 REM ESC
20 FORI=1630TO1751
30 READI$
40 POKEI,DEC(I$)
50 T=T+DEC(I$)
60 NEXT
70 IFT<>13999THENPRINTCHR$(130);"HIBA A DATA SORBAN";CHR$(132):END
80 SYS1630:NEW
100 DATA 78,A9,6F,8D,14,03,A9,06,8D,15,03
110 DATA A9,00,85,D0,58,60,A4,C6,C0,40,F0
120 DATA 38,A5,D0,D0,06,C0,34,D0,30,F0,04
130 DATA C0,34,F0,2A,49,FF,85,D0,AD,19,FF
140 DATA 49,FF,8D,19,FF,C0,00,D0,06,20,4E
150 DATA D3,20,0B,F3,C0,39,D0,10,78,A9,0E
160 DATA 8D,14,03,A9,CE,8D,15,03,58,A9,20
170 DATA D0,0C,A6,CD,D0,06,EA,A9,11,20,D2
180 DATA FF,A9,A0,A2,27,9D,00,0C,CA,10,FA
190 DATA A0,E9,07,10,05,A9,8D,8D,01,0C,A0
200 DATA EA,07,10,05,A9,81,8D,03,0C,4C,0E,CE
```

Ezzel végére értünk a billentyűzetről szóló fejezetnek. Ajánljuk, hogy minél gyakrabban használják az ESC műveleteket, mert ezekkel jelentős mértékben megkönnyíthető a programozói munka.

7.9. PÉLDAPROGRAM: AUTOVERSENY

A következő program néhány ESC műveletet használ. A képernyőn megjelenik egy meglehetősen kanyargós „út". Az „autót" úgy kell kormányozni, hogy ne ütközzünk az út széléhez. Az autó balra a Z billentyűvel, jobbra a / billentyűvel kormányozható. A „száguldás"-t a szóköz billentyű indítja.

Most is ajánljuk, hogy a programot indítás előtt feltétlenül tároljuk. A program tartalmaz egy gépi kódú rutint, amelytől a számítógép igen könnyen „fejreállhat", ha valamit rosszul írtunk be.

```
10 REM AUTOVERSENY
20 PRINTCHR$(147):COLOR0,1:COLOR1,2
30 FORI=1630TO1731:READH$:POKEI,DEC(H$):NEXT
40 FORI=1TO25:PRINTCHR$(27)"W";:PRINTTAB(13)"I"SPC(16)"I"CHR$(19):NEXT
50 L=13:S=16:VOLS
60 POKE239,0:GETKEYA$:IFA$=" "THENTI$="000000":GOTO70:ELSE60
70 SYS1630
80 X=X+1:IFX>10THENX=0:S=S-1
90 LA=INT(RND(0)*22+1):IFLA<LTHENLB=-1:ELSELB=1
100 FORI=LTOLASTEPLB:PRINTTAB(I)"I"SPC(S)"I"CHR$(27)"W"CHR$(19)
110 IFPEEK(211)=1THEN130:ELSENEXT
120 L=LA:GOTO80
130 SYS65418:REM VEKTOROK VISSZAALLITASA
```

```
140 FORI=1TO10:POKE65286,PEEK(65286)OR7:POKE65287,15:SOUND1,200,1
150 FORJ=1TO10:NEXTJ:POKE65286,27:POKE65287,8:NEXTI
160 CHAR,0,15,STR$(TI)+"PONTOT ERTEL EL."
170 PRINT:PRINTCHR$(27)"Q";
180 POKE239,0:INPUT"MEG EGY MENET (I/N)";IN$
190 IFIN$="I"THENRUN
200 END
210 DATA 78,A9,7F,8D,14,03,A9,06,8D,15,03
220 DATA A9,D4,85,D0,A9,0F,85,D1,A9,01,85
230 DATA D2,A9,F5,A0,00,91,D0,84,D3,58,60
240 DATA C6,D2,A5,D2,D0,0F,A9,06,85,D2,20
250 DATA 11,DB,C9,5A,F0,1E,C9,2F,F0,25,A0
260 DATA 00,B1,D0,C9,09,F0,07,A9,F5,91,D0
270 DATA 4C,0E,CE,A9,5A,91,D0,A9,01,85,D3
280 DATA 4C,0E,CE,A5,D0,C9,C0,F0,E0,C6,D0
290 DATA 4C,94,06,A5,D0,C9,E7,F0,D5,E6,D0
300 DATA 4C,94,06
```

8. A változók

Írjuk be a számítógépbe az alábbi két rövid sort:

```
A = 5
PRINT A
```

A gép egy 5-öst ír ki. A beírt számot tehát tárolta, mégpedig mint egy változót. A változó olyan érték, amelynek neve van, és amely ezen a néven a számítógépben bármikor elérhető és megváltoztatható. A példánkban szereplő változó neve A, és az értéke jelenleg 5. Ezt az értéket azonban megváltoztathatjuk, ha azt írjuk pl. hogy A = 10.

8.1. A VÁLTOZÓNEVEK

A változó neve egy vagy több karakterből állhat, de az első karakternek betűnek kell lennie. A többi karakter lehet betű is és számjegy is (alfanumerikus). A változó neve tetszőleges hosszúságú lehet, az azonosítás csak az első két karakter alapján történik. Aggodalomra azonban nincsen ok – egy programban soha nem kell annyi változót használni, ahány kombinációt két karakter ne tenne lehetővé. Ne feledjük tehát, hogy az olyan változókat, mint KE, KER, KERT a számítógép egy és ugyanazon változónak tekinti, mivel ezek neveiben az első két karakter azonos. Ezért azt javasoljuk, hogy a változókat csak két karakterrel jelöljük.

A változók nevei nem tartalmazhatnak BASIC kulcsszavakat. Ügyeljünk a lefoglalt változónevekre is. Próbáljuk ki pl. ezt:

```
VASALARCOS = 55
```

Erre a számítógép válasza:

```
?SYNTAX ERROR
```

Vajon mi lehet a hiba? A változó nevében szerepel a koszinusz függvény BASIC kulcsszava (cos), amit persze itt nem ilyen értelemben használunk. A számítógép így nem is tud mit kezdeni vele.

Néhány tiltott változónév:
MINTA, TERULET, VOLUMEN, EXPORT, MASINA stb.
Természetesen nem használható az ON, TO vagy az IF sem.

8.2. A VÁLTOZÓK TÍPUSAI

Eddig csak az úgynevezett numerikus változókkal ismerkedtünk meg. A változók azonban összesen háromféle típusúak lehetnek:

- numerikus változók (lebegőpontos változóknak is nevezik),
- egész típusú változók (integer),
- szöveg típusú változók (sztring-változó).

A számítógép a változók típusát a nevük szerint különbözteti meg. A numerikus változókról már volt szó. Ezeket megkülönböztetés nélkül névvel jelöljük, mint pl. A6 vagy ALARC (de nem ALARCOS!). A numerikus változók 10^{-39} és 10^{38} közötti értékeket vehetnek fel.

Az egész típusú változók neveinek végére % jelet (százalék) kell írni, tehát: A% vagy C6%. Ezek a változók csak egész számokat tartalmazhatnak, tehát tizedes részt nem. Az ilyen típusú szám értéke −32768 és +32767 között lehet. Előnye, hogy kevesebb helyet foglal el a tárban, és a program végrehajtási sebességét növeli. Ennek ellenére ritkán használják ezt a típusú változót.

Végül vannak a szöveg típusú, vagy más néven sztring-típusú változók. E változók neve után egy $ jel (dollár) áll, így: A$ vagy C6$. E változók nevei állhatnak betűkből, számjegyekből és más, különleges karakterből (így grafikus és vezérlőkarakterekből is). Egy programon belül egyidejűleg használható az A, az A% és az A$ változó, azaz a különböző típusú változóknak lehetnek azonosak a nevei.

Például:

```
A = 12.34
A% = 12
A$ = "COMMODORE"
```

8.3. FOGLALT VÁLTOZÓK

Azt már megállapítottuk, hogy egy változónak bármilyen neve lehet, feltéve, hogy a névben nincs „elrejtve" egy BASIC kulcsszó. Van azonban még egy korlátozás: a Plus/4-es foglalt változói. Ezek:

```
ST  TI  TI$  ER  ERR&  EL  DS  DS$
```

Nézzünk egyet e változók közül. Írjuk be:

```
PRINT TI$
```

A képernyőn megjelenik egy hatjegyű szám, amely a Plus/4-esben levő óra aktuális állását mutatja. Nos, igen, a számítógépbe be van építve egy (24 órás) óra. Ezt az órát be is állíthatjuk:

```
TI$ = "óóppss"
```

ahol az „óó" a beállítani kívánt időponthoz az órák, a „pp" a percek, az „ss" pedig a másodpercek számát jelöli.

Beállításkor mind a 6 helyet ki kell tölteni. Ily módon a beépített órát beállíthatjuk, és az időt a TI$ változóval bármikor le is kérdezhetjük. Ezt természetesen programon belül is megtehetjük.

A TI változó szintén az idő mérésére alkalmas. A számítógép a TI értékét minden 60-adik másodpercben 1-gyel megnöveli. A TI-ben levő számot 60-nal elosztva egy eltelt időtartamot kapunk másodpercekben. A TI$ és a TI a számítógép bekapcsolásakor vagy egy RESET után 0 értékről indul. Ha a TI$-ba új értéket írunk – tehát ha állítjuk az órát – akkor a számítógép a TI-ben levő értéket ehhez az álláshoz automatikusan hozzáigazítja.

Az ST az úgynevezett státusz változó. Ez a változó a számítógép és a külső egységek, így a kazettás egység, lemezmeghajtó vagy a nyomtató közötti adatbe- és kiviteli műveletek során jut szerephez. Így például ez a változó jelzi, ha az adatátvitel során valamilyen hiba történt.

Az ER, ERR$ és az EL változókkal a hibakezelésről szóló, 19. fejezetben foglalkozunk, valamint az A függelékben. A DS és a DS$ változókról a lemezegység programozását ismertető fejezetben írunk.

8.4. A DIMENZIONÁLÁS

Tegyük fel, hogy egy programban több nevet kell használnunk, így pl. a „Kovács" a „Molnár", a „Horváth" és a „Szabó" neveket. Ezeket a neveket beírhatjuk szöveg típusú változókba:

```
A$ = "KOVACS"
B$ = "MOLNAR"
C$ = "HORVATH"
$ = "SZABO"
```

Megoldhatjuk azonban ezt másként is:

```
A$(1) = "KOVACS"
A$(2) = "MOLNAR"
A$(3) = "HORVATH"
A$(4) = "SZABO"
```

A változóknak most azonos a nevük, de meg vannak számozva. Az ilyen változókat indexelt változóknak nevezzük. A programozási munkában ezeknek nagy jelentőségük van. Nézzük ehhez először a következő programot:

```
10 REM CIMEK BEIRASA
20 PRINT CHR$(147)
30 X=X+1
40 IFX=11 THEN 170
50 PRINT"IRJA BE A(Z) ";X;". CIMET!"
60 PRINT "(BEFEJEZES AZ ";CHR$(34);"XXX";CHR$(34);" BEIRASAVAL)"
70 PRINT
80 INPUT"VEZETEKNEV";V$(X)
90 IF V$(X)="XXX" THEN 190
100 INPUT"UTONEV";U$(X)
110 IFU$(X)="XXX" THEN 190
120 INPUT"IR.SZAM ES HELYSEGNEV";H$(X)
```

```
130 IFH$(X)="XXX" THEN 190
140 INPUT"UTCANEV";UT$(X)
150 IFUT$(X)="XXX"THEN 190
160 GOTO 30
170 REM A CIMEK KIIRASA
180 PRINT"MAXIMUM 10 CIM!"
190 FOR A = 1 TO X - 1
200 PRINT CHR$(147);"A(Z) ";A;". CIM: ";V$(A);" ";U$(A)
210 PRINT
220 PRINT SPC(15) H$(A)
230 PRINT SPC(15) UT$(A);" U."
240 PRINT "NYOMJON LE EGY BILLENTYUT!"
250 GETKEY B$
260 NEXT
270 INPUT"MEGISMETELJEM A KIIRAST (I/N)";IN$
280 IF IN$="I" THEN 190
```

Rövid programmagyarázat:

A 10-es és a 170-es programsorok úgynevezett REMark – megjegyzés – utasítássorok, amelyek a program működését nem befolyásolják, viszont segítségükkel a programlista áttekinthetőbbé tehető.

A 20-as programsor törli a képernyőt.

A 30-as sorban a szöveges változónk sorszámát (az úgynevezett indexet) 1-gyel megnöveljük.

A 40-es sorban megvizsgáljuk, hogy beírtunk-e már 10 címet. Ha igen, akkor elugrunk a 170-es sorra.

Az 50-es sor megadja a beírandó cím sorszámát.

A 60-as sor egy üzenetet ír ki, miszerint az adatbevitelt az "XXX" beírásával befejezhetjük.

A program a 150-es sorig az adatokat kéri be. Minden adatbevitel után megvizsgálja, hogy a beírt adat nem "XXX"-e, és hogy az adatok kiírásának kell következnie.

A 190-es és a 260-as sorok a FOR-TO-STEP-NEXT utasítássorozatot tartalmazzák (ezzel majd a programciklusokról szóló, 13. fejezetben foglalkozunk részletesebben). Itt az A ciklusváltozó értékét 1-gyel növeljük mindaddig, míg az X-1 értéket el nem éri. Az X változó tartalmazza ugyanis a beírt címek számát. Mivel ezt már az első cím beírása előtt 1-gyel megnöveltük, most, a kiíráskor 1-et le kell vonni, hiszen az "XXX" címet nem akarjuk kiírni. A programnak a FOR és a NEXT, azaz a 200-as és a 250-es sorok közötti része annyiszor kerül végrehajtásra, ahány címet beírtunk.

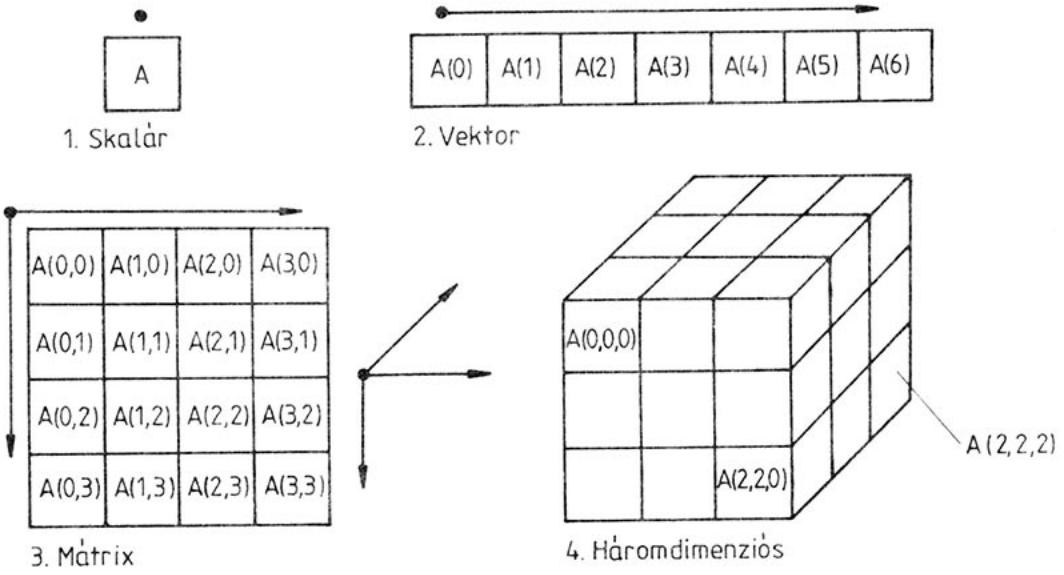
A 220-as és 230-as programsorban szerepel a "SPC(14)" bejegyzés, ami azt jelenti, hogy a programnak 14 üres karaktert kell a képernyőre írnia.

A 250-es sorban a GETKEY utasítás megállítja a program futását, és egy billentyű lenyomására vár.

A 270-es és 280-as sorban megkérdezi a program, hogy megismételje-e a kiírást. A program csak az "I" billentyű lenyomását figyeli.

Ez a kis program csak egy példa, amellyel komolyabb feladatokat természetesen nem lehet megoldani. Csak azt akartuk bemutatni, hogy hogyan lehet bánni olyan változókkal, amelyek indexeltek. Amint a példából látható, ezek a változók egészen egyszerűen sorszámozva vannak. A sorszámuk alapján aztán a különböző változókat egymással össze lehet kapcsolni, mint pl. az 1. számú vezetéknevet az 1. számú

utónévvel, az 1. számú helységgel és az 1. számú utcanévvel. Az indexes változókat egydimenziós változóknak is nevezik. Vannak többdimenziós változók is. Nézzük meg ehhez a 8.1. ábrát:



8.1. ábra Többdimenziós változók

Képzeljük el, hogy az első esetben, ahol csak egyetlen változónk van, ez a változó egy kártyalap. A második esetben már több kártyalapunk van a kártyadobozban, a harmadik esetben pedig már több kártyadobozunk is van. A negyedik esetben a változónk háromdimenziós lett. Ezt egy kockával tudjuk szemléltetni. A gondolatmenetet folytathatjuk tovább – lehetne négydimenziós változónk is (ennek megrajzolására azonban nem volt ötletünk). Ily módon akárhány indexű változó is elképzelhető, korlátot tulajdonképpen csak az elhelyezésükhöz szükséges gépi tárkapacitás jelent.

A dimenzionálásra a BASIC nyelv DIM parancsa szolgál.

A DIM parancs

A számítógépnek a változók elhelyezésére a tárban helyet kell lefoglalnia. Ezt a gép változókként 10 indexsorszámig automatikusan végzi. Ez 11 elemet jelent, mert az indexelés a 0. sorszámmal kezdődik. A mintapéldánkban ezt ki is használtuk. Ha viszont 11-nél több elemre van szükségünk, akkor a számítógépnek meg kell mondanunk, hogy mekkora tárhelyet foglaljon le. Ezt a műveletet nevezik „dimenzionálás"-nak vagy „deklarálás"-nak.

Ha azt akarjuk, hogy az A változó 101 elemű legyen, akkor ezt kell írnunk:

```
10 DIM A(100)
```

Egy programon belül, egy változót csak egyszer lehet – és egyszersmind kell is – DIMenzionálni. Egy DIM parancsal egyszerre több változó is dimenzionálható:

```
10 DIM A$(25,4), B(50), C%(150)
```

Ezt a dimenzionálást a CLR paranccsal érvényteleníteni lehet, viszont ezáltal az összes változó törlődik. A változókat természetesen ezt követően újra lehet dimenzionálni.

A DIM parancsban az index lehet egy változó is. Így pl. egy programban meg lehet kérdezni, hogy hány címet akarunk feldolgozni. Ilyenkor azonban vigyázni kell, mert a számítógép azonnal hibát jelez, ha a kívánt tárhely nem lenne biztosítható. Ezzel kapcsolatban utalunk a hibakezelésről írt fejezetre.

```
10 INPUT "IRJA BE A KIVANT DARABSZAMOT";A
20 DIM A$(A)
```

Ha egy változóra másodszor is kiadjuk a DIM parancsot anélkül, hogy azt előzőleg egy CLR paranccsal töröltük volna, akkor a számítógép a

REDIM'D ARRAY ERROR
(újradimenzionálás)

hibaüzenetet írja ki. Ha az indexes változó (amelyet tömbnek is neveznek) tárigénye nagyobb lenne a rendelkezésre álló tárkapacitásnál, akkor a számítógép az

OUT OF MEMORY ERROR
(kevés a tárhely)

hibaüzenetet írja ki.

8.5. PÉLDAPROGRAM: AKASZTÓFA JÁTÉK

A következő program egy kitalálós játék programja. A játékban két játékos játszhat. A lényege az, hogy az egyik játékosnak egy szót kell beírnia a számítógépbe, a másiknak pedig ezt ki kell találnia. Az első játékos által beírt szót – amit persze úgy kell beírnia, hogy az ellenfél ne lássa – egy változó tárolja. A beírás után a képernyőn annyi kötőjel jelenik meg egymás mellett, ahány betűből áll a szó. A második játékosnak kilenc lehetősége van arra, hogy az elrejtett szót kitalálja. Beírhat egyetlen betűt, de akár egy egész szót is. Ha a keresett szó tartalmazza a beírt betűt, akkor ez a betű megjelenik a képernyőn. Ha viszont a betű nem szerepel a szóban, vagy a beírt szó nem a keresett, akkor a képernyőre lépésről lépésre kirajzolódik egy akasztófa.

Jó szórakozást a játékhoz!

```
10 REM AKASZTOFA
20 GRAPHIC0,1:PRINTTAB(10);"******AKASZTOFA******"
30 CHAR,5,10,""
40 PRINT"1. JATEKOS: IRJA BE A SZOT"
50 PRINT
60 POKE239,0:INPUTS$:SA$=S$
70 PRINTCHR$(147)
80 COLOR0,1:COLOR1,2:GRAPHIC1,1
90 CHAR,0,19,"2. JATEKOS: IRJON BE EGY BETUT"
100 CHAR,11,20,"VAGY A GONDOLT SZOT"
110 FORI=1TOLEN(S$)
120 CHAR,I,23,"-"
```

```
130 NEXTI
140 X=0:LS$="":R=0:FORI=0TO39:CHAR,I,24," ":NEXT
150 POKE239,0:GETKEYV$
160 IFV$=CHR$(20)ANDX=0THEN150
170 IFV$=CHR$(13)ANDX=0THEN150
180 IFV$=CHR$(20)THENX=X-1:LS$=LEFT$(LS$,X):CHAR,1+X,24," ":
GOTO230
190 IFV$=CHR$(13)THEN250
200 IFV$<"A"ORV$>"Z"THEN150
210 LS$=LEFT$(LS$,X)+V$
220 X=X+1
230 CHAR,1,24,LS$
240 GOTO150
250 IFX>1ANDLS$=S$THEN900
260 IFX>1THEN320
270 FORI=1TOLEN(S$)
280 IFMID$(SA$,I,1)=LS$THENCHAR,I,23,LS$:R=1:RG=RG+1:
MID$(SA$,I,1)="1"
290 NEXTI
300 IFRG=LEN(S$)THEN900
310 IFR=1THEN140
320 F=F+1
330 ONFGOSUB400,450,500,550,600,650,700,750,800
340 IFF<9THEN140
350 CHAR,0,21,"EGYIK SEM JO!!! A HELYES SZO:"
360 CHAR,1,23,S$,1
370 POKE239,0
380 GETKEYA$:RUN
400 CIRCLE,80,140,40,20,275,85
410 DRAW,40,138TO120,138
420 PAINT,45,135
430 RETURN
450 FORI=0TO2:DRAW,79+I,120TO79+I,5:NEXT
460 RETURN
500 FORI=0TO2:DRAW,80,5+ITO140,5+I:NEXT
510 RETURN
550 FORI=0TO2:DRAW,80,40-ITO95-I,5:NEXT
560 RETURN
600 DRAW,130,5TO130,15
610 RETURN
650 CIRCLE,135,20,8,5,,,320
660 RETURN
700 CIRCLE,130,55,8,27
710 RETURN
750 CIRCLE,115,40,15,2,,,320
760 CIRCLE,145,40,15,2,,,40
770 RETURN
800 CIRCLE,120,99,20,3,,,285
810 CIRCLE,140,99,20,3,,,75
820 DRAW,133,22TO135,24
830 DRAW,135,18
840 DRAW,138,21
850 RETURN
900 CHAR,1,23,S$
910 FORI=0TO39:CHAR,I,19," ":CHAR,I,20," ":NEXT
920 CHAR,0,24,"EZT ELTALALTA!",1
930 GETKEYA$:RUN
```

9. Programok írása a Plus/4-esen

9.1. A PROGRAM MÓD

Ha a számítógép tárában egy program van, és ezt a BASIC nyelv RUN parancsával elindítjuk, akkor a számítógép a program módba kerül, azaz a számítógép vezérlését a program veszi át. A program használójának a számítógéppel nem kell törődnie, tennivalója mindössze annyi, hogy a program esetleges kéréseinek megfelelően a billentyűzeten keresztül adatokat írjon be.

Egy program futását a STOP billentyű lenyomásával lehet megszakítani. Ha a számítógép a program végrehajtása során számára nem értelmezhető utasításokat talál, mint pl. 0-val való osztást vagy egy hibásan beírt utasításszót, akkor a programot saját maga megszakítja, és kiír egy hibaüzenetet. A számítógépnek ezután a kezelője adhat parancsokat.

Ugyanez történik – természetesen hibajelzés nélkül – akkor is, ha a számítógép a program végrehajtása során egy END utasítást tartalmazó sorhoz ér, vagy ha elfogynak a végrehajtandó programsorok. Ebben az esetben a számítógép a „READY" (=kész) üzenettel jelentkezik vissza.

9.2. A PROGRAMOK INDÍTÁSA

Egy program kétféle parancsal indítható. Nézzük először a gyakrabban használatosat.

Szintaxis:

RUN

vagy

RUN sorszám

Ha beírjuk a RUN parancsot, akkor a számítógép először is törli az összes változót. Ezután megállapítja a program első sorszámát, és itt elkezdi a program végrehajtását. Ha a RUN után beírunk egy sorszámot is, akkor a program végrehajtása ezzel a sorral kezdődik. A beírt sorszámnak a programban létező sorszámnak kell lennie. Ellenkező esetben a gép hibaüzenetet ír ki (UNDEF'D STATEMENT).

Más paranccsal is elindítható egy program.

Szintaxis:

GOTO sorszám

Ez a parancs hasonlóan működik, mint a RUN sorszám, azzal a különbséggel, hogy

most a változók nem törlődnek, azaz megtartják a korábbi értéküket. A sorszámot feltétlenül meg kell adni. Ez a parancs nagyon jól használható programrészek vizsgálatára, ha egy programot a próbafuttatás során a számítógép hibaüzenettel megszakított.

9.2.1. Egy program megállítása

Néhány kivételtől eltekintve a programok futása a RUN/STOP billentyű lenyomásával bármikor megállítható. Ekkor a képernyőn megjelenik a "BREAK IN sorszám" üzenet. Ilyenkor lehetőségünk van arra, hogy pl. a fontosabb változók tartalmát kiírassuk és megváltoztassuk. A CONT parancs beírására a program futása azon a helyen folytatódik, ahol megszakítottuk.

Egy másik lehetőség a CTRL/S. Ha ezt a két billentyűt egyidejűleg lenyomjuk, akkor a program futása nem szakad meg, csak megáll. A képernyőn nem jelenik meg semmiféle üzenet. A program futása akkor folytatódik, ha egy karakterbillentyűt (tehát *nem* CTRL, SHIFT, stb.) lenyomunk. Ha azt akarjuk, hogy a programunkat a CTRL/S kombinációval ne lehessen megállítani, akkor a program első soraként ezt kell beírni:

```
POKE 2039,6
```

A billentyűkombináció hatását a

```
POKE 2039,0
```

beírásával kapcsolhatjuk be újra.

Nézzünk egy parancsot, amely programba írva szintén megszakítja a program futását. Ez a STOP parancs.

Szintaxis:

STOP

Ezt a parancsot bármely programba beírhatjuk. Feladata elsősorban hibák behatárolása, és a hatása ugyanaz, mintha megnyomtuk volna a RUN/STOP billentyűt. A változók tartalma ugyancsak kiíratható és megváltoztatható. A program futtatása a CONT paranccsal folytatható. A program végrehajtása abban a sorban folytatódik, amelyik a STOP utasítás után következik.

FIGYELEM: A program a CONT paranccsal nem folytatható, ha azt bármilyen módon megváltoztattuk vagy ha számolási hibát vétettünk. Ilyenkor a „CAN'T CONTINUE ERROR" (Nem folytatható hiba) hibaüzenetet kapjuk.

9.2.2. Programok befejezése

A számítógép a program végrehajtását a program utolsó sorában automatikusan befejezi, hacsak ez a sor nem tartalmaz valamilyen, a program belsejébe mutató visszaugrást. Ha a gép befejezte a program végrehajtását, akkor a „READY" üzenettel visszajelentkezik, és újabb parancsokra vár. Van azonban olyan utasítás is, amelylyel egy programot be lehet fejezni: END.

Szintaxis:

END

Hatása ugyanaz, mint a STOP-nak, csak a BREAK IN... üzenet elmarad. Ezt a parancsot mindazon hely(ek)re be kell írni, ahol a programnak esetleg be kell fejeződni. A program bármelyik helyére beírható (tehát nem szükségszerűen az utolsó sorba). Egy program gyakran tartalmaz olyan alprogramokat, amelyeket csak esetenként kell végrehajtani. Ha egy programnak esetleg egy alprogram előtti sorban kell befejeződnie, akkor ebben a sorban lennie kell például egy END utasításnak. Mivel az alprogramot mindig RETURN utasítás zárja le, ha nem írnánk be az END utasítást, akkor a programot a számítógép a „RETURN WITHOUT GOSUB ERROR IN sorszám" (RETURN GOSUB nélkül hiba a ... számú sorban) hibaüzenettel megszakítaná.

Az END a program dokumentálását végzi, mivel ez egy programlistában megmutatja, hogy a program hol, és milyen feltételek mellett fejeződik be. Az END utasítással befejeződő program a CONT paranccsal újra indítható.

9.3. A PROGRAMÍRÁST TÁMOGATÓ PARANCSOK

Azért, hogy a számítógép meg tudja állapítani, hogy egy beírt parancsot azonnal végre kell-e hajtania (közvetlen mód), vagy hogy a beírt parancs egy program része (program mód), az utóbbi esetben a parancsot egy sorszámmal kell ellátni. Ez a szám 0 és 63999 között lehet. Egy ilyen sor beírása, majd a RETURN lenyomása után ez a sor bekerül a számítógép programtárába.

A Plus/4-esen a programok beírásakor a kurzor a képernyő teljes felületén szabadon mozgatható, ún. képernyő-orientált szerkesztő. Ennek használatával a programok a képernyőn könnyen szerkeszthetők, a beírt sorok egyszerűen és gyorsan átírhatók. Javítás után azonnal folytatható a program írása.

Egy programsorban több utasítás is állhat. Ezeket az utasításokat kettősponttal (:) kell egymástól elválasztani. Egy programsor maximum 88 karaktert tartalmazhat, ami két képernyősor +8 betű hosszúságnak felel meg.

9.3.1. A LIST parancs

A programozás során bizonyára ezt a parancsot használjuk leggyakrabban. Ezzel bármikor megnézhetjük a teljes programot. Néhány magyarázat a szintaxishoz:

Szintaxis:

LIST	kilistázza a teljes programot.
LIST sorszám	csak a megadott sorszámú sort írja ki.
LIST sorszám –	a megadott sorszámtól kezdődően a teljes programot listázza.
LIST – sorszám	a megadott sorszámig (ezt is beleértve) listázza a programot.
LIST sorszám – sorszám	a megadott sorszámtól a megadott sorszámig listáz.

9.3.2. Az AUTO parancs

Szintaxis:

AUTO szám

E parancs hatására a számítógép saját maga végzi el a programsorok sorszámozását. Ha pl. beírjuk az AUTO 100 parancsot, akkor a sorszámok 100-asával követik egymást. Két sorszám közötti „lépésközt" bármikor meg lehet változtatni. Az automatikus sorszámozást az AUTO parancs sorszám megadása nélküli beírásával vagy AUTO 0 beírásával lehet kikapcsolni:

```
AUTO
```

A RUN vagy a GOTO sorszám szintén kikapcsolja ezt az üzemmódot.
Írjuk be a következőt:

```
AUTO 10
```

A képernyőn megjelenik a READY üzenet. Írjuk be most ezt:

```
10 PRINT "AZ AUTO PARANCS"
```

A képernyő bal szélén most a 20-as szám jelenik meg. Ha most ismét megnyomjuk a RETURN billentyűt, akkor a kurzor egy sorral lejjebb lép. Ha ezt megelőzően lett volna a tárban egy 20-as sorszámú programsorunk, akkor ez a sor most törlődött volna. A programsorok tárolásakor a számítógép megvizsgálja, hogy van-e a tárban egy, a tárolandóval megegyező sorszámú sor. Ha igen, akkor a régebbi programsort kicseréli az újra. A képernyőre nem íródott ki az új sorszám, az AUTO parancs még érvényben van. Ezt ki is próbálhatjuk:

```
5 PRINT "EZ"
```

§ RETURN után megjelenik a 15-ös szám. Az AUTO üzemmódot a RETURN, AUTO, RETURN sorozattal lehet kikapcsolni.

Ha beírjuk a LIST parancsot, akkor egy RETURN után megjelenik a képernyőn a programunk, mégpedig úgy, hogy a programsorok a helyes sorrendben követik egymást. A programot a RUN-nal el is indíthatjuk.

Sűrűn előfordul, hogy egy programba sorokat kell beiktatnunk. Ha egymást követően több sort kell beszúrni, és a sortávolságot kicsire választottuk, akkor könnyen előfordulhat, hogy már nem marad üres sorszám, ahová programsort lehetne beírni. A számítógép erre a problémára is ad megoldást.

9.3.3. A RENUMBER parancs (átszámozás)

Szintaxis:

RENUMBER új kezdősor száma, sorszám lépésköze, régi kezdősor száma.

Ez a parancs átszámozza a programsorokat. Ha a parancsot paraméterek nélkül adjuk ki, akkor az átszámozást az első sortól az utolsó sorig, 10-es lépésközzel végzi el. Az első sorszáma 10 lesz.

A paraméterek:

új kezdősor száma	ez lesz a RENUMBER után az első programsor
sorszám lépésköze	ilyen lépésközben követik egymást a sorszámok
régi kezdősor száma	a régi programnak ettől a sorától indul az átszámozás

A sorszámok átrendezése azzal a paraméterrel kezdődik, amelyet az „új kezdősor sorszámá"-val megadtunk, és a program végéig tart. Ha a „régi kezdősor sorszámá"-t nem adjuk meg, akkor a parancs a teljes programot átszámozza. Az új kezdősor sorszáma nem lehet kisebb egy olyan programsor sorszámánál, amelyre a parancsot nem vonatkoztatjuk.

VIGYÁZAT: a RENUMBER nem működik teljesen hibátlanul. A programozás során 63999-nél nagyobb sorszámok nem használhatók. Ha szükséges, akkor 63999 után is folytatja a számolást 65535-ig, majd 0-tól, de ezek a sorok LIST (sorszám) paranccsal nem hívhatók le és nem is cserélhetők. A LIST paranccsal meg is nézhetjük, hogy ezek a sorok a helyes sorrendben, a program végén vannak. Ezek a programsorok a RUN paranccsal végre is hajthatók. Ezeknél a sorszámoknál szintén a fent leírtak érvényesek.

Ennek a körülménynek előnyei is vannak. Így lehet pl. egy Copyright bejegyzést a RENUMBER-rel ebbe a tartományba tolni, úgy, hogy azt könnyűszerrel nem lehet törölni. Alapjában véve azonban ügyeljünk arra, hogy egy RENUMBER következtében ne lépjünk át 63999-es sorszámot. Másik hibaforrás lehet, ha nemlétező sorszámú sorra hivatkozunk, pl. GOTO 100 és még nincs 100-as sor. Ekkor a program ezt a sort 65535-re sorszámozza át, ami futtatáskor hibát okoz. Ha az előző program még a tárban van, akkor próbáljuk ki ezt:

```
RENUMBER 100.20
```

Ha listázzuk a programot, akkor láthatjuk, hogy az első sor sorszáma 100, a másodiké 120.

9.3.4. A parancsok rövidítése

A legtöbb BASIC parancs rövidítve is írható. Így pl. a PRINT parancs helyett ?-et (kérdőjelet) írhatunk. Ezekkel a rövidítésekkel a programok beírásakor rengeteg munkát megtakaríthatunk. A rövidítve beírt parancsok listázáskor már az eredeti, teljes alakjukban jelennek meg. A rövidítések mind a nagybetű/grafika, mind a nagybetű/kisbetű üzemmódban használhatók. A programok beírásához az utóbbit javasoljuk, mert a másik üzemmódban a parancsok rövidített beírásakor a legtöbb esetben valamelyik grafikus jel íródik ki, ami a program szövegét áttekinthetetlenné teszi. (Listázáskor viszont a teljes szöveg látható.)

A parancsok rövidítésével a programsorban helyet is megtakaríthatunk. Tulajdonképpen egy programsor 89 karakternél többet nem tartalmazhat, de: írjunk be egy 10-es sorszámot, majd ezt követően annyi ?-et, míg két képernyősor +8 karakterhely meg nem telik. Ha most ezt listázzuk, akkor látjuk, hogy több képernyősor is tele lesz PRINT-tel. A Plus/4-es tehát csak a beírt sor hosszát figyeli, és nem azt, hogy a benne levő utasítások később mennyi helyet foglalnak el. Ennek az oka a képernyőszerkesztőben keresendő, és van egy szépséghibája is. Ha ugyanis a kurzorral egy ilyen

kilistázott sorra állunk, és lenyomjuk a RETURN billentyűt, akkor a gép a „STRING TOO LONG ERROR" hibaüzenetet írja ki.

A lehetséges rövidítéseket a könyv C függelékében egy táblázatban foglaltuk össze.

9.4. A PROGRAMOK JAVÍTÁSA

A képernyőorientált szerkesztő használatával igen egyszerűen és gyorsan változtathatók, ill. javíthatók a programok. Anélkül, hogy a programsort újra kellene írni, a programsorban a karakterek megváltoztathatók, törölhetők vagy beszúrhatók.

Ehhez egyszerűen a kívánt karakterre kell állni a kurzorral, és a megfelelő billentyűt le kell nyomni. Ne feledkezzünk meg arról, hogy egy programsorban elvégzett változtatás után le kell nyomni a RETURN billentyűt. A billentyűzetről szóló fejezetben már ismertettük annak a módját, hogy hogyan kell egy karaktert beszúrni, vagy törölni.

A programozás során gyakran előfordul, hogy egész programsorokat vagy programrészeket is törölni kell, mert például jobb megoldás jutott az eszünkbe. Programsorokat a következő módon lehet egyszerűen törölni:

Írjuk be azt a sorszámot, amelyet törölni akarunk, a képernyő valamelyik üres sorába, és nyomjuk le a RETURN-t. Ezzel a sor törlődött.

Ha egy sort véletlenül töröltünk, és a sor még a képernyőn van, akkor a sort nagyon egyszerűen „visszahozhatjuk" úgy, hogy a kurzorral az illető sorra állunk, és lenyomjuk a RETURN-t. A sor most ismét a programtár megfelelő helyére került. Ugyanilyen módon a sorok meg is többszörözhetők. Menjünk a kurzorral az illető sor sorszámára, írjuk át az új sorszámra, és nyomjuk le a RETURN-t. Listázáskor látjuk, hogy ez a programsor bekerült a program megfelelő helyére.

Gyakran lehet szükség egész programrészek törlésére. A Plus/4-es erre is tud megfelelő megoldást.

9.4.1. A DELETE parancs

Szintaxis:

DELETE kezdősor – végsor

Ezzel a paranccsal meghatározott programrészek törölhetők. Ennél a parancsnál sem kell mindegyik paramétert megadni. Így pl. a DELETE 200 csak a 200-as sort törli a tárból, míg a DELETE 200-300 a 200-tól 300-ig terjedő programsorokat. Ha a DELETE –200 parancsot adjuk ki, akkor a 200-as sorig – ezt is beleértve – valamennyi előző sor törlődik. Végül a DELETE 200- a 200-as és az összes ezután következő sort törli.

Az előzőekben leírt valamennyi parancs csak közvetlen módban használható.

Most pedig lássunk egy olyan parancsot, amely a teljes tárat törli.

9.4.2. A NEW parancs

Szintaxis:

NEW

Ezzel a paranccsal nagyon óvatosan kell bánni, mert ez a tárban levő teljes programot törli. Ez a parancs mind közvetlen, mind program módban használható. Elképzelhető pl. egy olyan program, amely egy jelszó beírására vár, és ha nem a megfelelőt kapja, akkor önmagát törli a NEW utasítással. Egyébként: a 20. fejezetben bemutatjuk, hogy hogyan lehet egy NEW paranccsal törölt programot „visszahozni".

9.5. ELŐKÉSZÜLETEK A PROGRAMOZÁSHOZ

Ha 10 programozót felkérnek arra, hogy ugyanannak a problémának a megoldására készítsenek egy-egy programot, akkor minden valószínűség szerint 10 különböző program születik. Bizonyára mindegyik program megoldja az adott problémát, de az egyik program a legáttekinthetőbb, a másik talán a leggyorsabb. Ebben a könyvben az a szándékunk, hogy bemutassuk a számítógép által kínált lehetőségek sokféleségét. Ezenkívül természetesen útmutatást is akarunk adni a helyes és áttekinthető programozásra.

A programozást mindig pontos tervezéssel kell kezdeni. Nem csak azt kell tudnunk, hogy a programnak mit kell elvégeznie, hanem azt is, hogy hogyan végezze el. Fordítsunk elegendő időt a probléma elemzésére. Készítsünk eközben jegyzeteket, amelyek a programozást egyszerűbbé teszik. Készítsünk vázlatot a program lefolyásáról, és eközben ne takarékoskodjunk a magyarázatokkal. A lehető legkevesebb ugrási utasítást használjuk, ezek a programot nehezen áttekinthetővé teszik. Ne feledjük, hogy a gondos tervezés felér egy fél programmal. Az az idő, amit a tervezésre fordítunk, a program megírásakor, de legalábbis hibakereséskor megtérül. A gondos előkészületek jutalma egy olyan program, amely igen gyorsan, és kevés ráfordítással más feladatok megoldására is illeszthető, és amelyet még hónapok múlva is megértünk.

Bemutatunk még egy parancsot, amely szintén javítja a programok áttekinthetőségét.

9.5.1. A REM parancs

Szintaxis:

REM

A REM az angol „remark" (= megjegyzés) szó rövidítése. Ezt a parancsot főleg akkor használjuk, ha egy programrészt magában a programban akarunk megjegyzésekkel ellátni. A REM gondoskodik arról, hogy mindazt, ami ezután a parancs után áll, a számítógép a program végrehajtása során figyelmen kívül hagyja. Javasoljuk, hogy főként a hosszabb programrészeket és rutinokat lássuk el REM programsorok-

ba írt magyarázatokkal. Ha ezt nem tesszük, akkor az is előfordulhat, hogy néhány hét után még a saját programunkat is nehezen tudjuk áttekinteni. Hosszabb programoknál az első sorok lehetőség szerint REM sorok legyenek, amelyekben a programban előforduló változókat dokumentáljuk. Valahogyan így:

```
10  REM VN$ = VEZETEKNEV UN$ = UTONEV HE$ = HELYSEG
20  REM AZ I CIKLUSVALTOZO, AMELYET BARMELYIK PROGRAMRESZ HASZ-
    NALHAT
.
.
100 REM FOPROGRAM
.
.
300 REM BETU KIIRASA CSAK A, S, E IRHATO KI
.
.
500 REM STB.
```

Ha vesszük a fáradságot, és egy programsorba csak egy, vagy max. két, rövid utasítást írunk, akkor ezzel is növeljük a program olvashatóságát. Az igaz, hogy ez a futási sebességnek nem használ, de az időben kritikus szakaszoknál a BASIC által nyújtott lehetőségekhez képest amúgy is jobb megoldásokat kell választani, amivel behozhatjuk az egyébként esetleg elvesztett időt.

Fontos figyelmeztetés: Hosszabb programok írásánál a program egyes részeit időről időre írjuk lemezre vagy kazettára, mert egy esetleges áramkimaradás órák munkáját teheti semmivé. A programokat – különösen ha POKE utasítást is tartalmaznak – az első indítás előtt feltétlenül tároljuk. A számítógép ugyanis ha egyszer „fejre áll", akkor többnyire csak a RESET megnyomása segít. Ekkor pedig a programunk már elveszett.

9.6. STRUKTURÁLT PROGRAMOZÁS

Az, aki ezt a könyvet eddig figyelmesen olvasta, már bizonyára képes arra, hogy néhány programot saját maga is készítsen. Hiszen olyan egyszerű az egész: néhány programsorszám, ezekhez néhány BASIC utasítás, és már kész is a program – gondolhatná valaki –. A dolog azonban nem ilyen egyszerű. Az persze elképzelhető, hogy egy ilyen, sorról sorra, „fejből" beírt program esetleg hibátlanul fut, és még a feladatát is teljesíti, úgyhogy tulajdonképpen elégedettek is lehetnénk. Mi van azonban akkor, ha a program egy kicsit terjedelmesebbre sikerült, és egy idő után valamit változtatni akarunk rajta? Legyen mondjuk ez egy olyan program, amellyel a személyi jövedelemadónkat számítjuk ki. A program működik, és a számítások is helyesek. A programot természetesen tároljuk, hogy a következő évben is felhasználhassuk. De mi van akkor, ha a következő évben módosulnak az adótörvények, és ezért a számításokat is meg kell változtatni? Vagy új programot kell írni, vagy a meglevőt kell változtatni. Természetesen a második megoldást választjuk. Ehhez azonban egy fontos feltételnek teljesülnie kell: a programnak olyannak kell lennie, hogy a változtatást el is tudjuk végezni, azaz olyannyira áttekinthetőnek, hogy még később is tudjuk azt, hogy mi, mikor és hol történik a programban.

Ezek alapján könnyen belátható, hogy bizonyos programozási szabályokat be kell tartani. A programnak világos felépítésűnek kell lennie, és rövid, áttekinthető szakaszokból kell állnia. Röviden szólva, a programnak strukturáltnak kell lennie. Gondoljuk el, hogy a programunkat odaadjuk a barátainknak és ismerőseinknek, hogy azt a saját igényeiknek megfelelően átalakíthassák. Ha ez a program olyan, mint egy „tengeri kígyó", akkor elképzelhető, hogy a működését egy idő után már maga a program írója sem látja át.

A programnak a következő feltételeket kell teljesítenie:

1. Természetesen működnie kell.
2. Legyen áttekinthető.
3. Legyen könnyen kezelhető.
4. Legyen változtatható.
5. Hatékonyan működjön.

Bizonyos körülmények között kényszerülhetünk kompromisszumokra, hiszen előfordulhat, hogy a rendelkezésre álló tárkapacitás nem elegendő.

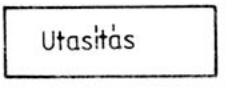
Ha az alábbiakban ismertetésre kerülő szabályokat betartjuk, akkor valószínűleg sok időt kell fordítanunk a programmal kapcsolatos meggondolásokra. Ezzel szemben a tulajdonképpeni „kódolás", azaz a gondolatainknak működőképes programmá való átírása lényegesen kevesebb időt vesz igénybe.

Megjegyzés: olyan, igen rövid programoknál, amelyet csak egyszer, és egy speciális feladatra írunk, nem követelmény a stukturáltság. Amint azonban a programok egyre terjedelmesebbé válnak, és többször használjuk őket, ill. univerzálisaknak kell lenniük, a strukturált programozás már egyenesen követelmény. Még egy megjegyzés ehhez: mi, a könyv írói, beismerjük, hogy még mindig beleesünk abba a hibába, hogy sokszor csak úgy, „fejből" programozunk. Abban az időben, amikor mi ismerkedtünk a BASIC és az assembly nyelvvel, még senki sem beszélt „strukturáltság"-ról. Ennek a szükségességére általában mindenki a saját tapasztalatai alapján jött rá, amikor állandóan csak töröltük a programokat, mert egy idő után már senki sem ismerte ki magát bennük.

A strukturált programozás egyik alapeszköze az ún. folyamatábra vagy blokkdiagram. Ezekkel a problémát grafikusan ábrázoljuk, kis lépésekre bontjuk, és ezeket az időbeli lefolyásuknak megfelelő sorrendben ábrázoljuk. A folyamatábrákat szimbólumokkal rajzoljuk meg, amelyeket most röviden ismertetünk.

9.6.1. A programok folyamatábráiban használt szimbólumok

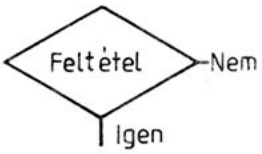
Mint ismeretes, mindent szabványosítanak. Így történt ez a szimbólumokkal is, az ide vonatkozó szabvány a DIN 66001. Lássuk a legfontosabb szimbólumokat:



9.1. ábra Általános művelet

Általános művelet

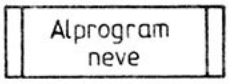
Pl. értékadások, deklarációk, egyszerre több utasítás is lehetséges.



9.2. ábra Elágazás

Elágazás

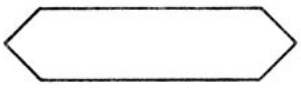
Az elágazási feltételt a szimbólumba be kell írni.



9.3. ábra Alprogram

Alprogram

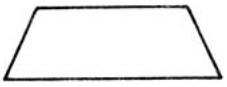
Az alprogram nevét be kell írni, ezen kívül célszerű a hozzá tartozó sorszám beírása is.



9.4. ábra Programmódosítás

Programmódosítás

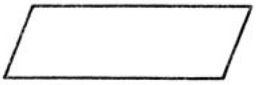
Alapvető változtatások, amelyek az ezután következő programokat érintik, pl. a tárterület átkapcsolása.



9.5. ábra Kézi műveletek

Kézi műveletek

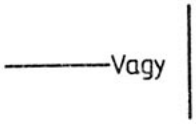
Ide tartoznak olyan műveletek, mint a készülékek bekapcsolása, vagy papír befűzése a nyomtatóba.



9.6. ábra Bevitel, kivitel

Bevitel, kivitel

Adatok kiírása a képernyőre vagy más kiíró készülékre, vagy adatok bevitele a billentyűzetről vagy más adatbeviteli készülékről.



9.7. ábra Folyamatvonal

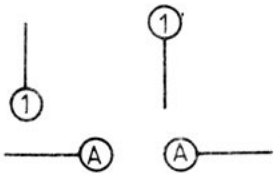
Folyamatvonal

A folyamatábrán az események időbeli lefolyása felülről lefelé, (ill. balról jobbra) a folyamatvonal mentén történik. Az egyes szimbólumokat folyamatvonalak kötik egymással össze.

9.8. ábra Összevezetés

Összevezetés

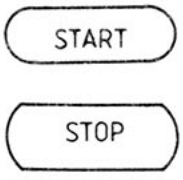
A programban egy olyan hely, amelyet több programrész is elérhet. Pl. GOTO.



9.9. ábra Átlépési hely

Átlépési hely

A folyamatábra valamely helyen megszakítható, és egy másik helyen folytatható. Az átlépési helyeket betűk vagy számok jelölik. Az összetartozó átlépési helyeket azonos betűk jelölik. Egy átlépési helyet több másik is elérheti.

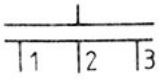


9.10. ábra Határpont

Határpont

A program kezdete vagy a vége. Indulójelből csak egy lehet.

A STOP jelből több is lehet, de mindig csak egy jelhez csatlakozik.



9.11. ábra Elosztás

Elosztás

Többszörös elágazást jelöl.
Pl.: ON...GOSUB...

9.12. ábra Megjegyzések

Megjegyzések

Ne fukarkodjunk a programban megjegyzésekkel. A szaggatott vonal ahhoz a szimbólumhoz vezet, amelyre a megjegyzés vonatkozik. Parancsok is beírhatók.

E folyamatábrák mellett egyre jobban elterjednek az úgynevezett struktogramok. A struktogramokat az olyan, magasabb szintű programnyelvek, mint a Pascal, Comal, Fortran 77, Elan stb. tették szükségessé, amelyek lehetővé teszik a strukturálást. A BASIC tulajdonképpen nem strukturált programozási nyelv, de a Plus/4-esben vannak olyan utasítások, amelyek a strukturálást lehetővé teszik.

9.6.2. A struktogramban használt szimbólumok

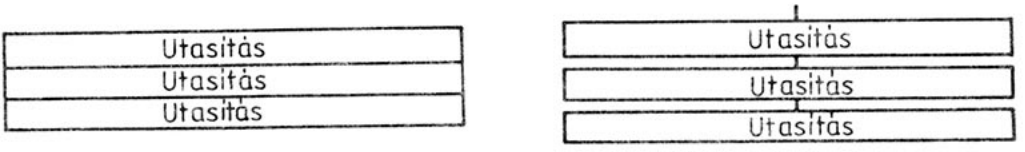
A program-folyamatábrákkal ellentétben a struktogramokban GOTO-parancs nem ábrázolható.

A struktogramokat Nassi–Shneiderman-diagramoknak vagy röviden NS-diagramoknak is nevezik. Most bemutatjuk az NS-diagramok néhány fontos struktúráját, feltüntetve mellettük a megfelelő folyamatábrát is.

Nassi–Shneiderman **Folyamatábra**

Szekvencia (sorozat)

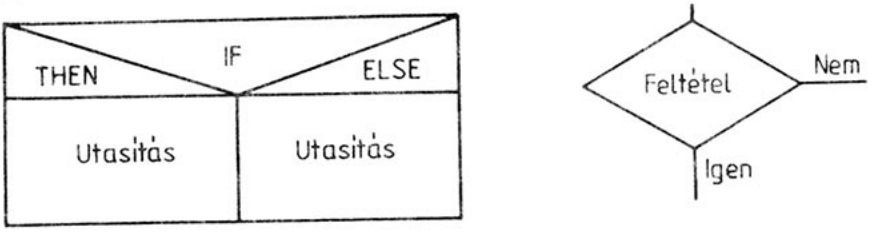
Több művelet:



9.13. ábra Több utasítás

Elágazás

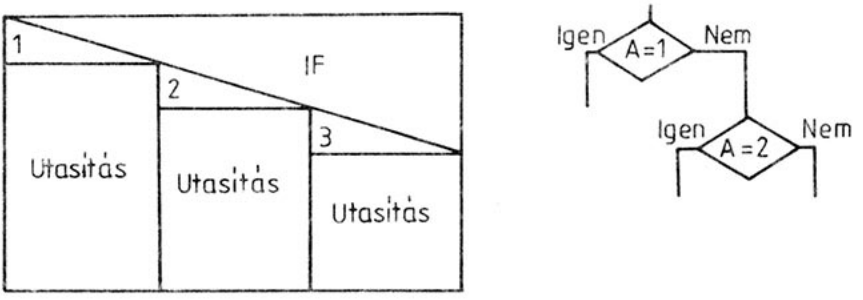
IF...THEN... vagy
IF...THEN...ELSE



9.14. ábra Elágazás

Többszörös elágazás

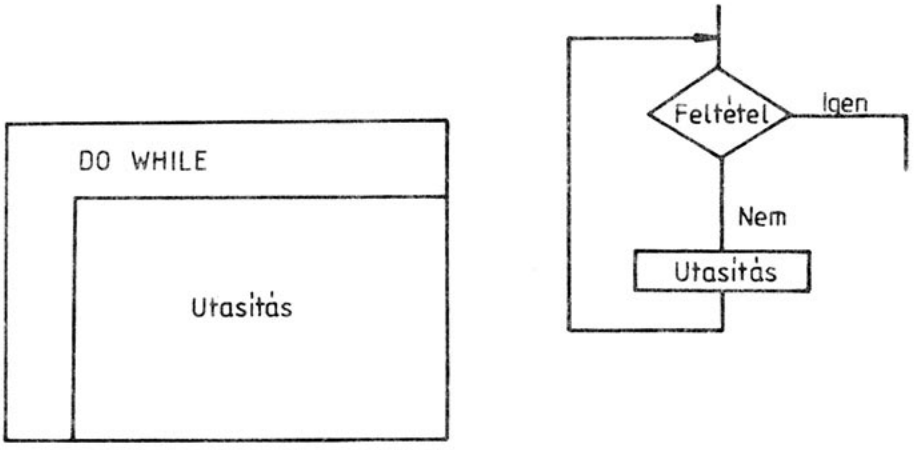
Pl. ON...GOSUB...



9.15. ábra Többszörös elágazás

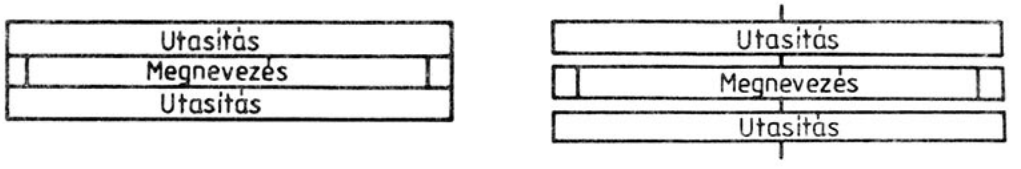
Ismétlődés

Egy program végrehajtása mindaddig ismétlődik, míg egy feltétel nem teljesül.



9.16. ábra Ismétlődés

Alprogram



9.17. ábra Alprogram

9.6.3. Példaprogram: KALENDÁRIUM

A következőkben bemutatjuk egy program tervezését és készítését. A programot folyamatábrával és struktogrammal, grafikusan is ábrázoljuk.

A megoldandó probléma: olyan programot kell írni, amelyik kiszámítja, hogy egy tetszőleges dátum a hét melyik napjára esett. A számításhoz a gregoriánus naptár szabályait használjuk: a néggyel osztható évszámok a szökőévek, ezek tehát 356 napból állnak. A 100-zal osztható évszámok nem szökőévek, viszont a 400-zal oszthatók azok.

Ennyit a szabályokról. Hogyan épüljön fel a programunk? Elegendő, ha a programunk három blokkból áll:

1. Dátum beírása
2. Számítás
3. Eredmény kiírása

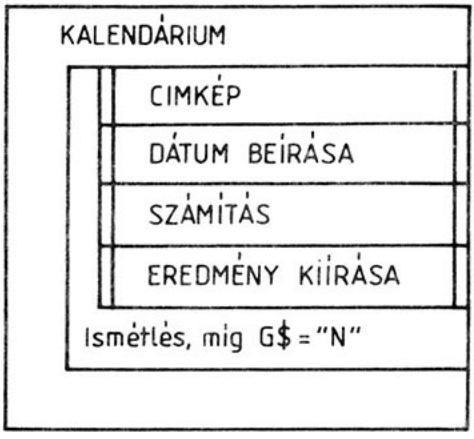
Az első blokkot tovább finomíthatjuk:

1.1. Képernyő törlése
1.2. Címkép kiírása
1.3. Dátum beírása

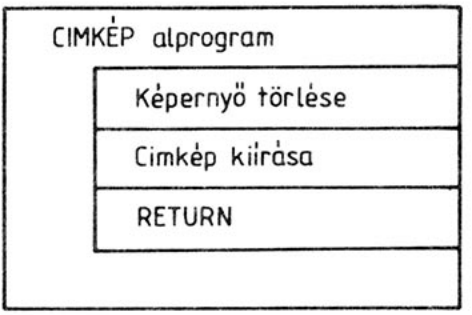
Az utolsó blokk ilyen lehet:

3.1. Képernyő törlése
3.2. A beírt dátum kiírása
3.3. A hét napjának kiírása
3.4. Kérdés arra vonatkozóan, hogy kell-e folytatni a számítást

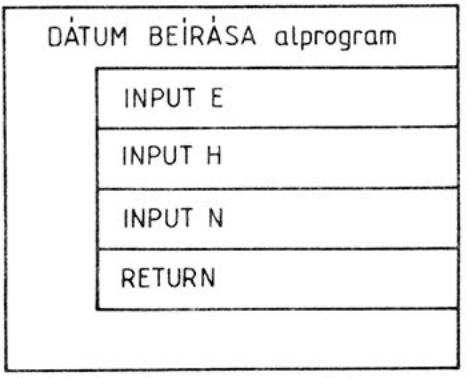
Nézzük akkor a folyamatábrát és a struktogramot a munka egyes fázisaiban:



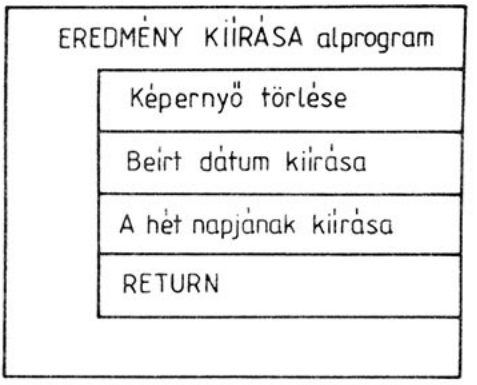
9.18. ábra A KALENDÁRIUM program struktogramja



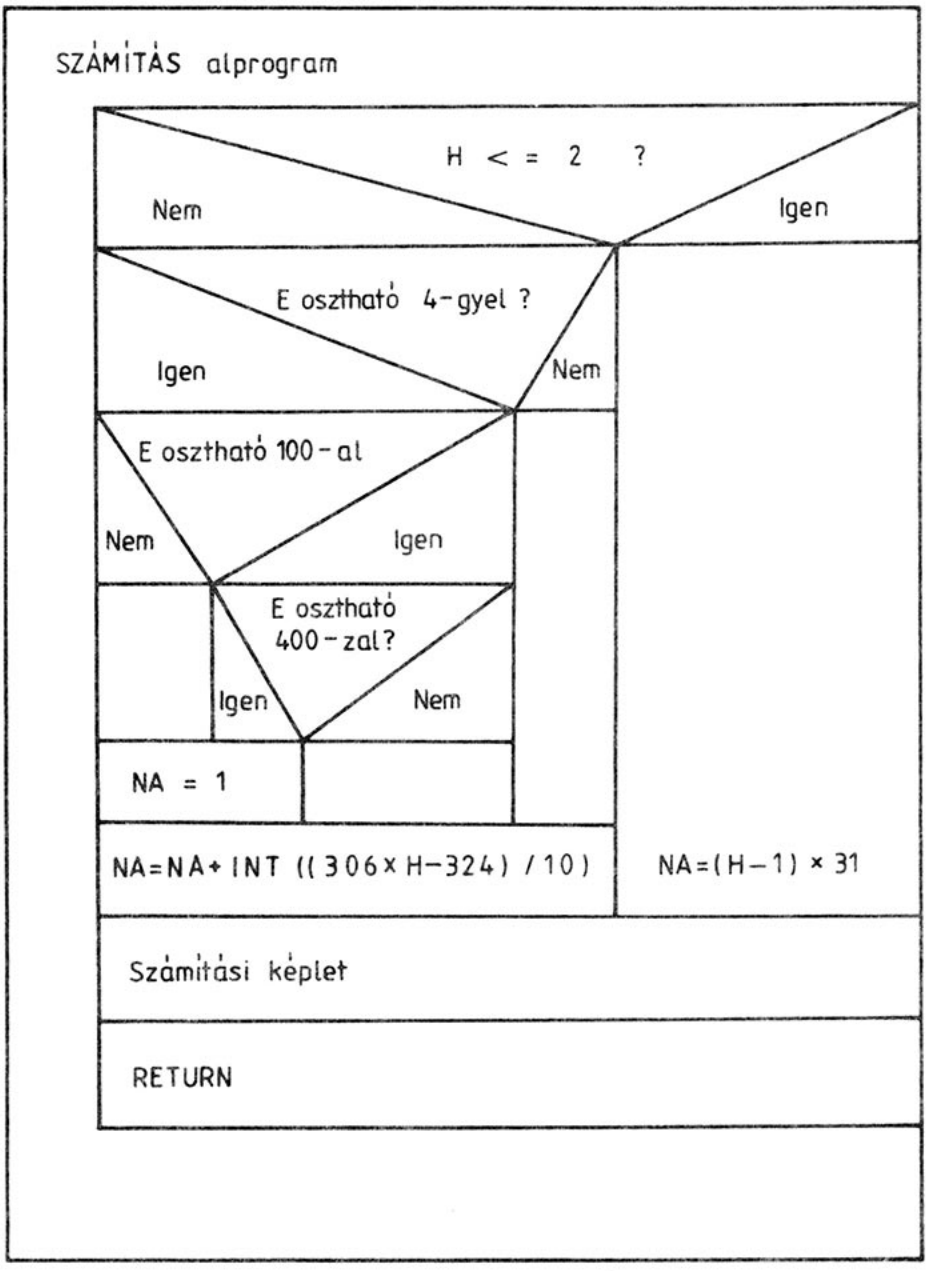
9.19. ábra A CÍMKÉP alprogram struktogramja



9.20. ábra A DÁTUM BEÍRÁSA alprogram struktogramja

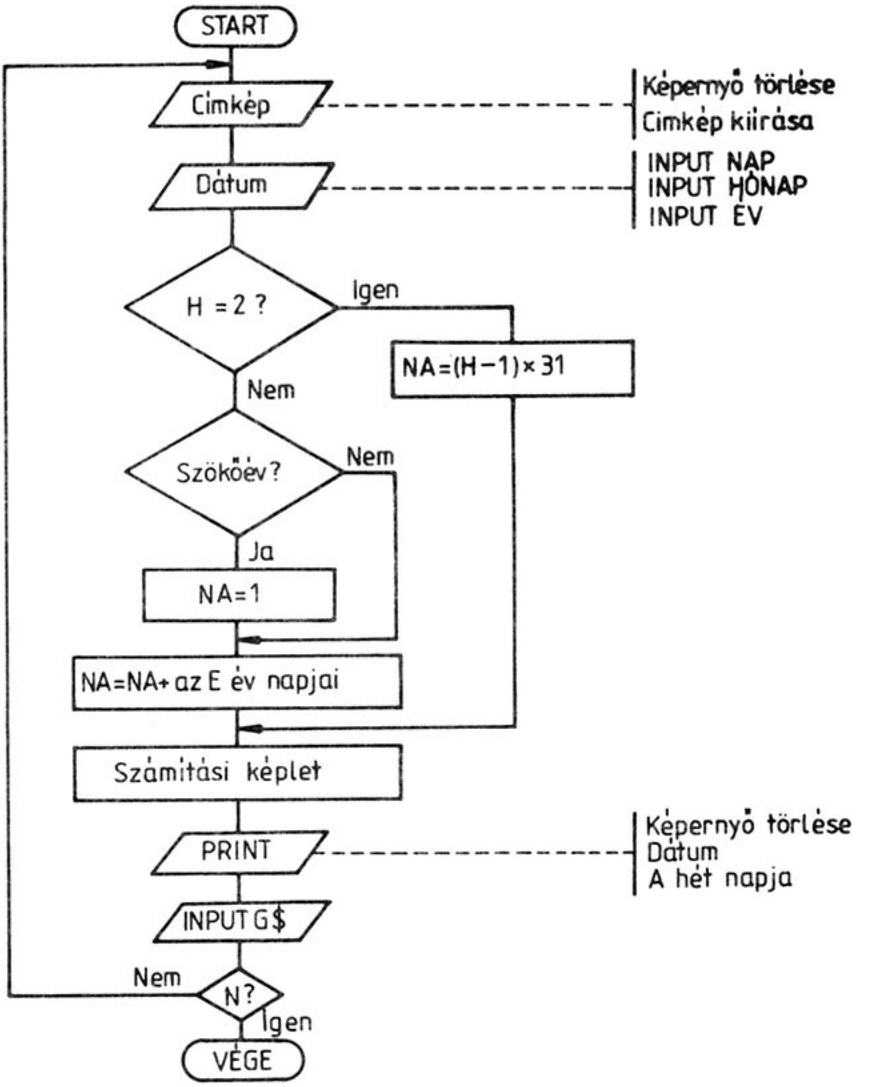


9.21. ábra Az EREDMÉNY KIÍRÁSA alprogram struktogramja



9.22. ábra A SZÁMÍTÁS alprogram struktogramja

Jól látható, hogy a struktogram esetében más módszert választottunk. Az egyes programblokkokat alprogramokként alakítottuk ki. Ezek az alprogramok mindaddig ismétlődnek, míg a program felhasználója a kérdésre „N"-nel nem válaszol. A struktogram esetén tehát van egy főrutin, amely az „igazgató" szerepét játssza, és meghívja az egyes alprogramokat.



9.23. ábra A KALENDÁRIUM program folyamatábrája

Mindegyik alprogramnak saját struktogramja van. Meg kell jegyeznünk, hogy a DÁTUM BEÍRÁSA alprogram struktogramja nem felel meg teljesen az előírt feltételeknek. Azt mondtuk, hogy (fordító megjegyzése: ez az állítás csak a programlistából derül ki) az 1583. év előtti dátumok nem számíthatók. Az erre vonatkozó kérdést tehát még be kell építeni. Ezenkívül volna még egy további alprogram, nevezetesen a hét napjainak beolvasása egy tömbváltozóba. Ezek elkészítését azonban már bizonyára rábízhatjuk az Olvasóra.

```
10 REM KALENDARIUM
20 FOR I=0TO6:READH$(I):NEXT:REM A HET NAPJAI
30 :
40 DO
50 GOSUB 200:REM CIMKEP
60 GOSUB 400:REM BEVITEL
70 GOSUB 500:REM SZAMOLAS
80 GOSUB 700:REM KIIRAS
90 PRINT:PRINT"UJABB SZAMITAS ? (I/N)";
100 GETKEY G$
110 IF G$="N"THEN EXIT
120 LOOP
130 END
200 :
210 REM CIMKEP
220 PRINTCHR$(147)
230 PRINT
240 PRINT" EGY MEGADOTT DATUM NAPJANAK KISZAMITASA"
250 PRINT
260 PRINT" IRJON BE EGY TETSZOLEGES DATUMOT !"
270 PRINT
280 PRINT" KISZAMITOM, HOGY A BEIRT DATUM A HET"
290 PRINT" MELYIK NAPJARA ESETT. KEREM, HOGY 1583"
300 PRINT" ELOTTI EVSZAMOT NE IRJON BE."
310 PRINT
320 PRINT
330 RETURN
400 :
410 REM BEVITEL
420 INPUT"EV";E
430 INPUT"HONAP";H
440 INPUT"NAP";N
450 IFE<1583THEN420
460 RETURN
500 :
510 REM SZAMOLAS
520 IF H<=2 THEN NA=(H-1)*31:GOTO560
530 IF E/4 = INT(E/4) THEN NA=1
540 IF E/100=INT(E/100)ANDE/400=INT(E/400)THEN NA=1:ELSENA=0
550 NA = NA+INT((306*H-324)/10)
560 NA = NA+(E-1)*365+INT((E-1)/4)
570 NA = NA-INT((E-1)/100)+INT((E-1)/400)
580 NA = NA+N
590 NA = NA-INT(NA/7)*7
600 RETURN
700 :
710 REM KIIRAS
720 PRINT CHR$(147)
730 PRINT"AZ";E;".";H;".";N;". NAPON ";H$(NA);" VOLT"
740 PRINT
750 RETURN
800 :
810 REM A HET NAPJAI
820 DATA VASARNAP,HETFO,KEDD,SZERDA
830 DATA CSUTORTOK,PENTEK,SZOMBAT
```

Nos, ha a programot hibátlanul beírtuk, akkor annak tulajdonképpen kifogástalanul kell működnie. Az 500-as sortól kezdődő programrész ugyanazokat a számításokat tartalmazza, amelyeket a struktogramban megadtunk. Az NA változóba először a 0. év és a beírt év közötti napok száma kerül, a szökőnapokat is beleértve. Itt az 1583-as évre is a Gergely-naptár szabályait alkalmaztuk. Mivel azonban ezeknek az éveknek a beírását a beíró rutinban megtiltottuk, a program nem végezhet hibás számítást. Természetesen kiszámíthattuk volna a beírt évszám és az 1583-as év közötti különbséget is, de ez nem szükséges. A program így is működik.

Még egy javaslat, hogy miként lehet a programot tovább javítani: a beíró rutinban csak azt vizsgáljuk, hogy a beírt évszám 1583-nál nagyobb-e. Mi történik azonban akkor, ha nem tartjuk be az egyéb szabályokat, és az 1986. 13. 45 dátumot írjuk be? A program azt válaszolja, hogy az 1986. 13. 45-e egy szombati nap. Ez persze képtelenség! A programba tehát egy olyan részt is be kell építeni, ami megvizsgálja a beírt hónapok és napok számát is. Úgy gondoljuk, hogy ezt rábízhatjuk az Olvasóra is.

10. Karakterek beírása

A gyakorlatban a programok többségénél szükség van arra, hogy a program használója valamilyen adatot – betűket, számokat – a program futása közben beírjon a gépbe. Enélkül egy program nem is lehetne rugalmas. Íme egy példa: írjunk egy programot, amellyel egy téglalap területét lehet kiszámítani. A program:

```
10 PRINT CHR$(147)
20 A=15
30 B=10
40 PRINT A*B
```

A program leírása:

10-es sor: Törli a képernyőt, és a kurzort a bal felső sarokba állítja.
20-as sor: Az A változóhoz a 15-öt rendeli.
30-as sor: A B változóhoz a 10-et rendeli.
40-es sor: Az A változót megszorozza a B változóval, és az eredményt kiírja a képernyőre.

A program lefutása után a képernyőn a 150-es szám jelenik meg.

Ha most egy A = 5 és B = 25 oldalhosszúságú téglalap területét akarnánk kiszámítani, akkor ehhez a programot módosítani kell. A programot először a LIST paranccsal a képernyőre kell íratni. Ezután a kurzorral a 20-as sorban az 5-ösre kell állni, és az INST/DEL billentyűvel az 1-est törölni kell. Még meg kell nyomni a RETURN billentyűt, hogy a számítógép a megváltoztatott sort vegye tudomásul. A 30-as sorban a 10-et át kell írni 25-re, és következik ismét a RETURN. Ezután már csak egy üres sorba kell állni a kurzorral, majd egy SHIFT-F3, és a képernyőn megjelenik az – ennyi idő alatt fejben már biztosan kiszámolt – eredmény, a 125. Hát ehhez igazán nem kell számítógép. A számítást a legegyszerűbb zsebszámológéppel is gyorsabban el lehet végezni. A programunknak az a baja, hogy kívülről nem adhatók át neki adatok. A most ismertetésre kerülő parancsok ezen a gondon segítenek.

10.1. AZ INPUT PARANCS

Szintaxis:

INPUT "szöveg"; változó; változó,...

A szöveg el is hagyható. A szintaxis ekkor:

INPUT változó, változó...

Ha a gép egy program végrehajtása közben ilyen utasítást talál, akkor megállítja a program futását, és a képernyőre kiírja – ha van – a szöveget, majd utána kitesz egy kérdőjelet és a kurzor is megjelenik. A program futása csak a RETURN billentyű lenyomása után folytatódik.

Változtassuk meg a programunkat a következők szerint, majd indítsuk el:

```
20 INPUT A
30 INPUT B
```

Írjunk be most az első kérdőjel mögé egy 4-est, és nyomjuk le a RETURN-t. Ekkor megjelenik egy második kérdőjel. Ehhez írjunk be 15-öt, majd ismét RETURN. A képernyőn most a 60-as szám jelenik meg.

Látható, hogy a programunk lényegesen rugalmasabb lett, csak az érthetőségével vannak bajok. Az, aki nem ismeri a programot – de néhány hét múlva talán mi sem tudnánk –, hogy a megjelenő kérdőjelhez mit is kell beírni. Ezen a problémán is könnyen lehet azonban segíteni, hiszen az INPUT parancsba beírható szöveg is. Változtassunk ismét a programon:

```
20 INPUT "AZ EGYIK OLDAL HOSSZA ";A
30 INPUT "A MASIK OLDAL HOSSZA ";B
```

A program működése nem igényel magyarázatot.

A parancs szintaxisából látható, hogy egy INPUT utasítással több változónak is lehet értéket adni. Ilyenkor az egyes változókat egymástól vesszővel kell elválasztani, és ugyanígy kell eljárni az értékek beírásakor is. Módosítsuk ismét a programot:

```
20 INPUT "AZ EGYIK ES A MASIK OLDAL HOSSZA ";A,B
```

A 30-as sort törölni kell!

A program indítása, és a szöveg megjelenése után írjuk be: 30, 40 majd RETURN. A képernyőn megjelenik az eredmény, azaz 1200. Amint látjuk, ez is használható. Ha két szám helyett csak egyet írnánk be, akkor a számítógép két kérdőjelet ír a képernyőre figyelmeztetésül, hogy valamit elfelejtettünk.

A hiányzó adatot most beírhatjuk.

Az INPUT utasítással azonban nem csak számok, hanem karakterláncok (betűket és számokat is tartalmazó, úgynevezett füzérek) is beírhatók. Ilyenkor a változó nevének a végére egy $ (dollár) jelet kell írni. Erről részletesen majd a karakterláncokról szóló fejezetben szólunk.

Az INPUT utasítás sajátosságai

Amikor a program egy adat beírására vár, a STOP billentyűvel nem állítható meg. Ilyenkor csak a RETURN billentyű segít, és ha a program folytatja a futást, használható ismét a STOP.

Ha az INPUT utasítás számot vár, és helyette betűt írunk be, akkor megjelenik a "?REDO FROM START" (kezdd elölről) hibaüzenet. A program futása nem szakad meg, hanem újra megjelenik az INPUT utasításba írt szöveg, és a program vár a helyes típusú adat beírására.

Ha az INPUT utasítás karakterláncot vár, akkor ez nem tartalmazhat vesszőt, kettőspontot, mivel ezek a változók egymástól való elválasztását végzik és így egy vesszőt is tartalmazó karakterláncot a gép két, önálló változónak értelmezne. Ha az

INPUT utasítás azonban csak egy változóhoz vár adatot, akkor megjelenik az "EXTRA IGNORED" (magyarul kb.: a felesleges figyelmen kívül marad) hibaüzenet. A program futása nem szakad meg, hanem a gép a vessző után álló karaktereket egyszerűen figyelmen kívül hagyja. (Ha mégis szükségünk van a szövegben vesszőre vagy kettőspontra, ill. szöveg végén, elején levő szóközre, akkor az egész szöveget idézőjelek közé kell tenni. Ebből következik, hogy idézőjelet sem fogad el a változó értékének.)

Arra az időre, amikor a kurzor látható, a képernyőszerkesztő is újra bekapcsol. Ez azt jelenti, hogy a kurzorvezérlő billentyűkkel a kurzort tetszés szerint lehet a képernyőn mozgatni. A hibásan beírt karaktert a szokásos módon, az INST/DEL billentyűvel lehet törölni. A képernyő a SHIFT CLEAR/HOME billentyűkkel törölhető is.

Ezek a sajátosságok azonban nem mindig kívánatosak. Elegendő lehet egy billentyű téves lenyomása, és ezzel elronthatjuk a szépen felépített képernyőt. Ezt az utasítást használva egy programozó nemigen tudja védeni a programját a hibás kezeléstől. Az sem mindig megoldás, hogy egy program állandóan külső adatra várjon, szükséges lehet, hogy bizonyos idő eltelte után folytassa a program végrehajtását (pl. játékprogramoknál). Ilyen meggondolásokból a Plus/4-es BASIC nyelve ismer még további, adatbeírási lehetőséget.

10.2. A GETKEY UTASÍTÁS

Szintaxis:

GETKEY változó,változó...

Ennek szintaxisa nem teszi lehetővé, hogy az utasításba szöveget írjunk. Ha szükség lenne valamilyen magyarázó szövegre, akkor ezt egy külön PRINT utasítással kell a képernyőre írni. Ha a gép a program végrehajtása során ilyen utasítást talál, akkor a következő történik: a program futása megáll, és a kurzor kikapcsolva marad. A program futása a SHIFT, CTRL és a C= billentyűk kivételével bármely billentyű lenyomására folytatódik. A lenyomott gomb kódja a változóba kerül.

Nézzünk egy programot, amely ezt mutatja be:

```
10 PRINT"VAROK EGY BILLENTYU LENYOMASARA"
20 GETKEY A$
30 PRINT"A LENYOMOTT BILLENTYU "A$" VOLT"
40 GOTO 10
```

Lehet, hogy a 30-as sor egy kicsit zavarosnak tűnik, de ne törődjünk vele. A PRINT utasítással majd a karakterek kiírásáról szóló fejezetben ismerkedünk meg.

Ha elindítjuk a programot, akkor az kiírja a képernyőre a 10-es sorban levő szöveget, és vár egy billentyű lenyomására. Ha most bármelyik billentyűt lenyomjuk (az előbb említettek kivételével), akkor az ehhez tartozó karakter megjelenik a képernyőn. A 40-es sorban arra utasítjuk a programot, hogy a futást a 10-es sorban folytassa, ami a jelen esetben azt jelenti, hogy kezdje elölről.

Ha most egyidejűleg lenyomjuk a SHIFT és a CLEAR/HOME billentyűket, akkor tapasztaljuk, hogy a képernyőszerkesztő most is működik. A látszat azonban csal. A képernyőszerkesztő a valóságban ki van kapcsolva! A képernyő törlését a

30-as sorban levő PRINT utasítás hajtja végre. Ugyanis a képernyőtörlést végző billentyűkombinációt nyomtuk le, és most ez került az A$ szöveg típusú változóba. A 30-as sorban arra utasítjuk a számítógépet, hogy a változóban levő tartalmat írja ki, és ez – tudva a kötelességét – meg is teszi. Ezt azonban meg is tilthatjuk. Egészítsük ki a programunkat az alábbi sorral, indítsuk el, és próbáljuk meg újra a képernyő törlését!

```
25 IF A$=CHR$(147) THEN GOTO 20
```

Most már láthatjuk az INPUT utasítással szembeni különbséget is. A GETKEY utasítás lehetővé teszi, hogy egy billentyűt „kikapcsolhassunk". Természetesen az előbb a CLEAR/HOME billentyűt igazában nem kapcsoltuk ki. A 25-ös sorba azt az utasítást írtuk, hogy abban az esetben, ha a változóba a képernyőtörlő billentyűkombináció kódja (147) érkezik, akkor a program ezt ne hajtsa végre, hanem menjen vissza az előző sorba, és várjon egy másik billentyű lenyomására. Hasonló módon zárhatjuk ki a kurzormozgató gombokat.

Bizonyára tapasztaltuk, hogy egy funkcióbillentyű lenyomása a program futását megszakítja, és hibaüzenetet ír ki. A probléma megoldása:

```
KEY 1,""
```

Ezzel a paranccsal töröltük az F1 billentyű tartalmát, és most ennek a billentyűnek a lenyomása nem „állíthatja fejre" a programunkat.

Térjünk vissza az előző programunkhoz, és írjuk át így:

```
20 GETKEY A$,B$
30 PRINT"A LENYOMOTT BILLENTYU "A$" ES "B$" VOLT"
```

A program indítása után megállapíthatjuk, hogy csak akkor folytatja a futását, ha a második billentyűt is lenyomtuk. Ekkor viszont mindkét karaktert kiírja.

A GETKEY utasítás tehát – hasonlóan mint az INPUT – lehetővé teszi, hogy egy programot megállítsunk azért, hogy a billentyűzetről adatokat írhassunk a programba. A GETKEY azonban csak annyi karaktert vesz át, ahány, egymástól vesszővel elválasztott változót tartalmaz. A RETURN billentyű lenyomására nincs szükség.

Ha a karaktert nem íratjuk ki a képernyőre, akkor ez a képernyő felépítését nem is ronthatja el. Ha viszont ki kell írni a karaktert – és a legtöbb esetben ez így is van –, akkor a nemkívánatos, vagy az adott programban nem megengedett billentyűket le lehet tiltani.

Most valaki közbevethetné: Mindez nagyon szép és jó, de én nem akarom, hogy a programom megálljon csak azért, hogy valamit bele lehessen írni! Erre is van megoldás:

10.3. A GET UTASÍTÁS

Szintaxis:

GET változó,változó...

Lássunk először egy rövid programot:

```
10 I=0
20 GET A$
```

```
30 PRINT A$
40 I=I+1
50 PRINT I
60 GOTO 20
```

Ha elindítjuk ezt a programot, akkor számok kezdenek futni a képernyőn alulról felfelé. A futást a COMMODORE billentyű lenyomásával lassíthatjuk. Ha lenyomunk egy másik billentyűt, akkor – a GETKEY utasításhoz hasonlóan – az ehhez tartozó karakter azonnal megjelenik a képernyőn. A program azonban nem áll le, hanem szünet nélkül fut tovább, függetlenül attól, hogy lenyomunk-e billentyűt vagy sem. Az F-billentyűk lenyomása sem állítja meg a programot.

A program leírása:

10-es sor: Az I változóhoz a 0 értéket rendeli. Ha a programot a RUN paranccsal indítjuk, akkor ez a sor tulajdonképpen fölösleges, de az áttekinthetőség kedvéért beírtuk a programba.

20-as sor: Ha lenyomunk egy billentyűt, vagy ha egy billentyű már korábban le lett nyomva, akkor ennek a kódja az AS változóba kerül.

30-as sor: Képernyőre írja az AS tartalmát, vagy végrehajtja a kódnak megfelelő műveletet.

40-es sor: Az I változó értékéhez 1-et ad.

50-es sor: Az I változóban levő értéket a képernyőre írja.

60-as sor: A program visszaugrik a 20-as sorra.

Lehet, hogy furcsának tűnik a 20-as sor leírása, de így igaz. A gép a billentyűzetet akkor is figyeli, ha a program pl. a 40-es sorban van, és ha ilyenkor nyomunk le egy billentyűt, a gép ezt is tárolja.

A bizonyítás:

```
10 REM
20 GET A$
30 PRINT A$
40 FOR I=0 TO 1500 : NEXT
50 GOTO
```

Indítsuk el a programot, majd nyomjunk meg gyors egymásutánban néhány billentyűt. Dőljünk hátra a karosszékben, és nézzük, mi történik.

Azt tapasztaljuk, hogy a beírt karakterek megjelennek a képernyőn, jóllehet egyetlen billentyű sincs lenyomva. Mivel a számítógépnek is megvannak a korlátai, a gép átmenetileg maximum 10 karaktert tárol. Ha ennél több billentyűt nyomtunk le, akkor a 10 fölöttieket figyelmen kívül hagyja.

Az itt leírtak vonatkoznak az INPUT és a GETKEY utasításokra is. Vizsgáljuk meg egy kicsit részletesebben magát a számítógépet, hogy megértsük, hogyan tudja a karaktereket átmenetileg tárolni.

Mint minden számítógépnek, a Plus/4-esnek is van egy, a programozó elől elzárt tárrésze, ahová a helyes működése szempontjából fontos értékeket helyezi el. Egyébként honnan tudná például azt, hogy a képernyő melyik pontján van a kurzor? Ennek a tárrésznek az egyik része az úgynevezett billentyűzet-puffer. Ennek a hossza 10 rekesz. E mellett van még egy rekesz, amely azt tudatja a számítógéppel, hogy az adott pillanatban éppen hány karakter van a billentyűzet-pufferben.

Ha a számítógép egy programot hajt végre, és most le kell nyomnunk egy billentyűt, akkor a karakter először a billentyűzet-pufferbe kerül, és a számláló rekesz tartalma 1 lesz. Amint már említettük, ez a szám maximum 10 lehet. Ha a gép most a programban egy billentyűzetet lekérdező utasításhoz – INPUT, GETKEY vagy GET – ér, akkor először megvizsgálja a számláló rekeszt. Ha az ebben levő érték 0, akkor az INPUT és a GETKEY utasítás esetén vár egy billentyű lenyomására. GET utasítás esetén a programot azonnal folytatja. Ha viszont a számláló tartalma nagyobb 0-nál, akkor a pufferből kivesz egy vagy több karaktert, és ez(eke)t átadja az utasításban megadott változó(k)nak. A számláló értéke a kivett karakterek számával csökken.

Ha a számítógép a program végrehajtása során nem talál ilyen beolvasó utasítást, akkor a pufferben levő karaktereket a program befejezése után közvetlen üzemmódban kiírja a képernyőre.

A későbbiekben, egy-egy probléma megoldása kapcsán biztosan előfordul, hogy azt akarjuk, a gép a program végrehajtása közben lenyomott billentyűket ne vegye figyelembe. A javasolt megoldásunk:

```
POKE 239,0
```

Ha ezt a sort beiktatjuk a programunkba, akkor a gép azokat a karaktereket, amelyeket a billentyűzet lekérdezése után írunk be, figyelmen kívül hagyja. Ez a parancs nullára állítja a számláló rekesz tartalmát, és így azt a látszatot kelti, mintha a billentyűzet-puffer üres lenne.

Vigyázat: Ez a parancs csak a GETKEY és az INPUT utasítással együtt használható, mert a gép a GET utasítást olyan gyorsan dolgozza fel (a program nem áll meg), hogy nincs elegendő idő karakter átadására.

Bizonyára feltűnt, hogy a GETKEY és a GET utasításokkal kapcsolatos példáinkban csak szöveg típusú változókat használtunk. Ha ugyanis numerikus változónak akartunk volna betűt átadni, akkor a program a "TYPE MISMATCH ERROR IN" hibaüzenettel leállna. Erre is van megoldás, példák a hibakereséssel és hibajavítással foglalkozó fejezetben találhatók. Jobb azonban, ha mindjárt az elején megszokjuk, hogy a hibalehetőségeket teljesen kizárjuk.

Az sem jelent problémát, ha a programba valahol numerikus értéket kell beírni. A Plus/4-essel egyszerűen megoldható karakterláncok numerikus változókká való átalakítása. Erről, valamint arról, hogy hogyan lehet egy beolvasó rutinnal egyszerű úton sok karaktert engedélyezni vagy letiltani, a karakterláncokkal foglalkozó fejezetben olvashatunk részletesen.

Ha e fejezet tanulmányozása után saját beolvasó rutinokat programozunk, hamarosan tapasztalni fogjuk, hogy a legjobban a GETKEY és a GET utsítások kezelhetők. Ezeknél ugyan nagyobb a programozói munka mint az INPUT-nál, viszont a nemkívánatos karakterek kiszűrhetők, és a program használójának nem kell azon gondolkodnia, hogy éppen egy számot vagy egy betűt kell-e beírnia.

11. Írás a képernyőre

Egy program használhatóságát és kezelhetőségét jelentős mértékben befolyásolja, hogy a program az általa kiszámított eredményeket milyen formában írja ki a képernyőre. Egy program felhasználójának nagyon fontos, hogy mindig tudja, hogy a programban mikor és mit kell tennie, valamint az is, hogy a számítások eredményei a képernyőn könnyen áttekinthetők legyenek. Nézzünk először egy példát:

```
20 INPUT "IRJON BE EGY SZAMOT";SZ
30 PRINT SZ*2
40 PRINT SZ*5
```

Indítsuk el a programot, és a felszólításra írjuk be a 2-es számot. Egy RETURN után egymás alatt megjelenik az eredmény, egy 4-es és egy 10-es. Az eredmény ugyan stimmel, de hogy néz ki a képernyő! Először is ott van még az éppen megírt program, alatta a felszólítás a szám beírására, majd következnek az eredmények. Hát szépnek éppen nem mondható. Kezdjünk hozzá a rendrakáshoz!

11.1. A PRINT PARANCS

Szintaxis:

PRINT "szöveg" vezérlő karakter

vagy

PRINT változó vezérlő karakter

vagy a kettő kombinációja.

A PRINT paranccsal karaktereket írhatunk a képernyőre. A kiírás formáját határozzák meg a vezérlő karakterek.

Ezek a vezérlő karakterek a , (vessző) és a ; (pontosvessző).

Magyarázatul álljon itt egy program. Beírása előtt írjuk be: NEW.

```
10 PRINT "A"
20 PRINT "PLUS/4"
30 PRINT "SZAMITOGEP"
```

Ha a programot elindítjuk, akkor az egyes szavak egymás alatt jelennek meg a képernyőn. Amikor a program egy PRINT utasítással találkozik, akkor az idézőjelek között található karaktereket úgy értelmezi, hogy azok egy változó nevét alkotják, és most a változóban levő értéket írja ki.

Ha a PRINT utasítás után nincs vezérlő karakter, akkor az ezután következő PRINT utasítás egy sort emel, azaz a kurzort a következő sor első oszlopába viszi. Ha már a képernyő legalsó sorába írunk, akkor a következő PRINT hatására a képernyő tartalma egy sorral felfelé gördül (scroll), és a legelső sor kifut a képernyőről. A Plus/4-esnél van lehetőség arra, hogy ezt a görgetést letiltsuk. Erről az ESCAPE műveletekkel foglalkozó fejezetben írtunk.

Gyakran van szükség arra, hogy egy sorba több kiírás kerüljön. Erre a célra szolgálnak a vezérlő karakterek.

11.1.1. A vessző szerepe a PRINT utasításban

Bár a kurzor egy program végrehajtása közben a képernyőn nem látszik, mégis vezérelhető a programmal. A Plus/4-es gép esetén a képernyő 40 oszlopra és 25 sorra osztható. Gondolatban osszuk fel a képernyőt négy azonos nagyságú felületre úgy, hogy egy-egy felület ekkor 10 oszlopból és 25 sorból áll. Ha a kiírandó szöveg vagy változó után egy vessző áll, akkor az ezután következő PRINT utasítás a kiírást az illető sorban, a képernyő soron következő negyedében kezdi. Például:

NEW (csak akkor szükséges, ha az előző, vagy valamilyen más program van a tárban)

```
10 PRINT"1","2","3","4"
```

Látható, hogy a számok egy sorba íródnak ki, és kilenc üres karakter van közöttük. Ha azt akarjuk, hogy csak a 3-as és a 4-es szám íródjon ki a harmadik ill. a negyedik negyed elejére, akkor ezt kell írni:

```
10 PRINT, ,"3","4"
```

Jegyezzük meg: ha egy PRINT utasítás végén vessző áll, akkor a kurzor a képernyő aktuális sorának következő negyedére áll.

Ha a PRINT utasítás végén több vessző van, akkor a kurzort mindegyik vessző 10-10 hellyel viszi jobbra. Nézzük meg ezt:

```
10 PRINT, ,"3", , ,"2"
```

Amikor a számítógép ezzel a programsorral találkozik, akkor tudja, hogy a PRINT utáni karaktereket ki kell írnia. Mivel az első karakter egy vessző, a kurzort 10 oszloppal jobbra állítja. A következő karakter ugyancsak vessző, tehát újabb 10 oszloppal jobbra állítja. Most találja meg az első idézőjelet, ami azt jelenti számára, hogy az ezután következő karaktereket a következő idézőjelig bezárólag a képernyőre kell írnia. Ebben az esetben ez csak egy karakter, és ezt ki is írja. Ezután ismét vesszők következnek. Az első a kurzort a 30-as oszlopba állítja. Mivel a Plus/4-es egy sorba csak 40 karaktert tud írni, a második vessző hatására a kurzor a következő képernyősor első oszlopába ugrik. A következő vessző a kurzort további 10 oszloppal jobbra állítja, és végül kiíródik a 2-es. Ezért a program végrehajtása után a 3-as a képernyő harmadik negyedében, a 2-es pedig a következő sor második negyedében jelenik meg. Ez a vezérlő karakter jól használható táblázatok készítésénél.

Lássunk erre is egy példát:

```
10 INPUT "A SZAM"; SZ
20 PRINT
30 PRINT "A SZAm 2-SZERESE=",2*SZ
```

```
40 PRINT
50 GOTO 10
```

A program magyarázata:

10-es sor: Képernyőre írja az "A SZAM" szöveget, és vár egy numerikus értékre.
20-as sor: Mivel a PRINT utasításban sem szöveg, sem változó, sem pedig vezérlő karakter sincs, a képernyőre egy üres sor íródik ki.
30-as sor: Kiíródik az "A SZAM 2-SZERESE =" szöveg, és a vessző hatására a kurzor a képernyő következő negyedébe ugrik. A gép kiszámítja a 2 * SZ értéket, és az eredményt a képernyőre írja.
40-es sor: Újabb üres sor következik.
80-as sor: Ugrás vissza a 10-es sorra.

A 30-as sorból látszik, hogy az eredmény kiírásához nem kellett külön PRINT-et írni. Ez mindig érvényes, ha az utasításban karakterláncok, változók és ezek kombinációi követik egymást, és ezeket csak üres karakter, vessző vagy pontosvessző választja el egymástól.

Ez a program csak úgy állítható meg, ha egy RETURN után nagyon gyorsan lenyomjuk a STOP billentyűt. Tegyük ezt egyszerűbbé:

```
50 PRINT "AKAR BEIRNI EGY MASIK SZAMOT ? (I)"
60 GETKEY A$
70 IF A$ <>"I" THEN END
```

E sorok magyarázata a logikai műveletekkel és a karakterek beírásával foglalkozó fejezetekben található meg.

Végül lássunk egy programot, amely ennek a vezérlő karakternek a szerepét demonstrálja:

```
10 REM DEMO A "," MINT VEZERLO KARAKTER     A PRINT UTASITASOKBAN
20 PRINT CHR$(147);REM KEPERNYO TORLESE
30 PRINT "AZ 1 ES A 40 KOZOTTI SZAMOK"
40 PRINT : REM URES SOR BEIRASA
50 PRINT "PARATLAN SZAMOK","PAROS SZAMOK"
60 FOR I=1 TO 40 STEP 2
70 PRINT I,,I+1
80 NEXT I
90 GETKEY A%
```

Ez a program a 60-as és a 80-as sorok kivételével – ezeket a programciklusokról szóló fejezet ismerteti – már magyarázat nélkül is bizonyára érthető. Ha valami nem világos, akkor lapozzunk vissza, és elevenítsük fel a tanultakat.

11.1.2. A pontosvessző szerepe a PRINT utasításban

Előbb-utóbb minden programozónak szüksége lehet arra, hogy a programjába hosszabb szövegeket, pl. kezelési útmutatót írjon. Ha egy ilyen szöveg hosszabb mint egy programsor, akkor a PRINT utasításban a pontosvessző (;) használata nélkül nem

boldogulnánk. Ez teszi lehetővé, hogy összefüggő szövegeket írhassunk a képernyőre akkor is, ha a szöveg egyes részei más-más programsorokban vannak. Változók tartalma is hézagmentesen beilleszthető egy szövegbe, úgy hogy folyamatos írásképet kapunk. Ennek szemléltetésére nézzük ismét a fejezet elején levő programot, és egy kis változtatással írjuk be újra.

```
10 PRINT "A";
20 PRINT "PLUS/4";
30 PRINT "SZAMITOGEP"
```

RUN után a szavak közvetlenül egymás után jelennek meg a képernyőn. Ahhoz, hogy a szöveg olvasható legyen, az "A" és a "Plus/4" szavak mögé tehát még egy szóközt (space) kell írni. Ha ezt elvégezzük, akkor a képernyőn megjelenő kiírás már könnyen olvasható. Ha a kiírandó szöveg 1 képernyősornál (40 karakter) hosszabb, akkor a kiírás a következő sor elején folytatódik.

Ha egy szövegen belül numerikus változót kell kiírni, akkor azt figyelembe kell venni, hogy a számítógép egy szám elé és mögé automatikusan kitesz egy üres karaktert. Változtassuk meg az 5-ös és a 20-as sort a következőképpen:

```
5 A=4
20 PRINT "PLUS/";A;
```

Látható, hogy ugyan sem a "PLUS/" mögé, sem a "SZAMITOGEP" elé nem írtunk üres karaktert, a gép a szöveget mégis üres karakterekkel írja ki. Ez azonban, amint mondtuk, csak numerikus változókra vonatkozik. A szöveg típusú változók szóköz kihagyása nélkül, közvetlenül a szöveg folytatásaként kerülnek a kiírásba. Nézzük meg ezt:

```
5 A$="4"
20 PRINT "PLUS/";A$;
```

A PLUS/ és az A$ közötti pontosvessző el is hagyható, a kiírás anélkül is folyamatos lesz. Aki nem hiszi, próbálja ki.

Ha egy programmal hosszabb szövegeket akarunk a képernyőre kiírni, akkor a szövegeket előbb meg kell szerkeszteni, ami nem mindig egyszerű feladat. A programsorokban levő sorszámok és a PRINT utasítások a programlistában „elrontják" a szöveg formáját, és megnehezítik, hogy lássuk, pontosan milyen formában is jelenik meg majd a képernyőn. Az első programfuttatás után általában még be kell iktatni üres karaktereket, vagy szavakat át kell helyezni más programsorba, hogy a szöveg a megjelenésekor jól olvasható legyen. Egy szó vagy egy mondat utólagos törlését vagy beszúrását csak nehézkesen lehet megvalósítani.

Az itt következő program egy speciális beviteli és kiírási eljárással segít ezen a gondon. A szöveget DATA sorokban kell elhelyezni. A program mindig egy DATA sort olvas el, és ebben megkeresi az üres karaktereket (szóközöket). Ha talált egyet, akkor megvizsgálja, hogy a szövegnek az a része, amelyik az üres karakterig tart, elfér-e az aktuális képernyősorban. Ha elfér, akkor ezt a pozíciót tárolja, és folytatja a keresést. Ezt mindaddig folytatja, míg vagy a szöveg végére nem ért, – ebben az esetben a még ki nem írt szöveget átmenetileg tárolja – vagy pedig az átnézett szöveg már hosszabb egy képernyősornál. Ha hosszabb, akkor az előző szóközig terjedő szöveget kiírja a képernyőre, a tárolt aktuális szövegből a kiírt karaktereket elhagyja, és folytatja a szövegben az üres karakterek keresését.

Ha a program a " – 1" karakterrel találkozik, akkor az átmenetileg tárolt szöveget képernyőre írja, és a program futása befejeződik. Igen könnyen lehet új bekezdéseket is szerkeszteni, vagy üres sorokat beírni.

A program használatához:

Ha ezt a programot alprogramként használjuk, akkor a 30-as sorban levő END utasítás helyett RETURN-t kell írni, hogy a végrehajtása után a vezérlés visszaadódjon a főprogramnak.

A program az új bekezdéseket és az üres sorokat üres karakterláncok segítségével állítja elő. Az első üres karakterlánc megtalálása után az ezt követő karakterek a következő sorban íródnak ki. Ha egy üres karakterláncot egy újabb, üres karakterlánc követ, akkor a program egy üres sort ír ki. A DATA sorokban levő szöveg teljes programsor hosszúságú lehet (88 karakter), és bármely ábrázolható karaktert tartalmazhatja. (Ez azzal a megszorítással igaz, hogy a szövegen belül idézőjel nem használható, helyette az aposztróf ajánlható. Ford. megj.)

A program a tárban a RENUMBER paranccsal eltolható. Ha egy programban több helyen is szerepelnek kiírandó szövegek, akkor a DATA mutatóját előzőleg a RESTORE paranccsal a kiírandó szövegrész első programsorára kell állítani. Mindegyik szövegblokkot " – 1"-gyel kell lezárni. Ügyeljünk arra, hogy mindegyik DATA sor üres karakterrel (szóköz) fejeződjön be. Ellenkező esetben a program hibásan működne.

A programozáshoz:

A REM sorokat, illetve a csak egy kettőspontot tartalmazó sorokat nem kell feltétlenül beírni. A jobb áttekinthetőség és a későbbi dokumentálás érdekében ez a kis többletmunka azonban megéri a fáradságot. A DATA sorok csak szemléltetési célokat szolgálnak. Később törölhetők, és beírhatók helyükre a saját programjainkban levő szövegek.

A programlista:

```
1 REM ***** SZOVEGKIIRO PROGRAM *****
2 :
3 REM *** A FELHASZNALT VALTOZOK: ***
4 REM A$= EBBEN VAN A BEOLVASOTT SZOVEG
5 REM B$= ATMENETILEG TAROLJA A KIIRANDO SORT
6 REM P0= A KOVETKEZO SZOKOZ POZICIOJA
7 REM X0= AZ UTOLSO SZOKOZ POZICIOJA
8 :
9 :
10 PRINT CHR$(147) : REM KEPERNYO TORLESE
20 READ A$          : REM SZOVEG BEOLVASASA
25 REM ALPROGRAM ESETEN END HELYETT RETURN
30 IF A$="-1" THEN PRINT B$:END
40 IF A$="" THEN PRINT B$:B$="":GOTO 20
50 P0=0
60 DO UNTIL P0+LEN(B$)>41
70 X0=P0
```

```
80 PO=INSTR(A$," ",PO+1)
90 IF PO=0 THEN EXIT
100 LOOP
110 IF PO=0 AND LEN(A$)>0 THEN XO=40-LEN(B$)
120 B$=B$+LEFT$(A$,XO)
130 A$=MID$(A$,XO+1,LEN(A$))
140 IF PO=0 AND LEN(A$)=0 THEN GOTO 20
150 PRINT B$;B$=""
160 GOTO 50
500 DATA "EZZEL A PROGRAMMAL KENYELMESEN KIIRHATOK SZOVEGEK "
510 DATA "A KEPERNYORE. LATHATO, HOGY "
520 DATA "A KIIRT SZAVAK NINCSENEK ELVALASZTVA VAGY RESZEKRE
    TORDELVE. "
530 DATA "UJ ","","BEKEZDESEK "
540 DATA "ES ","","","URES SOROK IS KONNYEN KESZITHETOK. "
700 DATA "-1"
```

11.2. A PRINT USING UTASÍTÁS

Szintaxis:

PRINT USING „formátum";változó,szöveg,változó,...

Ezzel az utasítással a képernyőre kerülő kiírás formátumát határozhatjuk meg.

Amint a szintaxisból látszik, az utasításban idézőjelek között meg kell adni a formátumot. Ezután egy pontosvessző következik, majd a kiírandó változókat kell felsorolni (a változók numerikus és karakter típusúak lehetnek). Az utasítással idézőjelek közé írt konstans szövegek is kiírathatók.

A formátum meghatározásához különböző karakterek használhatók. Van egy karakter, amellyel szám is és szöveg is meghatározható. Más karakterekkel viszont vagy csak számokat, vagy csak szöveget jelölhetünk. Nézzük először azt a karaktert, amely számokhoz és szöveghez egyaránt használható.

11.2.1. A # jel a PRINT USING utasításban, számok kiírásánál

Mielőtt folytatnánk az olvasást, írjuk be a következő kis programot:

```
20 INPUT "A FORMATUM";FO$
30 INPUT "A SZAM";A
40 IF A= -1 THEN END
60 GOTO 20
```

Amint az 50-es sorban látjuk, a formátumra vonatkozó utasítás karaktervváltozóban is megadható (itt FO$). A programot bármikor megállíthatjuk, ha az „A SZAM" kérdésre −1 beírásával válaszolunk.

Indítsuk el a programot. A formátum kérdésre írjunk be négy # jelet (####), a szám kérdésre pedig az 1234 számot. A következő képernyősorban, balra igazítva megjelenik az 1234. Most a formátum kérdésre csak a RETURN lenyomásával

válaszoljunk, a számra pedig az 123 számot írjuk be. Folytassuk ezt addig, míg utoljára az 1-es számot be nem írtuk.

Ebből már világosan látszik a PRINT utasítással szembeni eltérés. Míg a PRINT utasítás a szövegek és a számok kiírását – amennyiben vezérlő karakterekkel nem adtunk meg mást – mindig a sor elején kezdi, addig a PRINT USING a számokat „helyiértékre igazítva" írja ki, azaz egymás alá kerülnek az 1-esek, 10-esek stb.

Tartsuk meg az előző formátumot, és írjuk be az 12345 számot. A képernyőn négy csillag jelenik meg, amelyik mindegyike egy, a formátum mezőben levő # jelnek felel meg. A formátum mezőnek annyi # jelet kell tartalmaznia, ahány számjegyből áll a szám. Ha a szám ennél hosszabb, akkor a képernyőn annyi csillag jelenik meg, ahány # jelet tartalmaz a formátum. Arról tehát a programozónak kell gondoskodnia, hogy a programnak ne kelljen több számjegyet kiírnia, mint ahány # jel van a formátum mezőben. Írjunk be most változatlan formátum mellett egy tizedestörtet, pl. az 123.5 számot. Az eredmény 124 lesz. A számítógép a számot kiírás előtt tehát felkerekítette. Írjuk be most a 123.4 számot. A számítógép ezt lefelé kerekíti. Az utasításnak ez a szolgáltatása sok aprómunkától kíméli meg a programozót.

Figyelem: A Plus/4-es maximum 9 jegyből álló számokat tud feldolgozni. Adjunk meg a formátum mezőben pl. 11 # jelet, és írjunk be egy 11 jegyből álló számot. Az utolsó számjegyek 0-ként jelennek meg. Azt, hogy a PRINT USING segítségével hogyan lehet nagyobb számokat is ábrázolni, majd a hatvány jel ismertetésénél mutatjuk be.

11.2.2. A tizedespont a PRINT USING utasításban

Változtassuk meg a formátumot:

###.## (három # jel, tizedespont, kettő # jel)

A szám legyen

123.345

Az eredmény:

123.35

A tizedespont után tehát több számjegy is állhat, mint ahány # jelet a formátumban megadtunk. A gép ebben az esetben is elvégzi a fel-, ill. a lekerekítést. Ha a tizedespont után nincsenek számjegyek, akkor annyi 0 íródik ki, ahány # jelet megadtunk. Írjunk be most különböző számokat. Látható, hogy a tizedespontok mindig egymás alá kerülnek. Mi történik azonban negatív szám esetén? Próbáljuk ki, változatlan formátum mellett.

A −220.14 beírásakor hat csillag jelenik meg a képernyőn, ebből három az egész résznek, egy a tizedespontnak, kettő pedig a tört résznek felel meg.

Ha a −22.14 számot írjuk be, akkor ez a képernyőre kerül. A mínusz jel kiírásához tehát kell egy # jel.

11.2.3. Plusz és mínusz jel a PRINT USING utasításban

Írjuk be a −###.## formátumot (mínusz jel, három # jel, tizedespont, kettő # jel) és az 123.45 számot. A szám a következő sorban, a második oszloptól kezdődően íródik ki. Egy karakternyi hely tehát üresen maradt. Ez a formátum mezőben megadott mínusz jel számára van lefoglalva. Ha most ugyanis a −123.45 számot írjuk be, akkor a sorban az első pozíción megjelenik a mínusz jel.

Hasonló a helyzet a plusz jelnél is. Ha a formátum +###.##, és a szám 123, akkor a képernyőre +123.00 íródik ki. Minden pozitív szám előtt megjelenik a plusz jel, függetlenül attól, hogy azt a szám beírásakor beírtuk vagy sem. Természetesen ha pl. +123-at írunk be, az utasítás akkor is csak +123.00-t ír ki.

Negatív számok is ábrázolhatók ebben a formában. Például a −123 kiírási képe −123.00 lesz. A plusz és a mínusz jel a formátum végén is állhat. Például a ###.##+ formátum mellett a 123 megjelenési formája 123.00+ lesz, a −123 számé pedig 123.00−. Kiíráskor a plusz vagy a mínusz jel tehát a szám végére kerül.

Vigyázat: a −###.## − formátum megadása nem megengedett! Ha a számítógép ilyen formátumot talál, akkor a program a "SYNTAX ERROR IN..." hibaüzenettel leáll.

11.2.4. A hatvány jel a PRINT USING utasításban

Ha a számítógép a PRINT USING utasításban négy hatvány jelet (↑↑↑↑) talál, akkor az ez után következő számot úgynevezett normál alakban írja ki. Azok számára, akik esetleg nem ismerik ezt az ábrázolásmódot, álljon itt egy rövid magyarázat.

A tízes számrendszerben a számok különböző módon ábrázolhatók. Az 1000-es szám írható például 1*10↑3 formában, ami úgy olvasandó, hogy egyszer tíz a harmadikon. Ebben az írásmódban a 3-as a kitevő, a 10 az alap. A 10↑3, azaz tíz a harmadikon 10*10*10 alakban is írható. Ennek a szorzásnak az eredménye 1000. Hogyan lehetne a 2000-et hatvány alakban kifejezni?

Ha a 2000-et elosztjuk 10-zel, akkor az eredmény 200 lesz. Ha még kétszer osztjuk 10-zel, akkor 2-t kapunk. Mivel összesen háromszor osztottunk 10-zel, a 10-es alapon a kitevő 3 lesz, és eredményként a 2*10↑3 formát kapjuk.

A 2300 írható 2.3*10↑3, a 2340 pedig 2.34*10↑3 alakban. Ez utóbbi számot a Plus/4-es 2.34E+3 formában írja ki. Az E betű itt a 10-es számot jelöli.

Ez az egész meg is fordítható. Így a 0.003 normál alakban 3*10↑−3, azaz háromszor tíz a mínusz harmadikon lesz. A mínusz jel itt azt jelenti, hogy a hármat egymás után háromszor el kell osztani 10-zel ahhoz, hogy a normál alakban ábrázolt számnak megkapjuk a tízes számrendszerbeli alakját. A Plus/4-es ezt a számot 3E−03 alakban írja ki. Ez az ábrázolásmód azonban csak nagyon nagy, vagy nagyon kicsi számoknál célszerű, mert ezeknél az eredmény általában nem az utolsó számjegy értékén múlik.

Térjünk vissza a PRINT USING utasításhoz. Legyen most a formátum #↑↑↑↑, a szám pedig 1000.22. Az eredmény 1E+03. Megállapíthatjuk, hogy a gép a számot automatikusan kerekítette, hiszen a 0.22-t nem írta ki. Megjegyezzük, hogy a kerekítés csak a kiírásra vonatkozik, a változóban továbbra is az a szám van, amit beleírtunk.

A 0.5 kiírási képe 5E−01. Ebben a formátumban az E betű elé egy, mögéje pedig maximum három karakter (előjel és két szám) írható.

Ebben a formában ábrázolhatók azok a legnagyobb, ill. legkisebb számok, amelyeket a számítógép még fel tud dolgozni. Ezek a számok azonban nem pontosak. A Plus/4-es BASIC értelmezője az 1E+38 és az 1E−38 közötti számokat tudja feldolgozni. Nagyobb szám esetén a gép az ?OVERFLOW ERROR (túlcsordulás) hibaüzenetet írja ki, a kisebb számokat pedig nullának tekinti. A kitevő – csakúgy, mint a PRINT USING esetében – az előjellel együtt kerül kiírásra.

Készítsünk egy másik formátumot: ###.##↑↑↑↑. Írjuk be az 12345.6. számot. Az eredmény 123.46E+02. Ha ezt a számot megszorozzuk 10↑2 = 100-zal, akkor az eredmény 12346 lesz. Amint látjuk, a számítógép most is kerekített. Az ezzel kapcsolatban korábban elmondottak azonban most is érvényesek.

Figyelem: Az az állítás, hogy a négy hatvány jel a # jelek elé is írható, nem felel meg a valóságnak! Hibajelzést ugyan nem kapunk, de a kiírás meglehetősen furcsa képet ad.

11.2.5. A dollár jel a PRINT USING utasításban

Ha a dollár jel a formátum mezőben szerepel, akkor a képernyőn is megjelenik. Ezzel a jellel nem csak a dollárt, mint fizetési eszközt lehet jelölni, hanem a hexadecimális számok is egyszerűen, a kívánt formában megjeleníthetők a képernyőn.

Válasszuk most a $#### formátumot, és írjuk be a 12-es számot. A dollár jel a következő sor első pozícióján, a 12 pedig a negyedik és ötödik pozíción jelenik meg. Legyen most a formátumunk ##$##, a szám pedig ismét a 12. Az eredmény most más lesz. Az első két pozíció üres, ezután következik a dollár jel, majd a 12.

Jegyezzük meg: ha a formátum mezőben a dollár jel az első # jel előtt áll, akkor a képernyőn is a szám elé kerül úgy, hogy a dollár jel és a tizedespont közötti távolság annyi karakterpozíció lesz, ahány # jelet tartalmaz a formátum. Ha viszont a dollár jel valahol a # jelek és a tizedespont között van, akkor a képernyőn közvetlenül a szám elé kerül.

11.2.6. A vessző a PRINT USING utasításban

A formátum mezőben megadott vessző azt jelöli, hogy a képernyőn megjelenő számban hol álljon a vessző. A vessző akkor is megjelenik a képernyőn, ha a formátum mező első helyén áll.

A most következő példák az előző programunkkal nem mutathatók be, mert mint ismeretes, az INPUT utasításra beírt karakterlánc nem tartalmazhat vesszőt. (Természetesen megkerülhető a probléma, idézőjelek közé téve beírható: "##,##" és így elfogadja.) Ezért közvetlen módban írjuk be:

```
PRINT USING "##,##";1234
```

Az eredmény: 12,34

Most ezt írjuk be:

```
PRINT USING",##.##.##.##" ;1234.5678
```

A képernyőn megjelenő kiírás: .12,34.56,78

A második példából látszik, hogy a vesszőt és a tizedespontot nem szabad egymással összekeverni. A kerekítésekre továbbra is az előzőekben elmondottak érvényesek.

11.2.7. Szöveg formázott kiírása a PRINT USING utasítással

Szövegek formázott kiírására kevesebb karakter használható mint a számoknál. Ez azonban nem jelenti azt, hogy e karakterek megfelelő kombinálásával nem lehetne szövegeket szinte tetszőleges formátumban képernyőre írni.

A szövegek formázott kiírásának szemléltetésére módosítsuk az előző programunkat a következők szerint:

```
30 INPUT "SZOVEG ";A$
40 IF A$ = "-1" THEN END
50 PRINT USING FO$;A$
```

Ennyi volt az előkészület, és most lássuk a különböző formázó karaktereket.

11.2.8. A # jel a PRINT USING utasításban, szövegek kiírásánál

Nézzük először a #### formátumot (négy # jel). A vizsgált szöveg legyen a PLUS.

A számok formázott kiírásához képesti különbség rögtön felismerhető. A szöveg balra igazítva íródik ki. Írjuk be most a PLUS/4 szöveget.

Bár a beírt szöveg több karakterből áll, mint ahány # jelet a formátum mező tartalmaz, mégsem jelennek meg csillagok. A szövegből balról kezdve annyi karakter íródik ki, ahány # jel van a formátum mezőben.

Nézzük a következő formátumot:

(hat # jel, tizedespont, hat # jel)

A szöveg: A SZAMITOGEP

Láthatjuk, hogy a képernyőre a tizedespont nem íródik ki.

11.2.9. Az egyenlőségjel a PRINT USING utasításban

Az előbb megállapítottuk, hogy a szöveg kiírása a képernyőre az első oszlopban kezdődik, és annyi karakter íródik ki, ahány # jel van a formátum mezőben. Álljon most a formátum mező egy egyenlőségjelből és 39 db # jelből. Szövegnek írjuk be:

A PLUS/4 SZAMITOGEP. Ilyen egyszerűen lehet szövegeket egy képernyősor közepére írni. Az egyenlőségjel hatására egy szöveg a formátum mezőben középen jelenik meg.

Legyen most a formátum mező =###### (egyenlőségjel és hat # jel), a szöveg pedig PLUS/4. A szöveg a képernyőn jobbra igazítva jelenik meg. Ez a formátum megadás tehát erre is használható, még ha egy kicsit körülményes is. Ha a szöveg hétnél több karaktert tartalmazna, akkor a hetediktől jobbra levők a megadott formátumba már nem férnének bele, és a képernyőre nem is íródnának ki.

A formátum mezőn belül sem az egyenlőségjel helyének, sem a számuknak nincs jelentősége. Azt viszont számításba kell venni, hogy az egyenlőségjel szintén egy karakterpozíciót jelent, és a formátum mezőben legalább egy # jelnek lennie kell.

11.2.10. A nagyobb jel (>) a PRINT USING utasításban

Válasszuk most a

>####### (nagyobb jel, hét # jel)

formátumot.

A szöveg: PLUS/4

A kiírás a képernyő bal szélétől két pozícióval jobbra kezdődik. Ugyanilyen formátum mellett legyen a szöveg: A PLUS/4 GEP.

Most csak az A PLUS/4 szöveg jelenik meg, az ettől jobbra levő karakterek nem. A formátumot meghatározó karakterek helyére és számára az előbb elmondottak vonatkoznak.

11.3. A PRINT USING HASZNÁLATÁNAK TOVÁBBI LEHETŐSÉGEI

Az előzőekben ennek az utasításnak a különböző formázó karaktereivel ismerkedtünk meg. Nézzünk meg most néhány, a PRINT USING-gal használható programozói fogást.

Ha az utasítás több kiírandó szövegre vagy számra vonatkozik, akkor ezek mindegyikére érvényes az előzőleg definiált formátum. Például a

PRINT USING "####.##";1234.56," ",7890.125

eredménye:

1234.56 7890.13

Amint látjuk, a két szám azonos formátumban íródik ki, és hét üres karakter van köztük. A hetedik üres karakter a tizedespontnak felel meg. Ha az utasítás egy szöveg feldolgozásakor olyan formázó karakterrel találkozik, amelyik csak számokkal együtt használható, akkor ezt # jelnek tekinti. Ugyanez igaz fordítva is, azaz ha számok feldolgozásánál csak szöveggel együtt használható karakter fordul elő.

A

```
PRINT USING "=###.##";1234.56
```

eredménye:

1234.56

Bár tulajdonképpen egy # jel hiányzik, és ezért csillagoknak kellene a képernyőn megjelenniük, a gép a számot helyesen ábrázolja, mert az egyenlőségjelet # jelnek tekinti. Kísérő szövegek is egyszerűen kiírhatók:

A

```
PRINT USING "SIN##.####";SIN(123)
```

eredménye:

SIN-0.4599

Az egyes számok a formátum mezőben megadottaknak megfelelően jelennek meg a képernyőn.

Ha a PRINT USING után, vele azonos programsorban még egy PRINT utasítást adunk ki, akkor a PRINT USING által utolsóként kiírandó érték után egy pontosvesszőt kell kitenni. Például:

```
PRINT USING "######";"PLUS/4";:PRINT" SZAMITOGEP"
```

A PRINT USING után vezérlő karakterként vessző nem használható, mert a gép egy ilyen jel után további, kiírandó szöveget vagy számot vár. Ha az utasítássorban olyan vesszőt talál, amelyet nem követnek kiírandó karakterek, akkor hibajelzéssel leáll.

A PRINT USING utasításnak még további szolgáltatásai is vannak. Arra is lehetőséget nyújt, hogy a felhasználó néhány formázó karaktert átnevezzen. A következő fejezetben ezt a lehetőséget ismertetjük.

11.4. A PUDEF UTASÍTÁS

Szintaxis:

PUDEF "négy karakter"

Lehet, hogy valakinek jobban olvasható egy szám, ha abban a tizedespont helyett tizedesvessző szerepel, vagy a dollár jel helyett egy F betűt szeretne kiírni (mondjuk pl. a forint jelölésre) anélkül, hogy le kellene mondania a dollár jel pozicionálása által nyújtotta előnyökről. Ezeket a lehetőségeket kínálja a Plus/4-es BASIC nyelvében a PUDEF utasítás. Az utasítást négy karakterből álló karakterlánc követi, amely a számítógép bekapcsolása vagy egy RUN után a következő karaktereket tartalmazza:

" ,.$"

Ez a karakterlánc ugyan nem íratható ki a képernyőre, de a számítógép a PRINT USING utasítást úgy kezeli, mintha a PUDEF utasítás az alapértelmezés szerint a fenti karakterláncot definiálná.

Ezeknek a karaktereknek a szerepéről már részletesen szóltunk a PRINT USING utasítás használatánál. Ha esetleg valami nem volna érthető, célszerű az idevonatkozó részt ismét átnézni.

Nézzük meg közelebbről ezt a karakterláncot: az első egy üres karakter, a második egy vessző, a harmadik egy tizedespont és a negyedik a dollár jel. A PUDEF utasítással most e karakterek mindegyikét valamilyen másik karakterre cserélhetjük ki. Írjuk be a következőt:

```
PUDEF"#.,F"
PRINT USING "#$###,###.###";123456.78
```

A képernyőn ez jelenik meg:

```
#F123.456,780
```

Hogyan sikerült ezt kiírni? Az előzőekben említett karakterláncot átalakítottuk. Az első, eredetileg üres karakterből egy # jel lett. Ha ezek után a számítógép számok kiírásakor egyébként üres karaktert írna, akkor ehelyett most a # jelet írja ki. Ez egyébként csak a számok kiírására vonatkozik. Szöveg esetén továbbra is üres karaktert ír ki. Ki lehet próbálni!

Második karakterként az eredetileg vessző helyett tizedespontot definiáltunk. A harmadik, eredetileg tizedespontot vesszőre, a dollár jelet pedig F-re cseréltük ki. Ezek után mindannyiszor, amikor a számítógép egy szám kiírásakor eredetileg üres karaktert, vesszőket, tizedespontokat vagy dollár jelet írna, most rendre # jelet, tizedespontot, vesszőt vagy az F betűt fogja írni. A megváltoztatandó karaktereket karakterváltozóban is át lehet adni a PUDEF utasításnak. Arról azonban nem szabad megfeledkezni, hogy ez a karakterlánc csak négy karaktert tartalmazhat.

Figyelem: A PLUS/4 gépnél a dollár jel csak akkor cserélhető ki másik jelre, ha a PRINT USING formátummezejében a dollár jel előtt minimum egy # jel áll. Ha erre nem figyelünk, akkor a képernyőn továbbra is a dollár jel jelenik meg.

Nincs szükség arra, hogy mindig a teljes karakterláncot kelljen megváltoztatni. Ha például csak az üres karaktert akarjuk csillagra változtatni, akkor ez is elegendő:

```
PUDEF "*"
```

A másik három karakter – még azok is, amelyeket egy PUDEF utasítással előzőleg esetleg már megváltoztattunk – ennek a hatására nem változik meg. Arra kell csak ügyelni, hogy a karakterlánc négynél több karaktert nem tartalmazhat, mert ellenkező esetben a számítógép a futó programot az "?ILLEGAL QUANTITY ERROR IN..." hibaüzenettel megszakítja.

Ezzel befejezzük a PRINT USING és a PUDEF utasításokat ismertető fejezetet. Lehet, hogy ezek első olvasásra egy kicsit bonyolultnak tűnnek, de úgy gondoljuk, hogy némi gyakorlás után igen jól használhatók a különböző programokban.

11.5. A KURZOR POZICIONÁLÁSA

A Plus/4-es 3.5 változatszámú BASIC nyelve olyan utasításokat is ismer, amelyek lehetővé teszik a kurzor közvetlen pozicionálását. Ezek használatával a kurzor a képernyőre írás előtt a képernyő tetszőleges helyére állítható.

11.5.1. A kurzor pozicionálása a CHAR utasítással

Szintaxis:

CHAR oszlop,sor,szöveg

A Plus/4-es gépnél a sorok 0 és 24, az oszlopok pedig 0 és 39 között vannak számozva.

Ezzel az utasítással a kurzor nem csak grafikus, hanem a normál, karakteres üzemmódban is pozicionálható. A

CHAR,0,0,""

utasítás a kurzort a képernyő bal felső sarkába állítja. Egy ezt követő PRINT utasítás most ezen a helyen kezdi el a kiírást. A kurzor a képernyő tetszőleges helyére állítható, akár olyanra is, ahol már van valamilyen karakter. A CHAR utasítással szöveg is kiírható. A

CHAR,6,5,"A PLUS/4 SZAMITOGEP"

hatására a képernyő ötödik sorának hatodik oszlopától kezdődően kiírja az utasításban megadott szöveget. A grafikus üzemmóddal ellentétben mindazok a kiírást vezérlő karakterek is végrehajthatók, amelyek egy karakterlánc részei lehetnek. A szöveg akár inverz módban, vagy villogtatva, sőt különböző színekben is megjeleníthető. A CHAR utasítással részletesebben a grafikáról szóló fejezetben foglalkozunk.

11.5.2. Az SPC(X) utasítás

Szintaxis:

PRINT SPC (szám vagy numerikus változó)

Az utasítás hatására a kurzor a megadott értéknek megfelelő számú oszloppal lép tovább (másként fogalmazva az utoljára kiírt karakter után ennyi üres karakter, *space* jelenik meg). Az üres helyek számát a SPC után, zárójelben kell megadni. A legnagyobb szám 255, a legkisebb 0 lehet. Ennél nagyobb vagy kisebb szám hibajelzést okoz. Tizedestörtek, így pl. 12.45 vagy 0.12 is használhatók, de a gép csak az egész részt veszi figyelembe. A SPC(X) elé mindig ki kell írni a PRINT utasításszót. Egy példa a SPC(X) használatára:

```
10 PRINT CHR$(147)
20 FOR I=1 TO 3
30 PRINT SPC(5)"PLUS/4";
40 NEXT I
```

Látható, hogy a program a PLUS/4 szót először a képernyő bal szélétől számított öt üres karakterpozíció után írja ki, a második és az első szó között szintén öt üres pozíció van, és így tovább. A kurzor a SPC(5) hatására ugrik át öt üres pozíciót. Ha azt szeretnénk, hogy egy programban pl. az egyes kiírásokat elválasztó távolság változtatható legyen, akkor a SPC argumentumaként egy változót – pl. SPC(X) – kell megadni.

11.5.3. A TAB(X) utasítás

Szintaxis:

PRINT TAB (szám vagy numerikus változó)

Az argumentum értékének 0-255 között kell lennie. Ellenkező esetben hibajelzést kapunk. Az argumentum itt is lehet tizedestört, de a gép csak az egész részt veszi figyelembe.

A TAB(X) csak a PRINT utasításszóval együtt használható. Mint tudjuk, egy képernyősorba 40 karakter írható. A TAB segítségével meghatározható, hogy a karakterek kiírása a képernyő melyik oszlopában kezdődjön.

Írjuk be az alábbi kis programot:

```
10 PRINT CHR$(147)
20 FOR I=1 TO 6
30 PRINT TAB(16) "PLUS/4";
40 NEXT I
```

A programot futtatva világosan látható a PRINT SPC(X) utasítással szembeni különbség. Az első kiírás az első képernyősor 16-ik pozícióján kezdődik, a következő három közvetlenül egymás után jelenik meg. Csak az ötödik kiírás kerül ismét a második sor 16-ik oszlopába.

Ha a számítógép egy programban egy PRINT TAB(X) utasítást talál, akkor megvizsgálja, hogy a kurzor az aktuális képernyősor hányadik oszlopában van. Ha az oszlop száma nagyobb az argumentum értékénél, akkor a kurzor az aktuális pozícióján marad. Ha az oszlop száma viszont kisebb, akkor a kurzor a megadott értéknek megfelelően továbblép. A számítógép az oszlopokat 0-tól 39-ig számozza. Ha az argumentum 39-nél nagyobb, akkor a kurzor minden esetben továbblép. A TAB és a zárójel közé nem szabad szóközt írni, mert akkor változónak érti.

11.5.4. A POS(X) utasítás

Szintaxis:

PRINT POS (álargumentum)

A POS(X) utasítással a kurzornak az aktuális sorban elfoglalt pozíciója kérdezhető le. A pozíció kiírására a

PRINT POS(X)

paranccsal lehetséges. A kurzor aktuális pozíciója egy változóba is beírható:

A = POS(X)

Az aktuális pozíciót most az A változó tartalmazza, és ez felhasználható pl. a kurzor újabb pozicionálására. Ne felejtsük el, hogy a gép az oszlopok számolását 0-nál kezdi.

11.5.5. Pozicionálás a kurzormozgató karakterekkel

Közvetlen üzemmódban a kurzormozgató billentyűkkel a képernyő bármely pontját elérhetjük. Ez program üzemmódban is megtehető úgy, hogy a kurzort most nem billentyűk, hanem idézőjelek közé írt vezérlő karakterek mozgatják. Nézzük meg ezt egy példán:

```
10 PRINT CHR$(147)
20 PRINT "A";
30 ezt a sort a szöveges részben ismertetjük
40 PRINT "SZAMITOGEP"
```

30-as sor: Írjuk be a PRINT utasításszót, majd egy idézőjelet. Nyomjuk le nyolcszor egymás után a kurzor lefelé mutató nyílbillentyűjét (CURSOR DOWN). A billentyű minden egyes lenyomásakor egy inverz Q jelenik meg a képernyőn. Írjunk be végül még egy idézőjelet és egy pontosvesszőt.

A programot futtatva látjuk, hogy először megjelenik az A névelő, majd nyolc képernyősorral lejjebb a SZAMITOGEP szó. A számítógép az idézőjelek közé beírt vezérlőkaraktereket ugyanúgy értelmezte, mintha azokat közvetlenül a billentyűzetről adtuk volna be.

Az inverz Q úgy is megjeleníthető a képernyőn, hogy a CTRL és a RVS ON billentyűkkel bekapcsoljuk az inverz üzemmódot, és ekkor írjuk be a Q betűt. Ennek ugyanaz a hatása mint az előzőnek, csak körülményesebb a beírása. A Plus/4-esnek számos ilyen vezérlő karaktere van. Ilyen módon megválasztható például a karakterek színe, vagy be- és kikapcsolható az inverz üzemmód. A villogtatás be- és kikapcsolása is egyszerűen programozható. Ahhoz, hogy a gép az ilyenkor beírt karaktereket vezérlő karakterekként értelmezze, előzőleg be kell kapcsolni az idézőjel üzemmódot (az idézőjel beírásával). Ha pl. egy idézőjel után egyidejűleg lenyomjuk a SHIFT és a CLEAR/HOME billentyűket, akkor a képernyőn egy inverz szív jelenik meg. Ha a számítógép egy program végrehajtása során ilyen karaktert talál, akkor törli a képernyőt.

A kurzornak ezekkel a különleges karakterekkel való vezérlése éppenséggel nem javítja egy program áttekithetőségét, különösképpen akkor nem, ha több ilyen karakter is követi egymást. Azonban ezen lehet segíteni.

11.5.6. Írás képernyőre a CHR$ utasítással

Szintaxis:

PRINT CHR$ (szám vagy numerikus változó)

vagy

A$ = CHR$ (szám vagy numerikus változó)

Mint közismert, a számítógép csak számokat tud feldolgozni. A betűkkel nem tud mit kezdeni. Ezért minden beírt betűt számmá alakít át.

Éppen az előbb láttuk, hogy a Plus/4-es számítógép különböző képernyőszerkesztő karaktereket is „ismer". A képernyő törlésére alkalmas karakterhez például a 147-es szám, az inverz üzemmódot bekapcsoló REVERSE ON karakterhez pedig a 18-as szám van hozzárendelve. Az egyes karakterekhez ily módon hozzárendelt számokat az illető karakter ASCII-kódjának nevezik. A legtöbb számítógép ezeket a kódokat használja, bár a géptípusoktól függően előfordulhatnak eltérések. Nem rendelkezik például mindegyik számítógép annyiféle képernyőszerkesztő karakterrel, mint a Plus/4-es gép. Az ASCII-kódok 0 és 255 közé esnek.

Írjuk be:

```
PRINT "A"
```

Az eredmény nyilván nem meglepő, a képernyőn megjelenik az A betű. Ugyanezt másként is elérhetjük:

```
PRINT CHR$(65)
```

A CHR$ azt közli a számítógéppel, hogy a zárójelben utána álló számot tekintse ASCII-kódnak, és egy kódtáblázatból válassza ki az ehhez a kódhoz tartozó karaktert. A CHR$ elé írt PRINT utasításszó az így kiválasztott karaktert egyúttal a képernyőre is kiírja. Végrehajtva az utasítást láthatjuk, hogy a 65-ös ASCII-kódnak az A betű felel meg.

Van néhány olyan vezérlő karakter, amely egyszerűen, képernyőszerkesztő karakterekkel nem érhető el. Ebben az esetben a vezérlő karakter csak a PRINT CHR$ (kódszám) utasítással hajtható végre. Például:

```
PRINT CHR$(14)
```

A 14-es kóddal tehát a nagybetűs üzemmódról kisbetűsre kapcsoltunk át. A nagybetűs üzemmódra a SHIFT és a C= billentyűk lenyomásával kapcsolhatunk vissza. Ezt programból a

```
PRINT CHR$(142)
```

kiadásával érhetjük el.

A billentyűzet átkapcsolása meg is tiltható:

```
PRINT CHR$(8)
```

Próbáljuk meg most az üzemmódot átkapcsolni a SHIFT és a C= billentyűkkel! Az átkapcsolást ismét engedélyezhetjük:

```
PRINT CHR$(9)
```

A CHR$ segítségével valamely karakter is átadható egy karaktervváltozónak:

```
A$ = CHR$(X)
```

Az A$-ba most az a karakter kerül, amelynek az ASCII-kódja az X értéke.

11.5.7. Az ASC utasítás

Szintaxis:

ASC („szövegkifejezés")

Bár ez az utasítás nem ebbe a fejezetbe illik, mégis itt foglalkozunk vele, mivel ez a CHR$ megfordított művelete. Amint a szintaxisból látszik, ez a művelet karaktereket dolgoz fel, míg a CHR$ számokat.

Írjuk be:

```
PRINT ASC("A")
```

A képernyőn most a 65 jelenik meg. Az előzőekből már tudjuk, hogy a 65 az A betű ASCII-kódja. Azt a karaktert, amelynek az ASCII-kódját meg akarjuk tudni, zárójelbe, és ezen belül idézőjelbe kell tenni. Az idézőjelek közé több karakter is beírható, de a művelet eredménye mindig csak az első karakter kódja lesz, az ezután álló karakterek figyelmen kívül maradnak.

Ezek is használhatók:

X = ASC("A") vagy X = ASC(A$) vagy PRINT ASC(A$)

Ügyeljünk arra, hogy a karakterváltozót ne tegyük idézőjelek közé, mert ilyenkor az eredmény annak a betűnek a kódja lenne, amelyikkel a karakterváltozó neve kezdődik.

11.6. ÍRÁS A KÉPERNYŐRE A POKE UTASÍTÁSSAL

Szintaxis:

POKE tárcím,tartalom

Ezzel az utasítással részleteiben nem itt foglalkozunk, hanem majd a 17. fejezetben. Ha valaki esetleg nem ismerné az utasítást és a vele kapcsolatos trükköket, akkor először elolvashatja a hivatkozott fejezetet.

Figyelem: a fejezet hátralevő részében szereplő számokon a gépbe íráskor ne változtassunk! Ellenkező esetben előfordulhat, hogy a számítógép „lemerevedik", és csak egy RESET-tel lehet ismét „életre kelteni".

11.6.1. A képernyő felépítése karakteres üzemmódban

A Plus/4-es a hozzá csatlakoztatott képernyőn 25 sort és 40 oszlopot tud kezelni. Egyidejűleg tehát 25x40, azaz összesen 1000 karakter lehet a képernyőn. E karakterek mindegyike egy-egy képernyőpozíciót foglal el. Amint tudjuk, minden programozható számítógépben van úgynevezett RAM tár (olvasható és írható tár). A Plus/4-es gép

ennek a RAM tárnak egy részét képernyőtárnak, egy másik részét színtárnak használja. Azt is tudjuk, hogy egy számítógépben az egyes tárhelyek sorszámozva vannak – ezek a sorszámok jelentik az egyes tárcímeket. A Plus/4-es képernyőtárának és színtárának kezdő- és végcímei alapesetben:

decimális	**hexadecimális**	
képernyőtár	3072–4071	$0C00–$0FE7
színtár	2048–3047	$0800–$0BE7

A számozás a képernyő bal felső sarkánál kezdődik. Ez a kiinduló pozíció. A számozás balról jobbra halad. Ha egy sor végére értünk, akkor a számozás a következő sor elején folytatódik. Nézzünk egy példát egy képernyőpozíció tárcímének kiszámítására. Mi a címe a tárban a képernyő 3. sora és 25. oszlopa által kijelölt pozíciónak?

A bal felső sarokban levő pont tárcíme 3072. Adjunk ehhez 40-et (hiszen mindegyik képernyősor 40 karakter hosszú), így most megkapjuk a második sor első pozíciójának a tárbeli címét. Adjunk az eredményhez ismét 40-et, most 3152-t kapunk. Ez a szám a harmadik képernyősor első pozíciójának tárbeli címe. Ehhez a számhoz 24-et kell hozzáadni, hogy megkapjuk ebben a sorban a 25. pozíció címét. Az eredmény 3176, ami tehát a képernyő harmadik sora és 25. oszlopa által meghatározott pozíció tárcíme.

A fenti számítás alapján egyszerűen felállítható a képlet is: a képernyőn valamely pozíció tárcíme = kiinduló pozíció címe 40 * sorok száma + oszlopok száma.

Itt vegyük figyelembe, hogy a sorok és az oszlopok számozása 0-tól kezdődik.

Ellenőrizzük ezzel a képlettel az előző számításunkat.

A tár így kijelölt 1000 címének mindegyike tartalmaz valamilyen értéket. Egy-egy érték annak a karakternek felel meg, amely a képernyő megfelelő pozícióján éppen látható. Az egyes tárcímek tartalma 0–255 között lehet.

Nézzünk egy példát. Válasszuk ehhez az előbb kiszámított 3176-os tárcímet. Töröljük a képernyőt, és írjuk be:

```
POKE 3176,1
```

A képernyőn, a kiszámított helyen megjelenik egy nagy A betű. Amint látjuk, a képernyőtárra vonatkozóan ezek az értékek **nem** azonosak az ASCII-kódokkal, alapesetben. Az inverz ábrázolás is megvalósítható a POKE utasítással. Ehhez mindössze az a teendő, hogy a beírandó értékhez, esetünkben az 1-hez hozzá kell adni 128-at. Próbáljuk ki!

Ezzel az eljárással a képernyő bármely pozíciójába, bármely, a Plus/4-es által ismert karaktert kiírhatunk. Kivételt csak a képernyő vezérlő karakterei képeznek. A nagybetű/kisbetű üzemmódot a képernyőtáron keresztül, POKE utasítással nem lehet bekapcsolni.

A képernyőn megjelenő karakterek különböző színűek lehetnek. A számítógép tárának egy másik része, a színtár az egyes karakterek színére vonatkozó információkat tárolja. A színtárban is megfelel egy-egy tárcím a képernyő minden egyes pozíciójának. A képernyő pozíció, ill. a tárcím kiszámítása ugyanúgy történik, mint az előbb, csupán csak a kezdőcím más.

A Plus/4-es összesen 16 különböző színt tud megjeleníteni, és ezeknek a színeknek még a fényessége is változtatható. Ha közvetlenül akarunk a színtárba írni (azaz a POKE utasítással írunk), akkor ügyelni kell arra, hogy ekkor a színek „értéke" 0-nál kezdődik (ez a fekete szín), és a maximális érték a 15 (ez a világoszöld szín). Az egyes

színek fényességét úgy növelhetjük, hogy a szín „értékéhez" hozzáadjuk 16 valamely egész számú többszörösét (összesen 8 fényességi fokozat választható, 0 és 112 között). Az így beírható legnagyobb érték tehát a 127, ami a legnagyobb fényességi fokozatú világoszöld színnek felel meg (112+15). A fényességi fokozatok azt is lehetővé teszik, hogy a fehér szín különböző szürkeségű árnyalatait „keverjük ki". A fehér szín egyébként csak a legmagasabb fényességi fokozatban „igazán" fehér. Az egyes színeknek megfelelő értékeket itt külön nem soroljuk fel, ezeket bárki saját maga is kiszámíthatja (ill. a könyv végén levő függelékben megtalálhatja). A színek kiválasztását megkönnyíti, hogy a számbillentyűkön megtalálhatók az egyes színek angol nevének rövid alakjai. A POKE utasításhoz a szín értékét úgy választjuk ki, hogy az illető színhez tartozó számbillentyűn levő számból 1-et levonunk. A fekete szín értéke eszerint 0, a kék színé 6, és a narancsé 8 (8+1−1).

A színek változásai – a képernyő minőségétől is függően – csak a magasabb fényességi fokozatokban észlelhetők. Nézzünk egy példát a fényesség számítására. A világoskék szín értéke a 13; (8+6−1). A második fényességi fokozata 29; (13+16). Ennek a színnek a legmagasabb fényességi fokozatát a 125 adja (13+7*16). Töröljük a képernyőt, és írjuk be:

```
POKE 3472,1
POKE 2448,13+4*16
```

A képernyő tizedik sorában megjelenik egy A betű, világoskék színnel, az ötödik fényességi fokozatban.

Ha a képernyőn még nem lenne érzékelhető a fényesség változása, akkor a 4-es szorzó helyett válasszuk az 5-öst, vagy még nagyobbat.

Legyen most ez a szorzó tényező 11. A szín nem változik, viszont a betű villog. Ha a színtárba 127-nél nagyobb értéket írunk, akkor a hozzá tartozó tárcímen levő karakter villog. A villogást tehát igen egyszerűen hozhatjuk létre: a kívánt szín értékéhez hozzáadunk 128-at.

A színtár tartalma tetszés szerint változtatható. Ha a képernyőtár megfelelő tárcímén nincs karakter, akkor a képernyő ezen a helyen változatlan marad.

A POKE utasítással a karakterek törölhetők is. Írjunk a képernyőtárba egy üres karaktert (32). Ugyanez a hatása a SHIFT (szóköz) (96) „karakter"-nek is; de menjünk csak a kurzorral arra a helyre, ahol a törölt karakter volt, és nyomjuk le a RETURN billentyűt! Megjelenik a READY üzenet. Ennek a képernyő szerkesztő az oka.

Számos eset van, amikor egy karaktert közvetlenül, POKE utasítással írunk a tárba. Néha aránytalanul sok programozási munka szükséges ahhoz, hogy egy PRINT vagy egy PRINT USING utasítással a képernyő egy meghatározott helyére valamilyen karaktert írjunk ki. Ugyanakkor azt is vegyük figyelembe, hogy a túl sok POKE utasítás a programot áttekinthetetlenné teszi. Ha egyidejűleg sok karaktert kell megjeleníteni, akkor a PRINT utasítás lényegesen gyorsabb.

Ezzel ennek a fejezetnek a végére értünk. Lehet, hogy van, akinek helyenként túl részletesnek tűnt. Ezt a fejezetet azonban nagyon fontosnak tartjuk, ezért úgy gondoltuk, hogy inkább többet, mint kevesebbet írunk. Itt is igaz a közmondás: gyakorlat teszi a mestert. Próbálgassuk az ismertetett utasításokat különböző feladatokban. Bizonyára számos, új lehetőséget fedezünk fel.

12. A végrehajtást vezérlő utasítások

Mint ismeretes, a számítógép a programok végrehajtását a legkisebb sorszámú sorral kezdi, és az egyes programsorokat sorszámuk szerinti növekvő sorrendben hajtja végre. Ez mindaddig igaz, amíg a programban egy olyan utasítást nem talál, amely ettől a végrehajtási sorrendtől való eltérésre utasítja. A most ismertetésre kerülő utasításokkal a program végrehajtási sorrendjét meg lehet változtatni, és a program futását a programon belül, egy másik helyen lehet folytatni.

Ezek az utasítások az eddigiekben tárgyaltaktól nem csak írásmódjukban, hanem tulajdonságaikban is eltérnek. Olvassuk figyelmesen ezt a fejezetet, mert ezek az utasítások a hosszabb programok többségében nélkülözhetetlenek.

12.1. A GOTO UTASÍTÁS

Szintaxis:

GOTO sorszám

Nagyon gyakran előfordul, hogy egy programnak a végrehajtása közben egy eseményre reagálnia kell. Az esemény elemzéséhez és a reagáláshoz általában annál több helyre van szükség, mint amennyi abban a programsorban, amelyben az esemény bekövetkezett, rendelkezésre áll. Ilyenkor a program végrehajtását el kell ágaztatni. A GOTO utasítás szintaxisából látszik, hogy az utasításszó után egy sorszámnak kell állnia. Ez a sorszám annak a programsornak a száma, amelyre a programnak el kell ágaznia. Ennek a sorszámnak a programban valóban létező sorszámnak kell lennie, mert ellenkező esetben a gép a program futását hibaüzenettel megállítja.

Ha a számítógép egy sorszámot tartalmazó GOTO utasításhoz érkezik, akkor az első sorszámtól kezdve átnézi a programtárat, és keresi a megadott sorszámot. Ha megtalálta, akkor a program végrehajtását ebben a sorban folytatja. Egy példa:

```
10 PRINT"ELAGAZAS A 60-AS SORRA (I/N) ?"
20 GETKEY A$
30 IF A$="I" THEN GOTO 60
40 PRINT "UGRAS VISSZA A 20-AS SORRA"
50 GOTO 20
60 PRINT "AZ ELAGAZAS MEGTORTENT"
70 PRINT "UGRAS VISSZA A 10-ES SORRA":GOTO 10
```

Ez a program egyszerre több elágazást (vagy más, szintén elterjedt szóhasználattal ugrást) is tartalmaz, hogy a különböző tulajdonságaikat jobban megértsük. Jól

látható, hogy egy GOTO segítségével egész programrészeket is átugorhatunk. Az utasítással a program elejére is ugorhatunk. Változtassuk meg a 30-as sort:

```
30 IF A$ = "I" THEN 60
```

A program futásából látszik, hogy a számítógép ezt az írásmódot is megértette. A gép a sorszám elé „odagondolta" a GOTO utasításszót. Az 50-es és 70-es sorok viszont így nem változtathatók meg. Itt a számítógép továbbra is „elvárja" a GOTO-t.

12.2. AZ ON...GOTO UTASÍTÁS

Szintaxis:

ON numerikus kifejezés GOTO sorszám,sorszám,...,sorszám

Tegyük fel, hogy olyan programot akarunk írni, amelyben menü alapján három különböző programrész közül választhatunk. A menüt ki lehet alakítani úgy, hogy a kívánt programrészt egy szám beírásával választhassuk ki. A program a szám beírására megfelelő IF...THEN sorokban reagálhat.

Ennél azonban jobb megoldás is van. Írjuk be a következő programot:

```
10 PRINT CHR$(147)
20 PRINT "1.PONT (1)"
30 PRINT "2.PONT (2)"
40 PRINT "3.PONT (3)"
50 GETKEY A$
60 I=VAL(A$)
80 ON I GOTO 100,200,300
100 PRINT "EZ AZ 1-ES PONT" :GOTO 20
200 PRINT "EZ A  2-ES PONT" :GOTO 20
300 PRINT "EZ A  3-AS PONT" :GOTO 20
```

Indítsuk el a programot, és írjuk be a számokat 1-től 3-ig. A program minden esetben a megfelelő sorra ugrik. Ha az I változó értéke 1, akkor a GOTO utasításszó után elsőként megadott sorszámra, ha az I értéke 2, akkor a másodikként, ha pedig 3, akkor a harmadikként megadott sorszámra ugrik. Írjunk be most egy 4-est. Jóllehet nem adtunk meg negyedik sorszámot, a program a 100-as sorra ugrik. Ugyanez történik, ha egy betűt vagy egy 0-t írnánk be.

Jegyezzük meg: ha a változóban megadott ugrási helyek számánál nagyobb érték vagy 0 van, akkor a program végrehajtási sorrendje nem változik, azaz nincs elágazás. A program végrehajtása a sorszám szerint következő programsorban folytatódik. A programnak ez a reagálása gyakran nem kívánatos (hiszen a program ilyenkor nem azt teszi, amit mondunk neki). Programunkban a 70-es sort nem véletlenül hagytuk ki. Íme a megoldás:

```
70 IF I > 3 OR I < 1 THEN 50
```

A GOTO utasítás nagyon jól használható arra, hogy egy program végrehajtását valamely más részén folytassuk. Ha viszont a végrehajtásnak néhány programsor feldolgozása után ismét vissza kell térnie az elágazási helyre, akkor ezt külön bele kell írni a programba. Most megismerkedünk egy olyan utasítással, amely ennek a visszatérésnek a programozását minimálisra csökkenti.

12.3. A GOSUB UTASÍTÁS

Szintaxis:

GOSUB sorszám

Számos programban előfordul, hogy néhány, egymással összefüggő programsort igen gyakran kell végrehajtani. Nézzük például a billentyűzet lekérdezését. Mivel egy programot úgy kell megírni, hogy azt egy hibás adat beírása lehetőség szerint ne ronthassa el, ennek programozásához egy hosszabb rutinra van szükség. Egy programon belül általában több helyen van arra szükség, hogy adatokat vigyünk be a billentyűzeten keresztül. Természetesen ezt a rutint minden ilyen helyre be lehetne írni. Ez azonban nemcsak magának a programnak a hosszát növeli, és teszi ezáltal áttekinthetetlenebbé, hanem a tárban sok helyet foglal el, és a beírásához is több munka kell.

Ezért kézenfekvő az a megoldás, hogy ezt a rutint csak egyszer programozzuk, és magát a programot mindannyiszor, ahányszor erre a rutinra szükség van, ágaztassuk el erre a rutinra. Az adatbevitel után a rutinból ugorjunk vissza a főprogramba, hogy az adatokat ott, ahol ezekre szükség van, fel lehessen dolgozni. Pontosan ezt a műveletsort végzi el a GOSUB utasítás.

A GOSUB az angol nyelvű GO SUBROUTINE kifejezés rövid alakja, és magyarra az UGORJ AZ ALPROGRAMRA kifejezéssel fordítható. A GOSUB utasításban is meg kell adni egy sorszámot, hogy a számítógép tudomására hozzuk, hogy a program végrehajtását melyik sorban folytassa. Ha a számítógép ilyen utasítást talál, akkor megjegyzi magának, hogy a program végrehajtását melyik sorban szakította meg. Elágazik a megadott programsorra, és a végrehajtást itt folytatja mindaddig, míg egy RETURN utasítást nem talál. A RETURN utasítás hatására visszatér arra a programsorra, amely a GOSUB után következik.

Egy programot például így lehet tagolni:

```
FŐPROGRAM
.
GOSUB ALPROGRAM
.
.
GOSUB ALPROGRAM
.
.
END
ALPROGRAM
.
.
RETURN
```

Nézzük meg ezt egy konkrét példán:

```
10 REM ITT KEZDODIK A FOPROGRAM
20 GOSUB 100
30 PRINT "EZ ISMET A FOPROGRAM"
40 IF A$="I" THEN GOTO 10
50 PRINT "A PROGRAM VEGET ERT."
```

```
60 END:REM A PROGRAM VEGE
100 REM ITT KEZDODIK AZ ALPROGRAM
110 PRINT "EZ AZ ALPROGRAM"
120 PRINT "MEG EGYSZER ? (I/N)"
130 GETKEY A$
140 IF A$ <> "I" AND A$ <> "N" THEN 130
150 RETURN:REM AZ ALPORGRAM VEGE
```

Indítás után a program végrehajtása rögtön az alprogramra kerül. Az alprogram egy billentyű lenyomására vár. Ha nem az I vagy az N billentyűt nyomjuk le, akkor a végrehajtás visszakerül a 130-as sorra, és megismétlődik a billentyűzet lekérdezése. Ha viszont a lenyomott billentyű vagy az I vagy az N, akkor a program a 30-as sorra ugrik vissza. A program kiértékeli a választ: ha a válasz I volt, akkor a 10-es sorba való ugrással kezdődik elölről az egész, ha N volt, akkor pedig befejeződik.

Egy alprogramból meghívható egy másik alprogram, amely ugyancsak meghívhat újabb alprogramot stb. Mi a kísérleteink során azt tapasztaltuk, hogy összesen 39 alprogram hívható meg egyidejűleg. Ez a szám azonban az egyidejűleg aktív ciklusok számától is függ. A meghívott 39 alprogram után az OUT OF MEMORY ERROR hibajelzést kaptuk. A hibát az okozza, hogy a számítógépben a visszaugrási címek tárolására csak korlátozott tárhely áll rendelkezésre.

Ügyelni kell arra, hogy a számítógép egy program végrehajtása során ne találkozhasson RETURN utasítással úgy, hogy előzőleg egy GOSUB utasítás egy alprogramot ne hívott volna meg. Ilyen esetben ugyanis a gép nem találná a visszatérési címet, és a program feldolgozását a RETURN WITHOUT GOSUB ERROR (RETURN GOSUB nélkül) hibaüzenettel megszakítaná. A programot ekkor a CONT utasítással sem lehetne folytatni.

Lehetőség szerint kerüljük el egy alprogramnak egy másik alprogramból való meghívását. Az ilyen egymásba ágyazások a programot hamar áttekinthetetlenné teszik, és a további bővítését is megnehezítik.

12.4. AZ ON...GOSUB UTASÍTÁS

Szintaxis:

ON numerikus kifejezés GOSUB sorszám,sorszám,...,sorszám

Ennek az utasításnak ugyanaz a hatása, mint az előbb tárgyalt ON...GOTO utasításnak. Mivel azonban ez az utasítás alprogramokat hív meg, minden egyes, GOSUB-bal meghívott programrésznek (rutinnak) RETURN utasítással kell végződnie.

13. Programciklusok

Gyakran van szükség arra, hogy egy-egy programrészt egymás után többször hajtsunk végre. Ezeknek az ismétlődéseknek (ciklusoknak) a programozására alkalmasak a ciklusutasítások. Ismerkedjünk meg ezekkel.

13.1. A FOR...NEXT UTASÍTÁS

Szintaxis:

FOR ciklusváltozó = num. kif. TO num. kif. STEP num. kif.

Talán érthetőbb a szintaxis a következő nevekkel:

FOR ciklusváltozó = kezdőérték TO végérték STEP lépésköz

- a FOR után mindig egy változónak kell állnia,
- a TO után állhat egy változó vagy egy szám, num. kif.
- a STEP megadása nem kötelező, ha nem adjuk meg, akkor a számítógép úgy értelmezi, hogy a lépésköz 1.

Nézzünk meg először egy kis programot:

```
10 FOR I = 0 TO 5
20 PRINT I;". CIKLUSLEFUTAS"
30 NEXT I
40 PRINT:PRINT " AZ I ERTEKE A CIKLUS VEGEN";I
```

A programot végrehajtva láthatjuk az egyes ciklusleſutásokat, majd a programot lezáró közlést, miszerint az I értéke a ciklus befejeződése után 6.

Az I a ciklusváltozó, az 5 a végérték, és mivel a STEP paramétert nem adtuk meg, a lépésköz az alapértelmezés szerint 1. Ha a számítógép egy ilyen ciklusutasítással találkozik, akkor először megkeresi a kezdőértéket, és összehasonlítja a végértékkel. Ezután a lépésközt keresi, és kezdődhet a cikluson való első áthaladás. Amikor a NEXT utasítással találkozik, akkor a ciklusváltozó értékét a lépésközzel megnöveli, és az így kapott eredményt ismét összehasonlítja a végértékkel. Ha ez nagyobb lenne mint a végérték (csak pozitív lépésköz esetén), akkor a program rátér a NEXT utáni programsor végrehajtására. Ha viszont kisebb, akkor a ciklus még egyszer lefut, és újra kezdődik az összehasonlítás.

Egy ciklusban visszafelé is lehet számlálni. Változtassuk meg a programot:

```
10 FOR I = 5 TO 1 STEP -1
```

A képernyőn most megjelenő állítások ugyan már nem igazak, de ezzel most ne törődjünk.

Amint az első programpéldában az I ciklusváltozó értéke a program végén 6 lett, úgy most 0 lesz. Ennek az az oka, hogy egy ciklus csak akkor fejeződik be, ha a ciklusváltozó értéke vagy nagyobb, vagy – mint a második példában – kisebb a végértéknél. Ezt a tényt a programozás során – például ha ugyanazt a ciklusváltozót a program különböző részén is használni akarjuk – feltétlenül figyelembe kell venni.

Egyidejűleg több ciklus is lehet aktív. A NEXT ilyenkor mindig az utoljára elindított ciklusra vonatkozik. Lássunk egy példát erre is:

```
10 FOR I=1 TO 2
20 PRINT "AZ ELSO CILUS ERTEKE =";I
30 FOR A=1 TO 5
40 PRINT "A MASODIK CIKLUS ERTEKE =";A
50 NEXT A
60 NEXT I
```

Az első ciklus kétszer, a második ötször fut le. Az egymásba ágyazás következtében a második ciklusban levő állítás (az úgynevezett ciklusmag) tízszer kerül a képernyőre. Az 50-es és a 60-as sorokban a változót nem kell feltétlenül megadni, de mindenképpen célszerű, mert így sokkal gyorsabban megállapítható, hogy a ciklus a programon belül hol fejeződik be.

Változtassuk meg most a programot, hogy lássuk, hogyan lehet FOR...NEXT ciklusokkal pl. egy keretet rajzolni a képernyőre.

```
10 PRINT CHR$(147);
20 PRINT " ****************************************"
30 FOR I=1 TO 23
40 PRINT "*                                      *";
50 NEXT I
60 PRINT " ****************************************" ;
70 GETKEY A$
```

A 20-as és 60-as sorok egy üres karakterből és 38 csillagból állnak. A 40-es sorban a két csillag között 38 üres karakter van. A keret felrajzolása után a számítógép egy billentyű lenyomására vár. Ha lenyomunk egy billentyűt, akkor a program befejeződik. Ha a képernyőn egy ilyen keretben dolgozunk, akkor kínálkozik a lehetőség, hogy egy, ennek a keretnek megfelelő nagyságú ablakot is definiáljunk. Ekkor a keretet egy esetleges túl hosszú beírással nem lehet olyan könnyen elrontani. Az ablak definiálásra vonatkozó tudnivalókat az ESC billentyűkről szóló alfejezet tartalmazza.

Az ilyen jellegű ciklusokból elágazó utasításokkal nem ajánlatos kilépni, mert ezek mindaddig aktívak maradnak, míg a fent említett feltételek nem teljesülnek, vagy meg nem nyitunk ilyen néven ismét egy ciklust, illetve lezárunk egy külső ciklust. A Plus/4-es BASIC nyelve azonban olyan ciklusokat is ismer, amelyekből speciális utasításokkal ki lehet lépni.

13.2. CIKLUSSZERVEZÉS DO...UNTIL/WHILE UTASÍTÁSOKKAL

Szintaxis:

DO UNTIL feltétel
DO WHILE feltétel

Lássunk ismét egy példát:

```
10 X = 5
20 DO UNTIL X =0
30 PRINT "AZ X =";X
40 X = X-1
50 LOOP
60 PRINT "AZ X ERTEKE A CIKLUS VEGGEN";X
```

A 10-es sorban az X változóhoz az 5-ös értéket rendeltük hozzá. A 20-as sorban indul a ciklus. Az itt levő utasítást szabadon így lehet megfogalmazni: hajtsd végre a ciklust mindaddig, míg az X értéke 0 lesz. A 30-as sorban képernyőre írjuk az X aktuális értékét. A 40-es sorban az X értékét 1-gyel csökkentjük. Az 50-es sorban befejeződik a ciklus. A számítógép a program végrehajtását a 20-as sorban folytatja. Először megvizsgálja az X értékét. Ha az X kisebb vagy nagyobb mint 0, akkor a ciklus végrehajtása folytatódik. Ha X = 0, akkor a program a LOOP utasítás utáni sorban folytatódik. A mi példánkban ez a 60-as sor. Az UNTIL utasításszó gondoskodik arról, hogy a ciklusból csak akkor lépjünk ki, ha az UNTIL után definiált feltétel teljesül.

Nézzük meg a WHILE szerepét:

```
10 A$ = "I"
20 DO WHILE A$ = "I"
30 PRINT "FOLYTATJA ? (I)"
40 GETKEY A$
50 LOOP
60 PRINT "A$ = ";A$
```

A ciklus mindaddig ismétlődik, amíg a kérdésre "I"-vel válaszolunk.

Ha a ciklusban a WHILE utasításszót használjuk, akkor a ciklusból csak akkor léphetünk ki, ha a WHILE után definiált feltétel már nem teljesül. Ezeket a ciklusokat is egymásba lehet ágyazni. Hasonlóan a FOR utasításhoz, amelyhez tartoznia kell egy NEXT-nek, mindegyik DO UNTIL/WHILE utasításhoz tartoznia kell egy LOOP-nak. Az első LOOP az utoljára indított ciklusra vonatkozik.

Amint már említettük, ezekből a ciklusokból bármikor ki lehet lépni. A kilépéshez azonban a GOTO és a GOSUB elágazó utasítások nem használhatók, mert ezek a ciklust nem zárják le. A kilépésre külön utasítás van.

13.3. AZ EXIT UTASÍTÁS

A DO...LOOP utasításokkal végtelen ciklusok is felépíthetők. Futtassuk ezt a programot:

```
10 DO
20 A = A+1
30 PRINT A
40 LOOP
```

Ezt a ciklust csak a STOP billentyűvel lehet megszakítani. Az ilyen ciklusból az EXIT utasítással lehet kilépni. Nézzünk egy másik programot, amelyen megismerhetjük az EXIT hatását:

```
10 X = 0
20 DO UNTIL X = 500
30 GET A$
40 IF A$ = "S" THEN EXIT
50 X =X+1
60 LOOP
70 PRINT "X =";X;A$
```

A programban levő ciklus futása kétféle módon fejeződhet be. Az egyik lehetőség, hogy a ciklust „magára hagyjuk", és megvárjuk, míg az X értéke 500 lesz. Ekkor a ciklus a „normál" módon, végig lefut. A másik lehetőség az, hogy a program futása közben lenyomjuk az S billentyűt. Ezt a 40-es sor teszi lehetővé. Ha lenyomjuk az S billentyűt, akkor a ciklusfeltétel „igazzá" válik, és a program futása a ciklust követően folytatódik. A számítógépet az EXIT utasítja, hogy hajtsa végre ezeket a műveleteket.

A gép számára az EXIT utasítás azt jelenti, hogy tekintse a ciklust befejezettnek, és ágazzon el a LOOP utasítást követő programsorra. A mi esetünkben ez a 70-es sor, amellyel az X változó tartalmát a képernyőre írjuk.

E ciklusok sokoldalú alkalmazásának egyik területe lehet például igen kényelmes beviteli rutinok programozása. Gyakran kell előírni azt, hogy a programba beírandó adatok egy bizonyos hosszúságot nem léphetnek túl, másrészt viszont ha rövidebbek, egy RETURN-nel legyen mód az adatbevitel befejezésére. Az itt következő program csak arra kíván ösztönzést adni, hogy miként lehet egy ilyen feladatot a Plus/4-essel megoldani. Kényelmes beviteli rutin természetesen még sok mást is tartalmaz, de legyen ez már mindenki számára önálló, programozási gyakorlat.

```
10 REM BEVITELI RUTIN
20 H=5
30 DO UNTIL LEN (A$)=H
40 GET E$
50 PRINT E$;
60 IF E$ = CHR$(13) THEN EXIT
70 A$=A$+E$
80 LOOP
90 PRINT:PRINT A$
```

A 20-as sorban meghatározzuk a bevitel megengedett hosszát. Ha ezt a hosszat kitöltjük adatokkal, akkor a ciklus normál módon fejeződik be. Ha viszont ezt

megelőzően lenyomjuk a RETURN billentyűt, akkor a ciklust a 60-as sorban levő EXIT fejezi be, és a ciklusból kilépünk.

Ennek a fejezetnek a végére értünk. Reméljük, hogy sikerült érthetően ismertetni a ciklusok programozását. Egészen biztosan mindenki gyakran fogja tapasztalni, hogy a programok többségét ciklusok nélkül nem lehet megírni. Ez különösen akkor igaz, ha egy adott feladatra rövid és áttekinthető programot akarunk készíteni.

14. Karakterláncokra vonatkozó utasítások

Ezt a fejezetet bizonyára sokan többször is fellapozzák majd. A 3.5 változatú BASIC nyelvben igen hatékony utasítások vannak karakterláncok feldolgozására. Annak érdekében, hogy ezeket összefüggésükben tárgyalhassuk, nem tartjuk magunkat az ábécé sorrendhez. Néhány utasításnak létezik a fordítottja is. Ezeket szintén a logikai összefüggésükben tárgyaljuk.

Tisztázzuk előbb a fogalmakat. Karakterlánc alatt egymással összefüggő, egy egységet alkotó karakterek sorozatát értjük. Egy karakterlánc állhat egy karakterből is. A karakterláncban lehetnek számjegyek is, ezeket a számítógép azonban karaktereknek tekinti, ezért ezekkel a számokkal matematikai műveletek nem végezhetők. A karakterláncot programon belül mindig idézőjelek között kell megadni. A fogalom elnevezésére karakterlánc, ill. karakterlánc típusú állandó helyett használják a szöveg, ill. szöveges típusú állandó elnevezést is. Ugyancsak ismert az eredeti angol string elnevezés is.

A változókról szóló fejezetben foglalkoztunk már a karakterlánc (vagy szöveges, ill. string) típusú változókkal. Azt is megbeszéltük, hogy a karakterlánc típusú változókat (vagy röviden a karakterváltozókat) az utánuk írt dollár jel ($) különbözteti meg a más típusú változóktól. Rendeljünk tartalmat egy karakterváltozóhoz:

```
A$ = "COMMODORE PLUS/4"
```

A változó tartalmát a

```
PRINT A$
```

utasítással írhatjuk a képernyőre.

A tartalmat átadhatjuk egy másik, azonos típusú változónak is:

```
B$ = A$
```

A PRINT B$ utasítás most ugyanazt a tartalmat írja ki, mint a PRINT A$.

A karakterváltozók egymással össze is kapcsolhatók:

```
C$ = A$+B$
```

A C$-ban most az A$ és a B$ tartalma van, tehát a C$ hossza ezek együttes hossza (az A$ vagy a B$ kétszerese). Ez is írható:

```
C$ = C$+C$
```

A C$ tartalmát tehát megkétszereztük. Írjuk ki ezt:

```
PRINT C$
```

Csak arra kell ügyelni, hogy a karakterlánc hossza a 255 karaktert ne haladja meg.

Érdekel talán valakit is, hogy hány karakter van a karakterváltozóban? Természetesen ezeket meg is lehet számlálni, de létezik olyan utasítás is, amely ezt elvégzi helyettünk. (Ebben a fejezetben az utasítások ismertetéséhez többnyire karaktervál-tozókat használunk. Helyettük használhatók azonban karakterláncok is, amelyeket idézőjelek között kell megadni.)

14.1. A LEN FÜGGVÉNY

Szintaxis:

LEN (karakterváltozó)

Ez az utasítás egy karakterváltozó tartalmának a hosszát adja meg (length = hosszúság). A karakterláncok feldolgozása során gyakran nagyon fontos, hogy ismerjük a karakterlánc pontos hosszát. Sokszor előfordul ugyanis, hogy egy karakterláncnak pontosan meghatározott részét kell megváltoztatni.

14.2. A LEFT$ FÜGGVÉNY

Szintaxis:

LEFT$ (A$, numerikus kifejezés)

Ezzel az utasítással egy karakterváltozóból annyi karaktert rendelhetünk egy másik karakterváltozóhoz, ahányat a számmal vagy a változóval megadtunk. A karakterek hozzárendelése a bal első karaktertől kezdődik (left = bal).

Egy példa:

```
10 INPUT "IRJON BE EGY KARAKTERLANCOT ";A$
20 INPUT "HANY KARAKTERT KELL A B$-BA ATADNI ";S
30 PRINT  "A LANC HOSSZA";LEN(A$);"KARAKTER"
40 B$ = LEFT$(A$,S)
50 PRINT B$
60 PRINT "TOVABB ? (I)"
70 GETKEY K$
80 IF K$ = "I" THEN 10
```

Az első kérdésre írjuk be a "SZAMITOGEP" karakterláncot, a karakterek száma pedig legyen hat. A karakterlánc hossza 10 karakter, amit a számítógép állapít meg (LEN A$), és ír ki. A program most az első hat karaktert hozzárendeli a B$-hoz, és ezt ugyancsak kiírja. Eközben az A$ hossza és tartalma nem változik, csupán csak az A$ első hat karaktere másolódik át a B$-ba. Arra is van lehetőség, hogy egy karakterlánc bizonyos, meghatározott részét megváltoztassuk. Erre a fejezet későbbi részében még visszatérünk.

Írjunk be egy olyan számot, amely nagyobb, mint az A$-ban levő karakterek száma. A program hiba nélkül lefut, és a B$-nak ugyanaz lesz a tartalma mint az A$-nak. Arra azonban feltétlenül ügyeljünk, hogy a karakterláncról leválasztandó karakterek száma ne legyen 0-nál kisebb és 255-nél nagyobb, mert ellenkező esetben hibajelzést kapunk.

14.3. A RIGHT$ FÜGGVÉNY

Szintaxis:

RIGHT$ (A$, numerikus kifejezés)

Ezzel az utasítással egy karakterlánc jobb oldali (right = jobb) karakterei adhatók át egy karakterváltozónak, illetve írhatók a képernyőre. Változtassuk meg az előző programot eszerint:

```
40 B$ = RIGHT$(A$,A)
```

Ha most megadunk egy karakterláncot, és a B$-ba átadandó karakterek számát, akkor ezek a karakterek a helyes sorrendben jelennek meg a képernyőn. A kiírás tehát nem a jobb szélső karakterrel kezdődik, hanem azzal, amelyet jobbról számolva az A-val kijelöltünk. Ez a szám a RIGHT$ utasítás esetén is lehet nagyobb, mint a karakterláncban levő karakterek száma. Ilyenkor a teljes karakterlánc kiíródik. Az átadandó karakterek száma azonban nem lehet 0-nál kisebb és 255-nél nagyobb.

Bár már az imént bemutatott utasítások is számos lehetőséget kínálnak a karakterláncok feldolgozására, van egy, ezeknél még sokoldalúbb utasítás is.

14.4. A MID$ FÜGGVÉNY

Szintaxis:

MID$ (A$,T,S)

ahol a T jelentése: -TÓL, (szöveges kifejezés, numerikus kifejezés)
az S jelentése: karakterek száma (numerikus kifejezés)

Ezzel az utasítással lehetőség van karakterláncok egyes részeinek átadására. A karaktersorozat T-edik karakterétől kezdődően S számú karakter kerül átadásra. A számlálás balról kezdődik.

A T értéke 1-nél nem lehet kisebb, ellenkező esetben hibajelzést kapunk. Az S értéke 0-nál nem lehet kisebb. Egyik változó értéke sem haladhatja meg a 255-öt. Ismét változtassunk a fejezet elején levő programon:

```
25 INPUT "HANYADIK KARAKTERNEL KEZDODJON A MASOLAS ";T
40 B$ = MID$ (A$,T,S)
```

A 25-ös sor új sorként kerül a programba. A 40-es sort ennek megfelelően kell változtatni.

Néhány kísérlet után könnyen megállapítható, hogy a B$-ba a karakterláncnak valóban mindig csak a megadott része kerül át, és íródik ki a képernyőre. Ha a T értékére 1-et, az S-re pedig a karakterlánc hosszát adjuk meg, akkor a B$-ba a teljes karakterlánc kerül át.

14.5. EGY KARAKTERLÁNC MEGVÁLTOZTATÁSA A MID$ UTASÍTÁSSAL

A MID$ segítségével igen egyszerűen megváltoztathatók a karakterláncok. A MID$ az egyenlőségjel bal oldalán is állhat. Írható ezért például ez is:

A$ = "1234567890"
B$ = "ABCD"
MID$ (A$,5) = B$

E műveletek elvégzése után az A$ tartalma:

1234ABCD90

A karakterlánc egy része tehát a B$ karakterváltozó tartalmával felülíródott. A behelyettesítés, ahogyan ezt a MID$ utasításnál megállapítottuk, az ötödik karakternél kezdődött. A hosszra vonatkozóan semmit sem kell megadni, mivel ezt a B$-ban levő karakterlánc hossza határozza meg.

Ugyanezt az eredményt adja a

MID$(A$,5) = "ABCD"

hozzárendelés is.

Figyelem: A megváltoztatandó karakterlánc hosszát az átalakítás nem változtathatja meg, mert különben hibajelzést kapunk.

Természetesen egy karakterlánc részei helyettesíthetők más karakterlánc részeivel is. Ekkor az egyenlőségjel jobb oldalán olyan utasításoknak kell állni, mint a LEFT$, RIGHT$, vagy egy újabb MID$.

14.6. AZ INSTR FÜGGVÉNY

Szintaxis:

INSTR (A$,S$, kezdő pozíció)

Ezzel az utasítással megvizsgálható, hogy az S$-ban levő karakterlánc része-e az A$-ban levő karakterláncnak. Ha az A$ tartalmazza az S$-t, akkor a függvény értéke az a szám lesz, ahányadik pozíción kezdődik az S$ az A$-on belül. Ha a keresett karakter vagy karakterlánc nem része a megadott karakterláncnak, akkor a függvény értéke a keresés után nulla. Ha nem adunk meg kezdő pozíciót, akkor a keresés a karakterlánc elején kezdődik.

Nézzünk egy példát:

```
10 A$="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
20 GETKEY B$
30 X=INSTR (A$,B$)
40 IF X=0 THEN 20
50 PRINT B$;X
60 GOTO 20
```

Ez a program az indítása után csak olyan karaktereket fogad el, amilyeneket az A$-ban definiáltunk. A program az illető karaktert és a karakterláncon belül elfoglalt pozícióját a képernyőre írja. Ez a kis program egyszerű megoldással lehetővé teszi, hogy bizonyos karakterek beírását engedélyezzük, míg másokét nem. Egy ilyen beviteli rutinnal megtakarítunk néhány IF...THEN lekérdezést, ennek következtében pedig tárhelyet és időt is.

14.7. A VAL FÜGGVÉNY

Szintaxis:

VAL(X$)

Ismeretes, hogy azokkal a számokkal, amelyek karakterláncok részét képezik, nem lehet számításokat végezni, mert a számítógép ezeket a számokat karakterekként értelmezi. A VAL utasítással ezeket a karaktereket számokká lehet átalakítani.

Az utasítás az X$ karakterváltozóban balról jobbra keresi a számokat. Ha más karaktert talál, akkor a keresést befejezi. A karakterláncnak a számokat az előírt alakban kell tartalmaznia. Ahhoz, hogy egy számot teljes hosszában átadhassunk a változónak, a szám tizedespont helyett nem tartalmazhat tizedesvesszőt. Mivel a gép befejezi a keresést, ha meg nem engedett karaktert talál, az előállítandó számnak a karakterlánc elején kell állnia. (A normál alak használata is megengedett.)

Néhány példa:

utasítás	**eredmény**
PRINT VAL "ABC123"	0
PRINT VAL "123.45"	123.45
PRINT VAL "123AB4"	123
PRINT VAL "−1.2A1"	−1.2
PRINT VAL "−3.5E+2"	−350

14.8. AZ STR$ FÜGGVÉNY

Szintaxis:

STR$ (numerikus kifejezés)

Ezzel a függvénnyel számokat karakterlánccá alakíthatunk át. Az ily módon átalakított számok ezután feldolgozhatók a karakterláncokra kiadható utasításokkal. Ez az utasítás a VAL utasítás fordítottja. Az utasítás hatására a karakterlánc a számokat mint karaktereket tartalmazza. Most tehát ezzel a karakterlánccal számítások is végezhetők. Az

```
A$ = STR$(1234.56)
PRINT A$
```

utasítások eredménye:

1234.56

Ezzel a fejezet végére értünk. A programozási munkák során nagyon gyakran kell karakterláncokat feldolgozni. A Plus/4-es ehhez számos segítséget nyújt. Ahhoz, hogy ezeket ki is tudjuk használni, meg kell tanulni a helyes alkalmazásukat. Reméljük, hogy ehhez a szükséges alapokat ebben a fejezetben meg tudtuk adni.

15. A finomfelbontású grafika

A Commodore Plus/4-es számítógép egyik legnagyobb erőssége kétségkívül a grafikai képességeiben rejlik. A számítógép a képernyőn 320 × 200, tehát összesen 64000 képpontot külön-külön tud vezérelni. A képernyőre 16 féle szint tud varázsolni, ráadásul mindegyiket nyolc-nyolc különböző fényességi fokozatban.

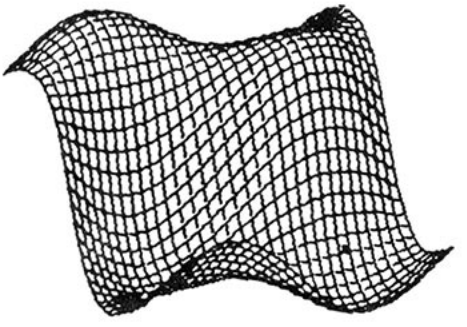
Ez azonban még nem minden. Lehetőség van szöveg és grafika egyidejű megjelenítésére is, amivel jócskán túltesz számos nagyobb és drágább számítógépen. Lehetőség van továbbá meghatározott alakzatok megrajzolására és tárolására is. Ezek a tulajdonságok egy ilyen árkategóriájú géphez képest igazán bámulatosak.

15.1. A KÉPERNYŐ FELÉPÍTÉSE

A bekapcsolást követően a Plus/4-es gép az úgynevezett szöveg (karakteres) üzemmódba kerül. Ez azt jelenti, hogy a képernyőre csak azok a karakterek írhatók ki, amelyek a billentyűzeten láthatók.

Ebben az üzemmódban a képernyőn 25 sor, és soronként 40 karakter fér el. Összesen tehát 25 × 40 = 1000 karakter ábrázolható. Egy karakter 8 × 8 képpontból áll.

A finomfelbontású grafikus üzemmódban (grafikus 1. üzemmód) ezeket a képpontokat külön-külön lehet vezérelni. Ekkor a képernyő vízszintes (X) irányban 40 × 8 = 320, függőleges (Y) irányban pedig 25 × 8 = 200 képpontra van felosztva. Ez összesen 320 × 200 = 64000 képpontot jelent. Ezzel már egészen szép grafikát lehet előállítani. Mutatunk erre egy példát is (15.1. ábra).



15.1. ábra Példa a grafikus ábrázolásra

Néha arra is szükség lehet, hogy egyidejűleg grafikát és szöveget is megjelenítsünk a képernyőn. A Plus/4-es géppel ez is megvalósítható. Ez a grafikus 2. üzemmód: a finomfelbontású grafika és szöveg, osztott képernyőn. A képernyő ilyenkor két részre van osztva: egy felső, 320 × 160 képpontot tartalmazó részre a grafika számára, és egy alsó részre, ahová öt sorban szöveg írható.

A finomfelbontású grafikus üzemmódnak sajnos van egy hátránya: egy 8 × 8-as pontmátrixon belül (lásd normál, szöveges üzemmód) csak kétféle szín használható. A Plus/4-esnek is vannak korlátai. Ennek ellenére a számítógéppel nagyon szép, színes grafikák készíthetők.

A Plus/4-es a 8 × 8-as pontmátrixonként csak két szín használatát megengedő finomfelbontású grafikus üzemmód mellett ismeri még az úgynevezett többszínű grafikus üzemmódot is (grafikus 3.). Ebben az üzemmódban egy-egy 8 × 8-as pontmátrixon belül négy szín használható. A több színért azonban durvább képernyőfelbontással kell fizetni. A többszínű grafikus üzemmódban a képernyő felbontása 160 × 200 képpont. Ezt a grafikát is lehet szöveggel kombinálni: ez a többszínű grafika és szöveg, osztott képernyőn (grafikus 4.).

15.2. PLUS/4-ES GRAFIKUS ÜZEMMÓDJAI

Üzemmód	Normál felbontás	Módosított felbontás (SCALE)	Megnevezés
0	25 × 40	karakter (szöveg)	Normál üzemmód
1	320 × 200	1024 × 1024 pont	Finomfelbontású grafika, grafikus 1.
2	320 × 160	1024 × 816 pont 5 sor szöveg	Finomfelbontású grafika és szöveg, grafikus 2.
3	160 × 200	1024 × 1024 pont	Többszínű grafika, grafikus 3.
4	160 × 160	1024 × 816 pont 5 sor szöveg	Többszínű grafika és szöveg, grafikus 4.

A grafika bekapcsolásához a

GRAPHIC 1
GRAPHIC 2
GRAPHIC 3
GRAPHIC 4

utasítást kell beírni, és a képernyőn máris láthatóvá válik a finomfelbontású grafika. A számítógép most a grafikus tár tartalmát írja a képernyőre. A tárban levő mindenkori adatok képpontokként vagy színként jelennek meg. Ha a gépet és a grafikus tárat éppen most kapcsoltuk be, akkor a tár tartalma teljesen véletlenszerű. Ezért láthatunk a képernyőn esetleg valamilyen tarka összevisszaságot. A grafikus tárat tehát törölni kell. Ezt megtehetjük akár már a grafikus üzemmód bekapcsolásakor, akár később, egy erre alkalmas utasítással. Ahhoz, hogy a grafikus tárat már a bekapcsoláskor töröljük, egy második paramétert is meg kell adni. Például:

GRAPHIC 4,1

A GRAPHIC 4 utasítással az osztott képernyőjű, többszínű grafikus üzemmódot kapcsoljuk be, az 1 paraméter hatására pedig ez a tár törlődik (lásd a grafikai utasításokról szóló alfejezetet is).

Egy másik lehetőség a képernyő törlésére a

SCNCLR

utasítás.

Mindkét eljárás használható a 0-ás grafikai üzemmódban, azaz a normál, karakteres üzemmódban is. Írjuk be a GRAPHIC 0,1 utasítást. A képernyő azonnal törlődik.

Foglaljuk össze: A grafikát a

GRAPHIC üzemmód (esetleg paraméter a törléshez)

utasítás kapcsolja be.

A grafikus üzemmódot úgy kapcsolhatjuk ki, hogy visszakapcsolunk a normál, a karakteres üzemmódra:

GRAPHIC 0 (esetleg paraméter a törléshez)

Vegyük figyelembe, hogy a finomfelbontású vagy a többszínű grafika a Plus/4-esben jelentős nagyságú tárhelyet foglal le. A képernyő minden egyes képpontját egy-egy bit tárolja. Ehhez jönnek még a színre vonatkozó információk. Összességében a finomfelbontású, ill. a többszínű grafika a tárban 10 kbyte helyet foglal el.

A grafika számára így lefoglalt 10 kbyte tárterülettel a grafikus üzemmód kikapcsolása után sem rendelkezhetünk szabadon. Ennek így is kell lennie ahhoz, hogy egy egyszer már megrajzolt grafikát egymás után többször is be, ill. ki lehessen kapcsolni anélkül, hogy újból meg kelljen rajzolni. Az így lefoglalt területet csak akkor szabadítsuk fel, ha a benne levő grafikára a későbbiekben már valóban nincs szükségünk.

A tárat a

GRAPHIC CLR

utasítás teszi ismét szabaddá.

Ezután a Plus/4-esnek megint a teljes tára a rendelkezésünkre áll. Valójában azonban a grafikus tár még mindig nem törlődött. A benne levő „képet" csak akkor rontjuk el ill. töröljük, ha erre a tárterületre betöltünk vagy beírunk egy programot. Ugyanez a hatása azoknak a változóknak, amelyekre a programunknak esetleg szüksége van.

15.3. A GRAFIKUS (PIXEL) KURZOR

A kurzorról már beszéltünk a korábbiakban. A képernyőn villogó, kis, négyzet alakú jelet nevezzük így. A kurzor mindig azon a helyen villog, ahová a következő karakter kerül.

A finomfelbontású grafikus üzemmódban is létezik kurzor, csak éppen nem látható. Ebben az üzemmódban a képernyő egy elemi képpontját angol rövidítéssel pixel-nek nevezik, innen származik a pixel-kurzor (PC) elnevezés. Mi a könyvben ezt grafikus kurzornak nevezzük.

De vajon miért nem látható a grafikus kurzor? A válasz egyszerű: amikor adatot vagy szöveget írunk a számítógépbe, akkor természetesen látnunk kell, hogy mit és hová írunk. A grafikát viszont nem mi magunk, hanem az általunk megírt program rajzolja a képernyőre. Természetesen vannak olyan alkalmazások is – például rajzoló programok – ahol a grafikus kurzort esetleg láthatóvá kell tenni.

A számítógép belül tárolja a grafikus kurzor pozícióját. Ezt a tárolt pozíciót egy BASIC függvénnyel lekérdezhetjük. A függvény

RDOT(n)

ahol n = 0, 1 vagy 2 lehet.

n = 0 az X irányú pozíciót adja,
n = 1 az Y irányú pozíciót adja,
n = 2 az egy utasításban utoljára megadott színzóna számát adja.

Amint látjuk, a számítógép a színzóna számát a grafikus kurzor pozíciójával együtt tárolja. Mire jó ez?

Ha egy programban egymás után sok grafikai utasítást kell feldolgozni, akkor minden egyes alkalommal meg kellene adni a színzónát, az X és Y kezdő- és végpozícióját. Ezt a munkát azonban jelentős mértékben le lehet egyszerűsíteni, mivel a grafikus kurzor mindig a végértékeket tárolja. Így ezek az értékek a következő utasítás számára egyszersmind kezdő értékekként is szolgálhatnak. Ezt az egyszerűsítést a programozási gyakorlatban úgy valósítjuk meg, hogy az utasításokból ezeket az értékeket egyszerűen kihagyjuk. Nézzünk erre egy példát:

```
10 GRAPHIC1,1
20 DRAW1,10,10 TO 20,20
30 DRAW1,20,20 TO 30,10
40 DRAW1,30,10 TO 40,20
```

RUN

Kapcsoljuk ki a grafikus üzemmódot:

GRAPHIC 0

Változtassuk meg most a 30-as és 40-es sorokat a következők szerint:

```
30 DRAW TO 30,10
40 DRAW TO 40,20
```

RUN

Ugyanazt a grafikát kapjuk, mint az előző példában. A második programban viszont a kezdő pozíciókat és a színzónát egyszerűen kihagytuk.

Mellékesen megjegyezve a DRAW utasítás írásmódja még tovább egyszerűsíthető. Beírhatjuk például a következőt is:

```
10 GRAPHIC1,1
20 DRAW1,10,10TO20,20TO30,10TO40,20
```

Itt csak a végpozíciókat adtuk meg.

Térjünk még egy kicsit vissza a grafikus kurzorra. Arra is van lehetőség, hogy a grafikus kurzort a képernyő tetszőleges helyére vigyük (láthatatlanul). Ezt a következő utasítás végzi:

LOCATE x,y

Az x és az y a grafikus kurzor pozícióját határozza meg. Próbáljuk ki ezt egy programon keresztül:

```
10 GRAPHIC1,1
20 LOCATE10,10
30 DRAWTO20,20TO30,10TO40,20
```

RUN

A LOCATE 10,10 utasítás hatására a grafikus kurzor az X = 10 és Y = 10 pozícióba kerül. Ezért lehet a DRAW utasításból a kezdő pozíciót elhagyni.

Bemutatunk egy kis példaprogramot, amely a Commodore Plus/4-es számítógépnek ezeket a nagyszerű lehetőségeit használja ki. Bizonyára mindenki ismeri azt az órát, amelyiket a tv-ben az esti tv-híradó előtt lehet látni. Egy ilyen óra megrajzolása és működtetése igazán érdekes programozói feladat. A gép a feladattal egyszerűen megbirkózna, ha lenne erre megfelelő program. Most segítsünk ezen. A következő oldalakon megtalálható a program ismertetése, és a teljes programlista.

15.4. PÉLDAPROGRAM: TÉVÉÓRA

A feladat első pillantásra egyszerűbbnek látszik, mint amilyen valójában: a programot BASIC nyelven kell megírni.

```
10 REM TEVEORA
20 DIM HX%(60),HY%(60),MX%(60),MY%(60),SX%(60),SY%(60)
30 X=150:Y=100:RA=0:VOL8
40 COLOR0,1
50 COLOR1,2
60 COLOR4,1
70 PRINTCHR$(147)
80 INPUT"AZ IDOPONT MOST (OOPPSS)";TI$
90 PRINT"KIS TURELMET..."
100 REM AZ ORAMUTATO ADATAINAK SZAMITASA
110 FORW=450TO96STEP-6
120 RE=40:GOSUB330
130 HX%(T)=X2:HY%(T)=Y2
140 RE=60:GOSUB330
150 MX%(T)=X2:MY%(T)=Y2
160 RE=70:GOSUB330
170 SX%(T)=X2:SY%(T)=Y2
180 T=T+1:NEXT
190 REM AZ ORA RAJZOLASA
200 GRAPHIC1,1
210 RA=71:RE=80:FORW=450TO96STEP-6:GOSUB330:DRAW,X1,Y1TOX2,Y2:NEXT
220 RA=81:RE=85:FORW=450TO96STEP-30:GOSUB330:DRAW,X1,Y1TOX2,Y2:NEXT
230 DRAW0,X,YTOMX%(M%),MY%(M%):M%=VAL(MID$(TI$,3,2)):
DRAW,X,YTOMX%(M%),MY%(M%)
240 DRAW0,X,YTOHX%(H%),HY%(H%)
```

```
250 H%=(5*VAL(LEFT$(TI$,2)))+INT(M%/12):IFH%>59THENH%=H%-60
260 DRAW,X,YTOHX%(H%),HY%(H%)
270 S%=VAL(MID$(TI$,5,2)):IFS%<>S1%THEN280:ELSE270
280 DRAW0,X,YTOSX%(S1%),SY%(S1%)
290 DRAW,X,YTOSX%(S%),SY%(S%):SOUND1,900,1
300 IFS%=0ORS1%=M%THENS1%=S%:GOTO230
310 IFS1%=H%THENS1%=S%:GOTO240:ELSES1%=S%:GOTO270
320 REM SZOGFUGGVENYEK
330 X1=X+INT(COS(W*3.14/180)*RA)
340 X2=X+INT(COS(W*3.14/180)*RE)
350 Y1=Y-INT(SIN(W*3.14/180)*RA)
360 Y2=Y-INT(SIN(W*3.14/180)*RE)
370 RETURN
```

A programban az idő méréséhez a Plus/4-es beépített óráját használjuk, amelyet előzőleg természetesen be kell állítani. Az első probléma az, hogy a programnak valamilyen módon minden másodpercben kapnia kell egy jelet ahhoz, hogy a másodpercmutató szépen szabályosan, és pontosan forogjon. Ezt BASIC nyelven igazán jól sajnos nem lehet megvalósítani. Gépi kódban ki lehetne használni a gép megszakításait (interrupt). A feladatot azonban BASIC nyelven kell megoldani, tehát nincs más választás, mint hogy a programban folyamatosan lekérdezzük a belső órát, hogy megállapíthassuk, elmúlt-e már egy másodperc. Ez a megoldás ugyan nem túl elegáns, de használható.

15.2. ábra „TÉVÉÓRA" példa

A második problémát az órának a képernyőre való rajzolása jelenti. A Plus/4-esnek ugyan nagyon hatékony grafikai utasításai vannak, de a perceket jelölő 60, és az órákat jelölő 12 vonalkának a megrajzolása azért mégsem olyan egyszerű dolog.

A harmadik probléma az előző kettőhöz képest valóban jóval nehezebb. Ezt a problémát a program futási sebessége jelenti. Természetesen erre is van megoldás, hiszen különben nem szerepelne a könyvben ez a példaprogram.

Írjuk be a gépbe a programot. A program néhány, meglehetősen hosszú programsort, és igen sok változót tartalmaz, ezért a beíráshoz nagy figyelem szükséges. A programot karakterről karakterre, nagyon pontosan írjuk be, és indítás előtt tároljuk.

Miután a programot tároltuk, indítsuk el RUN-nal. A program felszólít bennün-

ket arra, hogy írjuk be az aktuális időt. Ehhez egy megjegyzés: mindig két-két számmal kell megadni az órát, percet, másodpercet. Például:

Az aktuális idő:	A beírás alakja:
9 óra 15 perc, 0 másodperc	091500
21 óra 46 perc, 20 másodperc	214620

A számítógép kb. 30 másodpercig számol, majd a képernyőn megjelenik az óralap és a mutatók.

A programot a RUN/STOP billentyűvel lehet megállítani. Ezt követően mindig írjuk be a GRAPHIC 0 utasítást (grafika ki, normál üzemmód be).

Azt mondtuk, legyen a programunk elég gyors ahhoz, hogy a másodpercmutatót elfogadhatóan rövid idő alatt megrajzolhassa. Ez azonban még nem elegendő, hiszen a perc- és az óramutatót is egy másodpercen belül kell tovább léptetni. Ennek olyan gyorsan kell megtörténnie, hogy az egyes képpontok „továbblépése" közötti időbeli eltolódást – legalábbis szemmel – ne lehessen érzékelni. Ehhez jön még, hogy a Plus/4-es gazdag grafikai lehetőségei ellenére sem egyszerű feladat, hogy egy középpontból 6 fokos osztásokban azonos hosszúságú vonalakat rajzoljunk.

A legegyszerűbb az, hogy a mutatók lehetséges 60 pozícióját már a tulajdonképpeni óraprogram indítása előtt kiszámítjuk. Ekkor ugyanis a (gyors) DRAW utasítással megrajzolhatjuk a mutatókat a középpont és a tárolt pozíciók között. Ezt a megoldást választottuk a programunkban is. Ezért van szüksége a számítógépnek a program indításakor a 30 másodperces „gondolkodási" időre. Az X és az Y értékeihez egész típusú változókat (integer) használunk (ezt a % jel jelöli). A számítógép az egész típusú változókat ugyanis gyorsabban dolgozza fel.

A 20-as sorban a mutatók pozícióit tartalmazó, egész típusú változókat dimenzionáljuk. Az óramutató pozíciói a HX% és HY% változókba, a percmutató pozíciói az MX% és MY%, a másodpercmutató pozíciói az SX% és SY% változókba kerülnek. Mivel az óralap 60-as felosztású, a dimenzionálást is 60-60 pozícióra végezzük.

A 30-as sorban meghatározzuk az óra középpontjának koordinátáit. Az RA változó a körgyűrű belső sugarát tartalmazza – most egyelőre a mutatók részére, amelyek természetesen a kör középpontjából indulnak ki, ezért az RA értéke 0.

A 40-60-as sorokban beállítjuk a háttér, a rajzolás és a képernyőkeret színét.

A 70-es sorban töröljük a képernyőt.

A 80-as sorban az aktuális időt beírjuk a TI$ rendszerváltozóba.

Mivel a számítások kb. 30 másodpercig tartanak, szükségesnek tartottuk a 90-es sor beírását.

Az alprogramban (a 330-as sortól kezdődően) használni kell a SIN és COS szögfüggvényeket, hogy a kör kerületén levő 60 mutatóállást kiszámíthassuk.

A 110-es sorban a kiszámítandó szögeket rendre átvesszük a W változóba. Az senkit se zavarjon, hogy a szögértékek 450 és 96 között vannak, és ezeket −6 lépésközökkel csökkentjük. Ennek az az egyszerű oka, hogy az óramutató jobbra forog, és a 12 óra az óralapon felül van.

A számításhoz feltétlenül szükség van a mutatók hosszára, amit az RE és az RA változók különbsége ad meg. Mivel RA = 0, a mutatók hosszát közvetlenül az RE tartalmazza. Az óramutató hossza legyen 40 képpont, a percmutatóé 60, a másodpercmutatóé pedig 70 képpont.

Az egész típusú változók index-számát a T változó tartalmazza, amelyet a 180-as sorban a ciklus minden egyes lefutásakor 1-gyel megnövelünk.

A 200-as sorban bekapcsoljuk a finomfelbontású grafikát.

A 210-es sor – ugyancsak a szögfüggvények segítségével – megrajzolja a percbeosztás 60 vonalkáját.

A 220-as sor az órákat jelölő vonalakat rajzolja meg.

Most következik a tulajdonképpeni időmérő óra. A 230–310-es sorokban lekérdezzük a TI$ rendszerváltozót, és ennek megfelelően megrajzoljuk az óramutatókat. Ezután a program a 270-es sorban arra vár, hogy az óra 1 másodpercet „továbbmenjen".

Ekkor a 280-as sor törli a régi másodpercmutatót, és a 290-es sor megrajzolja az újat. Beírtunk ebbe a sorba egy SOUND utasítást, hogy az óra „ketyegjen" is.

A 300-as és a 310-es sor megvizsgálja, hogy a másodpercmutató a 12-n áll-e, és ha igen, akkor lépteti a percmutatót.

Megjegyzés: Később, amikor majd megismerkedünk a BASIC nyelv CIRCLE utasításával, felmerülhet az a kérdés, hogy az óraprogramban miért kell ilyen „körülményesen", szögfüggvényekkel dolgoznunk. A kérdés csak annyiban jogos, hogy a CIRCLE utasítással valóban fel lehet rajzolni egy kör kerületére pontokat, és így az óralapot néhány körrel valóban el is lehetne intézni. Magának az időmérő órának a programozása azonban ugyanilyen bonyolult maradna, és gyorsabb se lenne. Ezért választottuk ezt a megoldást.

Az óralap alakját könnyen meg lehet változtatni. Hol van az előírva, hogy egy órának kör alakúnak kell lennie? Próbáljunk ki néhány változatot:

```
330 X1=X+INT(COS(W*3.14/180)*RA*2)
340 X2=X+INT(COS(W*3.14/180)*RE*2)
```

vagy

```
330 X1=X+INT(COS(W*3.14/180)*RA*2)
340 X2=X+INT(COS(W*3.14/180)*RE*1.5)
```

vagy

```
330 X1=X+INT(COS(W*3.14/90)*RA)
340 X2=X+INT(COS(W*3.14/90)*RE)
350 Y1=Y-INT(SIN(W*3.14/90)*RA
360 Y2=Y-INT(SIN(W*3.14/90)*RE
```

vagy

```
340 X2=X+INT(COS(W*3.14/180)*RE*2)
```

vagy

```
330 X1=X+INT(COS(W*3.14/180*2)*RA)
340 X2=X+INT(COS(W*3.14/180*2)*RE)
350 Y1=Y-INT(SIN(W*3.14/180/2)*RA)
360 Y2=Y-INT(SIN(W*3.14/180/2)*RE)
```

Találjunk ki további változtatásokat is. Nyugodtan kísérletezhetünk. Jó szórakozást a tv-órához!

15.5. A KÉPERNYŐ FELOSZTÁSÁNAK MEGVÁLTOZTATÁSA – A SCALE UTASÍTÁS

A Plus/4-es egyik különlegessége a SCALE utasítás. Ezzel az utasítással a képernyőt finomabban, mind vízszintes, mind függőleges irányban 1024 pontra lehet felosztani. Ez természetesen csak a számításokra vonatkozóan igaz, hiszen a képernyő felosztása a valóságban továbbra is „csak" 320x200 képpont.

A megváltoztatott felosztást a

SCALE 1

utasítás kapcsolja be, és a

SCALE 0

utasítás kapcsolja ki.

Írjuk be az alábbi programot, hogy lássuk a SCALE hatását:

```
10 COLOR 0,1
20 COLOR 1,2
30 GRAPHIC 1,1
40 CIRCLE 1,100,100,50
50 CHAR 1,20,24,"BILLENTYUT LENYOMNI",1
60 GETKEY A$
70 GRAPHIC 0
80 END
```

A programot elindítva a képernyőn egy közönséges kört látunk. Nyomjunk le egy billentyűt, és írjuk a programba ezt a sort:

35 SCALE 1

Ha most ismét elindítjuk a programot, akkor sokkal kisebb kört kapunk. Annak következtében, hogy most a képernyő mind az X, mind az Y irányban 1024 pontra van felosztva, a 100 sugarú körünk most a képernyő bal felső részén jelenik meg.

Lássuk röviden a programunkat:

A 10-es sorban beállítottuk a háttér színét (itt feketére).

A 20-as sorban a képpontok (azaz a rajzolás) színét állítottuk be (itt fehérre).

A 30-as sor bekapcsolja a finomfelbontású grafikát, és törli a képernyőt.

A 40-es sor egy kört rajzol. A kör középpontjának koordinátái: X = 100 és Y = 100. A kör sugara 50.

Az 50-es sorral szöveget íratunk ki a finomfelbontású képernyőre. A szöveg a 24-es sorban, a 20-as oszlopban kezdődik.

A 60-as sor egy billentyű lenyomására vár.

Ha lenyomtunk egy billentyűt, akkor a 70-es sor törli a grafikát.

A 35-ös sor beszúrásával megváltoztattuk a képernyő felosztását. A kör most sokkal kisebb lesz.

Ahhoz, hogy a módosított felbontású képernyőn ugyanazokat a koordinátákat kapjuk mint a normál felbontásnál, az X és az Y koordinátákat át kell számítani:

X * 3,2 = módosított X koordináta
Y * 5,12 = módosított Y koordináta
X * 6,4 = módosított X koordináta a többszínű üzemmódban

A többszínű üzemmódban az X értékét 6,4-del kell megszorozni, mert ebben az üzemmódban az X irányban csak 160 képpont ábrázolható. Tapasztalataink szerint a SCALE utasítást használva a grafika elkészítése valamelyest tovább tart (kb. 0,5–2,5%-kal).

Ennyi a SCALE utasításról. Ki lehet próbálni ezt az utasítást a tv-óra programban is. A módosított felosztást még az óra megrajzolása előtt be kell kapcsolni. Ehhez be kell írni egy új programsort:

```
205 SCALE 1
```

15.6. A GRAFIKAI UTASÍTÁSKÉSZLET

Az eddigi példaprogramjaink során már megismerkedtünk a Plus/4-es néhány grafikai utasításával. A teljes grafikai utasításkészlet azonban jóval bővebb. A Plus/4-es BASIC nyelve számos, igen hatékony grafikai utasítást ismer. Itt most felsoroljuk, és röviden ismertetjük a Plus/4-es valamennyi utasítását, amelyek a finomfelbontású és a többszínű grafika programozásával összefüggnek.

Néhány utasításhoz egy vagy több, további paramétert is meg kell adni. A paraméterekről elegendő annyit tudni, hogy ezek az utasításokhoz tartozó olyan járulékos információkat adnak meg, amelyekre a számítógépnek az utasítás végrehajtásához szüksége van. Paramétereknek változókkal is lehet értéket adni.

Vannak olyan utasítások is, amelyeknél bizonyos paraméterek el is hagyhatók. Ilyen esetben a hiányzó paramétereknek a számítógép saját maga ad értéket (többnyire 0-t vagy 1-et, 360 fokot).

Az utasítások ismertetésénél a paramétereket betűkkel vagy névvel adjuk meg. Minden egyes utasításnál megadjuk, hogy az egyes paraméterek milyen értékeket vehetnek fel. Azokat a paramétereket, amelyek megadása nem kötelező, zárójelbe tettük. Néhány utasításnál az úgynevezett színmezőt is meg lehet, vagy meg kell adni.

15.7. A SZÍNMEZŐK

Öt színmező van:

0 = háttér
1 = írás (karakterek)
2 = többszínű grafika I.
3 = többszínű grafika II.
4 = képernyőkeret

Mindegyik színmezőben egy-egy szín tárolható. Ezt a COLOR utasítás végzi el. Ezzel az utasítással határozhatjuk meg, hogy milyen színeket használjunk a grafikánkban.

Szintaxis:

COLOR színmező, szín (, fényesség)

Példa:

COLOR 1,2,4

(1-es színmező, 2-es szín, 4-es fényesség)

Ennek az utasításnak a hatására a képernyőre kerülő írás színe szürke lesz. A kurzoron már látszik is ez a szín.

COLOR 4,3,7

Most a képernyőkeret színe világospiros lesz.

Az alábbi példaprogram segítségével megnézhetjük a Plus/4-es teljes színskáláját. Egyidejűleg az is vizsgálható, hogy mennyire jól (vagy rosszul) adja vissza a színeket a számítógéphez csatlakoztatott tévékészülék (vagy monitor).

```
10 FOR LU = 0 TO 7
20 FOR T = 1 TO 2
30 FOR CO = 1 TO 16
40 COLOR 1,CO,LU
50 PRINT CHR$(18);"  ";
60 NEXT CO
70 PRINT
80 NEXT T
90 NEXT LU
100 COLOR 1,2
```

A 10, 20, 30, 60, 80 és 90-es sorok ciklusokat képeznek, amelyek a fényességet, a színeket, és a fényességi értékekként ábrázolandó sorok számát adják meg.

A 40-es sor az írás (karakterek) színét határozza meg.

Az 50-es sor inverz üzemmódban két üres karaktert (szóközt) ír ki. Az inverz üzemmódot a CHR$(18) kapcsolja be.

A 100-as sor visszakapcsolja a normál szint.

A Plus/4-es BASIC nyelvében van két utasítás, amelyekkel az egyes színmezők éppen beállított értékét lehet lekérdezni.

Az RCLR és az RLUM függvény

Szintaxis:

RCLR (színmező)

A zárójeleket is be kell írni! Ezzel az utasítással megtudhatjuk a megadott színmezőhöz rendelt szín számát. Vigyázzunk, ez az utasítás közvetlen üzemmódban, így beírva a

?SYNTAX ERROR

hibaüzenetet küldené. Ha a függvényt használni akarjuk, akkor eléje még egy utasítást is kell írni, pl. PRINT.

Szintaxis:

RLUM (színmező)

A függvény segítségével a megadott színmezőben levő szín fényességének az értékét tudhatjuk meg.

Fontos: A COLOR utasítás, RCLR és az RLUM függvényekhez minden esetben meg kell adni a színmezőt. A többi utasításnál elhagyható ez a paraméter. Ilyenkor a számítógép a paraméternek az 1 értéket adja.

A BOX utasítás

Szintaxis:

BOX (színmező),x1,y1,x2,y2(,szög) (,festés)

A BOX utasítás téglalapot rajzol.

Az egyes paraméterek jelentései:

színmező (0–3)	Ha nem adjuk meg, akkor a gép az 1 értéket rendeli hozzá (az írás színe). Az utasítás a téglalapot azzal a színnel rajzolja meg, amelyet egy COLOR utasítás előzőleg kijelölt.
x1,y1	Bal felső sarok koordinátái.
x2,y2	Jobb alsó sarok koordinátái.
szög	Az utasítás a téglalapot a (középpont körül) a megadott szöggel elforgatva rajzolja meg. Ha nem adunk meg értéket, akkor az elforgatás szöge = 0 fok.
festés	A téglalapnak megrajzolható csak a körvonala (ha ezt a paramétert nem adjuk meg, vagy 0 értéket adunk), és a teljes területe be is „festhető" (ha a festés = 1). Alapérték 0

Nézzünk egy példát a BOX utasításra:

```
10 GRAPHIC1,1
20 BOX,100,50,200,150
30 BOX,100,50,200,150,45,1
```

A számítógép az első téglalapot normál módon, a megadott koordináták szerint rajzolja meg. A téglalapot nem festi be.

A második téglalapot viszont a koordinátákból adódó középpont körül 45 fokkal elforgatva és befestve rajzolja meg.

A CHAR utasítás

A CHAR utasítással bármelyik grafikus üzemmódban (karakteres üzemmódban is) a képernyő tetszőleges helyére kiírható valamilyen szöveg. Vigyázat! Ne cseréljük össze a CHAR utasítást a CHR$ utasítással!

Szintaxis:

CHAR (színmező),x,y,szöveg(,inverz)

Az egyes paraméterek jelentése:

színmező (0-3)	Mint az előzőnél. Ha nem adjuk meg, akkor az 1-es színmező (írás).
x	Az első karakter X koordinátája (mint a karakteres üzemmódban az oszlop száma 0-39 között).
y	Az Y koordináta (a sor száma 0-24 között).
szöveg	A kiírandó szöveg kifejezés

inverz	Ha inverz = 1 akkor inverz kiírás, ha inverz = 0 vagy ha nem adjuk meg, akkor normál kiírás.

Ez az utasítás nagy segítséget jelent a képernyőre való íráshoz. Ez nem csak a grafikus, hanem a szöveges üzemmódban is igaz. Enélkül az utasítás nélkül különböző, körülményes manipulációkat kellene végezni a kurzorvezérlő billentyűkkel és a PRINT utasításokkal ahhoz, hogy egy szöveget a képernyő tetszőleges helyére kiírathassunk. Ezt a munkát a CHAR utasítás lényegesen leegyszerűsíti. Egészen egyszerűen, a CHAR után meg kell adni azokat a koordinátákat, ahol a szövegnek meg kell jelennie, és a szöveg máris ott van. A CHAR utasítás azonban nem használható úgy, mint a PRINT utasítás. Numerikus változók például nem írathatók ki (de STR$-ral szöveggé átalakítva már igen!). Használhatunk egy trükköt is:

Állítsuk a kurzort a CHAR utasítással a képernyő kívánt helyére, és dolgozzunk tovább a PRINT-tel. Például:

```
CHAR,10,10,"":PRINT"TIZES SOR, TIZES OSZLOP"
```

Ezzel a kurzort a kívánt pozícióba vittük (X = 10, Y = 10). A CHAR utáni idézőjelek közé semmit sem írtunk, tehát nem is íródik ki semmi. A kurzor marad a megadott helyen. Egy ezután következő PRINT utasítás most ettől a helytől kezdődően ír a képernyőre.

A CHAR utasításban szereplő utolsó paraméter, amely az inverz üzemmódot kapcsolja be, karakteres üzemmódban (GRAPHIC 0) nem használható. Ezzel szemben viszont a CHAR utasításba beírható valamennyi, a karakteres üzemmódban használható vezérlő karakter (tehát a REVERSE ON is). Például:

```
CHAR,10,10,CHR$(18):PRINT"TIZES SOR, TIZES OSZLOP"
```

vagy karakterláncok összekapcsolásával

```
CHAR,10,10,CHR$(18)+"TIZES SOR, TIZES OSZLOP"
```

A CIRCLE utasítás

A CIRCLE utasítással köröket, ellipsziseket, körcikkeket vagy sokszögeket rajzolhatunk.

Szintaxis:

CIRCLE(színmező),(x,y),xr,(yr),(ksz),(vsz),(elfsz),(szegsz)

Az egyes paraméterek jelentése:

színmező	Mint az előzőekben.
x,y	A középpont koordinátái. Ha nem adjuk meg, akkor a grafikus kurzor értékeit veszi fel.
xr	X irányú sugár.
yr	Y irányú sugár. Ha nem adjuk meg, akkor megegyezik az xr értékével.
ksz	A kezdőszög. Ha nem adjuk meg, akkor 0 fok.
sz	A végszög. Ha nem adjuk meg, akkor 360 fok.
elfsz	Az alakzat elforgatása a megadott szöggel.
szegsz	Két szegmens közötti szög. Ha nem adjuk meg, akkor a szög értéke 2 fok.

A körcikk kezdőszögével kapcsolatban tudni kell, hogy a 0 fok ill. a 360 fok felül van. A gép jobbra, az óramutató járásával megegyező irányban rajzol. Ha tehát a kezdőszögre 0-nál nagyobb értéket adunk meg, akkor az ennek megfelelő körcikk az indulási helyhez képest jobb oldalról fog hiányozni. Ugyanígy viselkedik a végszög is. Ha az elforgatás szögére nullánál nagyobb értéket adunk meg, akkor a 0 fokot jelölő pont, tehát a körünk kezdőpontja az elforgatási szög értékével jobbra fordul el.

Lássunk néhány példát a CIRCLE utasításra:

CIRCLE 1,100,100,100,100

Ez az utasítás az X = 100 és Y = 100 koordinátákkal megadott középpont körül, az X irányban 100-as és az Y irányban 100-as sugárral, az 1-es színmező színével egy kört rajzol. A

CIRCLE,100,100,100

utasítás ugyanazt a kört rajzolja meg, mivel a színmezőt nem kell megadni, ha ez az 1-es. Ugyanígy nem kell megadni az Y sugarat, ha az az X sugárral megegyezik.

CIRCLE,100,100,100, ,180

Az utasítás egy körcikket rajzol az X = 100 és Y = 100 koordinátájú középpont körül 100-as sugárral, és a 180 fokos szögtől kezdődően.

CIRCLE,100,100,100,50,270,90

Az utasítás egy ellipszisnek a 270-90 fok közötti részét rajzolja meg.

CIRCLE,100,100,100,50, , , ,90

Az utasítás egy rombuszt rajzol.

CIRCLE,100,100,100, , , ,45,90

Az utasítás egy négyzetet rajzol.

Az már ezekből a példákból is megállapítható, hogy a CIRCLE rendkívül hatékony és sokoldalú utasítás. A sok paraméter ugyan egy kicsit bonyolulttá teszi a használatát, de az elérhető eredmény bőséges kárpótlást nyújt.

15.8. PÉLDAPROGRAM A FINOMFELBONTÁSÚ GRAFIKÁRA

Írjuk gépbe az alábbi programot, és indítsuk el RUN-nal.

```
10 COLOR0,3,0
20 COLOR1,3
30 GRAPHIC1,1
40 RA=30
50 XA=120
60 GOSUB100
70 XA=200
```

```
80 GOSUB100
90 END
100 FORKR=0TO360STEP5
110 X=XA+INT(COS(2*3.14*KR/360)*3*RA)
120 Y=100-INT(SIN(2*3.14*KR/360)*2*RA)
130 CIRCLE,X,Y,RA
140 NEXTKR
150 RETURN
```

A 10-es és a 20-as sorokban meghatározzuk a háttér és az írás színeit.

A 30-as sorban bekapcsoljuk a finomfelbontású grafikát.

A 40-es és az 50-es sorokban az RA változóhoz 30-at, az XA változóhoz pedig 120-at rendelünk (RA a körök sugara, XA az első ellipszis középpontja).

A 60-as sorban meghívjuk a 100-as sorban kezdődő alprogramot.

Az alprogramból való visszatérés után a 70-es sorban új értéket rendelünk az XA-hoz.

A 80-as sorból ismét az alprogramra ugrunk.

A 90-es sorban befejezzük a programot.

A 100, 110, 120 és 140-es soroknak gyakorlatilag ugyanaz a hatásuk, mint a CIRCLE,120,100,90,60, , , ,5 utasításnak. Látszik, hogy milyen nehéz CIRCLE nélkül ellipszist rajzolni.

A 130-as sorban rajzoljuk meg az egyes kis köröket.

A 150-es sorban az alprogramból visszaugrunk a főprogramba.

Ebben a programban is használjuk a szögfüggvényeket az ellipszisek számításához. Elméletileg most is lehetne dolgozni a CIRCLE utasítással, de ez megnövelné a programozási munkát.

A DRAW utasítás

Ezzel az utasítással már találkoztunk néhányszor. A DRAW használatával vonalak rajzolhatók a finomfelbontású grafikai üzemmódban.

Szintaxis:

DRAW (színmező) (,x1,y1) (TOx2,y2) (TOx3,y3) ...

Az egyes paraméterek jelentése:

színmező	Mint az előzőekben.
x1,y1	A kezdőpont X és Y koordinátái.
TOx2,y2	A végpont X és Y koordinátái, ha vonalat kell rajzolni.
TOx3,y3	Egy x2 és y2 kezdőpontú egyenes X és Y koordinátái.

A DRAW utasítással tehát egyedi pontok ábrázolhatók, és egy vagy több vonal rajzolható.

DRAW 1,100,100 pont
DRAW 160,80, TO 300,90 vonal

A PAINT utasítás

A PAINT utasítással zárt felületek, azaz képpontokkal körülhatárolt felületek a megadott színmező színére „befesthetők".

Szintaxis:

PAINT (színmező),[x,y](,m)

Az egyes paraméterek jelentése:

színmező	Mint az előzőekben. Alapeset: 1.
x,y	A zárt felületen belül levő valamely pont koordinátái, amely körül a festés történik (alapeset pixel cursor).
mód	Üzemmód. Ha a módot nem adjuk meg, vagy mód = 0, akkor a felületet a színmezőben levő színre festi be. m = 1 : ha a megadott színmezőben a háttér színe van megadva, akkor a felületet ugyan befesti, de a kigyújtott képpontok olyan színűek lesznek, hogy azokat a háttértől meg lehessen különböztetni.

Példa:

```
10 GRAPHIC1,1
20 CIRCLE,100,100,100
30 PAINT1,100,100
```

A SCNCLR utasítás

A SCNCLR utasítás törli a képernyőt. Ez a képernyőtörlő utasítás bármelyik grafikus üzemmódban – tehát karakteres módban is – használható.

15.9. TÁROLHATÓ ALAKZATOK (SHAPES)

Ebben az alfejezetben a Plus/4 számítógép egyik speciális lehetőségéről, a tárolható alakzatokról lesz szó. Mit kell értenünk a tárolható alakzatok alatt? Ezek az alakzatok a grafikus képernyő olyan, téglalap alakú felületei, amelyek a számítógép tárában, karakteres változókként tárolhatók. Ezek a karakteres változók a tárban bármikor elolvashatók.

Másképpen fogalmazva: lehetőségünk van arra, hogy a grafikus képernyő egy-egy részét beírjuk egy-egy változóba, majd ezeket a változókat a későbbiekben bármikor elolvashatjuk, és a tartalmukat a képernyőn megjeleníthetjük.

Az elmondottak megvalósítására két utasítás létezik:

SSHAPE és GSHAPE

Nézzünk meg ezekkel kapcsolatban egy programot:

```
10 COLOR1,1
20 COLOR0,2
30 GRAPHIC1,1
40 CIRCLE1,20,20,10
50 DRAW1,10,10TO30,30
60 DRAW1,10,30TO30,10
70 SSHAPE A$,10,10,30,30
80 SCNCLR
90 FORA=0TO24
100 CHAR1,0,20,"GSHAPE A$,X,Y,"+STR$(A)
110 DRAW1,0,0TO100,100
120 FORX=1TO100STEP15
130 FORY=1TO100STEP10
140 GSHAPE A$,X,Y,A
150 NEXTY:NEXTX:SCNCLR:NEXTA
160 GRAPHIC0
```

A 10, 20 és 30-as sorokban beállítjuk a színeket, és bekapcsoljuk a grafikát.

A 40, 50 és 60-as sorokban egy körből és két egyenesből álló alakzatot rajzolunk.

A 70-es sorban ezt az alakzatot beírjuk az A$ változóba. Az erre szolgáló utasítás a SSHAPE.

A 80-160 sorokban töröljük a képernyőt, és az A$ változó tartalmát a GSHAPE utasítással a különböző módokban ismét képernyőre írjuk.

A 110-es sorban egy átlót rajzolunk a képernyőre, hogy lássuk, miként változtatják az alakzatok a képernyő aktuális tartalmát.

A programunkkal az alakzatot tehát az A$ változóban tároljuk. Ezt az alakzatot a képernyő bármely részén újra megjeleníthetjük. Azt is látjuk, hogy ez a megjelenítés különböző módon történhet. Vizsgáljuk meg ehhez közelebbről a SSHAPE és a GSHAPE utasításokat.

A SSHAPE utasítás

Szintaxis:

SSHAPE karakterváltozó,x1,y1 (,x2,y2)

karakterváltozó	bármilyen változónév használható
x1,y1	az alakzat egyik sarkának koordinátái
x2,y2	az alakzat másik sarkának koordinátái (átló másik végpontjai). Ha ezeket nem adjuk meg, akkor az utasítás a grafikus kurzor koordinátáit helyettesíti be ide.

A SSHAPE utasítással egy alakzatot, tehát a grafikus képernyő egy részét tárolhatjuk. Mivel az alakzatot egy karakterváltozó tárolja, és egy karakterváltozóba maximum 255 karakter írható be, természetesen a tárolható képernyőrész is korlátozott nagyságú.

A grafikus képernyőn egy byte-on nyolc képpont ábrázolható. Azt tudjuk, hogy a grafikus képernyő 64000 képpontból, többszínű üzemmódban pedig 32000 képpontból áll. Ekkora mennyiséget 255 byte-on természetesen nem lehet tárolni. Ehhez jön még, hogy a karakterváltozónkba szigorúan véve csak 251x8 képpont „fér" bele,

mivel a négy utolsó karakter az alakzatunk hosszúságára és szélességére vonatkozó adatokat tárolja. Elméletileg tehát 251x8 képpontból álló alakzatokat lehet tárolni. A gyakorlatban azonban még ez sem igaz: a tárolható képpontok száma maximum 2000. A tárolható képpontok száma függ az alakzat formátumától, azaz a szélesség és a hosszúság egymáshoz képesti arányától. Így például egy X = 200 és Y = 10 méretű alakzat tárolható, míg egy 100 képpont hosszú és 20 képpont széles alakzat nem. Utóbbi esetben a számítógép a

?STRING TOO LONG ERROR

hibaüzenetet küldené (karakterlánc túl hosszú). Ennek az az oka, hogy a karaktérváltozó rendre nemcsak az alakzat pontjait tárolja, hanem olyan, további információkat is, amelyek az alakzat formátumára vonatkoznak. A Commodore cég a kézikönyvében ezzel kapcsolatban két képletet közöl, amelyekkel megállapítható, hogy egy bizonyos alakzat belefér egy karaktérváltozóba vagy sem.

Nekünk az a véleményünk, hogy ezek a hosszú képletek igazából nem könnyítik meg a munkát. Ahelyett, hogy ezeket a képleteket használnánk, egyszerűen próbáljuk ki, hogy az illető alakzat milyen hosszú és milyen széles lehet. Erre természetesen egy program is írható. A Plus/4-es a TRAP utasítással lehetőséget ad arra, hogy egy olyan hibaüzenetre, mint a fenti STRING TOO LONG ERROR a célnak megfelelően reagáljunk. Ez az jelenti, hogy egy program egy hiba észlelésekor nem áll meg, hanem tovább fut. Nézzünk erre egy példát a SSHAPE utasítással kapcsolatosan:

```
10 YE=20
20 GRAPHIC1
30 TRAP60
40 SSHAPE A$,0,0,200,YE
50 GRAPHIC0:PRINT YE;"A LEGNAGYOBB ERTEK":END
60 YE=YE-1:GOTO30
```

A TRAP60 utasítás hatására a program futása egy esetleges hiba fellépésekor a 60-as programsorban folytatódik. Ebben a sorban az YE értéke 1-gyel csökken, majd a végrehajtás visszakerül a 30-as sorra.

Ha ezt a programot nem akarjuk használni, de mégis szeretnénk megállapítani, hogy egy bizonyos alakzat belefér-e egy karaktérváltozóba, akkor ezt egész egyszerűen, közvetlen üzemmódban is kipróbálhatjuk.

Legyen például a tárolni kívánt alakzatunk hossza az X irányban 30 képpont. A 2000-et 30-cal elosztva 66-ot kapunk, tehát kb. ilyen hosszú lehet az Y oldal. Próbáljuk ezt ki. Írjuk be a gépbe közvetlen üzemmódban:

SSHAPE A$,0,0,30,66

Ha a számítógép a

?NO GRAPHICS AREA ERROR

hibaüzenetet küldi (nincs grafikus terület), akkor a grafikus üzemmódot egyszer be, majd utána ki kell kapcsolni. A számítógépnek tehát először területet kell lefoglalnia a tárban a grafika számára, mert egyébként nem tudná az alakzatot tárolni. Ha ezzel megvagyunk, akkor a számítógépnek a beírt parancsra a

?STRING TOO LONG ERROR

hibaüzenetet kell kiírnia. Az alakzatunk tehát túlságosan hosszú. Menjünk most a

kurzorbillentyűk használatával a parancssorba (SSHAPE), és írjuk át az ott levő 66-ot először 63-ra. Nyomjuk le a RETURN billentyűt. Most is hibaüzenetet kapunk. Ha viszont az utolsó számra 62-t adunk meg, akkor a RETURN lenyomása után a számítógép a READY üzenettel jelentkezik vissza. Az X = 30 mérőszámú alakzat Y irányban tehát csak 62 képpont hosszú lehet (összesen 1860 képpont). Véleményünk szerint ez a próbálgatás a legegyszerűbb módszer egy alakzat méretének a vizsgálatára.

A GSHAPE utasítás

A GSHAPE utasítással az alakzatokat ismét láthatóvá tehetjük.
Szintaxis:

GSHAPE karakterváltozó (,x,y,megjelenítés módja)

Az x és y koordináták annak a területnek a bal felső sarkát jelölik ki, ahol az alakzatnak meg kell jelennie. Az x és az y, valamint a megjelenítési mód paraméterek el is hagyhatók. Ebben az esetben a számítógép a kijelölt pontnak a grafikus kurzort tekinti, a megjelenítés módja pedig 0. Az alakzat összesen öt különböző módon jeleníthető meg:

Mód	Megjelenítés
0	Ahogyan tárolva lett
1	Inverz
2	Háttér OR alakzat
3	Háttér AND alakzat
4	Háttér XOR alakzat

Emlékezzünk vissza az ötödik fejezetben ismertetett logikai műveletekre. Az előbbi példaprogramunk a GSHAPE utasításnak mind az öt megjelenítési módját bemutatja. Ezek között olyanok is vannak, amelyek bitenként logikai műveleteket végeznek a háttér tartalma és az alakzat tartalma között. Sajnos az a sebesség, amellyel az alakzatokat meg lehet rajzolni, nagyon kicsi. Ezért az alakzatokat a C64-esen vagy C128-ason használható, jólismert sprite-okkal nem lehet összehasonlítani. Az alakzatokból felépített, mozgatható grafikák egész egyszerűen túlságosan lassúak. Ezért ezek aligha használhatók játékokhoz úgy, mint amilyen kiválóan a sprite-ok.

15.10. A GRAFIKA TÁROLÁSA

Már volt arról szó, hogy lehetőség szerint tároljuk a programokat kazettán vagy hajlékony lemezen. Ekkor ugyanis lehetőség van arra, hogy a programokat bármikor visszatöltsük a gépbe. Ezáltal egyrészt megkíméljük magunkat a programok fáradságos újragépelésétől, másrészt a gépeléskor mindig becsúsznak gépelési hibák. Jogosan kérdezheti valaki azonban, hogy vajon miért kell egy grafikát tárolni, ha magát a programot, amely a grafikát előállítja, tárolhatjuk.

A kérdés jogos, és a legtöbb esetben valóban egyszerűbb is lenne a grafika újbóli előállítása, mintsem hogy ugyanezt a grafikát egy külső tárolóegységből betöltsük.

Vannak azonban olyan szituációk, amikor mégiscsak célszerűbb a kész grafika betöltése. Gondoljunk csak arra, hogy esetenként igen sokáig is tarthat, míg egy program egy grafikát elkészít. Ha egy programban bonyolult és terjedelmes számításokat kell végezni, akkor néha perceket is igénybe vehet a grafika megrajzolása. Ilyen esetekben gyorsabb lehet a kész grafika betöltése.

Az is előfordulhat, hogy egy meglevő grafikát egy új programnak meg kell változtatnia. Ilyenkor ugyancsak szükséges lehet, hogy a grafikát változatlan formában tároljuk.

Másik példa: olyan programunk van, amelyben egymás után több, különböző grafikát kell megjeleníteni. Lehetséges, hogy ehhez nem kell minden esetben a teljes képernyőtartalmat kicserélni, hanem elegendő, ha csak egyes részeit cseréljük. Ilyenkor is jól használható a tárolt grafika.

Úgy gondoljuk, hogy a programok tárolását és betöltését nem kell külön ismertetnünk. Erre vannak a SAVE és a LOAD, ill. a DSAVE és a DLOAD parancsok. Egy grafika tárolása vagy betöltése azonban nem ennyire egyszerű. Ezt a műveletet a BASIC sajnos nem támogatja. Ahhoz, hogy ezt a problémát megoldjuk, vizsgáljuk először meg, hogyan tárolja a számítógép a tárában a grafikát. Továbbá azt is tudnunk kell, hogy a tárban mely címeken helyezkedik el a grafika.

Vágjunk a dolgok közepébe: egy grafika például: úgy tárolható, hogy azt a tárból byte-ról byte-ra kiolvassuk, és ezeket a byte-okat kazettára vagy mágneslemezre írjuk. A grafika tárolásához használhatjuk a beépített monitorprogramot. Ezt a programot BASIC-ből a

MONITOR

paranccsal indíthatjuk.

A monitorprogramban lehetőségünk van arra, hogy akár egy, vagy tetszőlegesen sok byte-ot tároljunk. Adjuk ki az erre szolgáló parancsot:

S"név",d,kkkk,vvvv

ahol a

név	a tárolandó állomány (itt a grafika) neve
d	tárolóegység azonosító száma (lemezmeghajtó esetén 8, kazettás magnó esetén 1)
kkkk	a tárolandó tartomány kezdőcíme hexadecimális alakban
vvvv	a tárolandó tartomány végcíme + 1 (hexadecimális).

Ahhoz, hogy a teljes grafikatárat tároljuk (pl. mágneslemezen), adjuk ki a következő parancsot:

S"GRAFIKA",8,1800,4000

Nézzük meg ezt egy példán is. Előbb azonban elő kell állítanunk egy grafikát. A program:

```
10 COLOR0,1
20 COLOR1,2
30 GRAPHIC1,1
40 FORX=0TO300STEP10
50 DRAW,X,0TOX+10,199TOX+10,0
60 NEXT
70 GRAPHIC0
```

RUN

Amikor a programfutás befejeződött, adjuk ki a

MONITOR

parancsot, és az előbbi paranccsal tároljuk a grafikát. Ha ehhez kazettás egységet használunk, akkor a 8-as helyett természetesen egy 1-est kell írni. Kazetta esetén a tárolás közel 4 percig tart, mágneslemez esetén csak kb. 30 másodpercig. A grafikát a színinformációk nélkül is tárolhatjuk. Ebben az esetben a színeket utólag, egyszerű COLOR utasításokkal megváltoztathatjuk. Ekkor a monitorprogramból ezt kell beírni:

S"GRAFIKA",8,2000,4000

15.11. A GRAFIKA BETÖLTÉSE

Egy grafika betöltéséhez a grafikus tárat elő kell készíteni. Ez a GRAPHIC utasítással érhető el, pl.:

GRAPHIC1,1

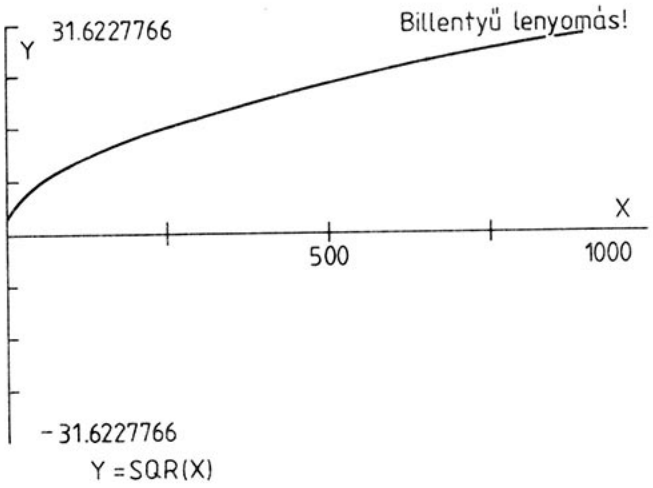
Ezután töltsük be a grafikát, mint egy gépi kódú programot:

LOAD"név",d,1

A szintaxisban szereplő 1-est, ami az ún. másodlagos cím, feltétlenül meg kell adni ahhoz, hogy az állomány a megfelelő címtől kezdődően kerüljön betöltésre. A grafikát programból is be lehet tölteni. Itt is úgy kell eljárni, mint a gépi kódú programrészek esetén. Ehhez a grafikus képernyőt nem kell feltétlenül bekapcsolni. Csak azt kell biztosítani, hogy a grafika számára a tárterület le legyen foglalva. Ezzel kapcsolatban utalunk a 16. fejezet Gépi kódú programok betöltése című pontjára.

15.12. FÜGGVÉNYEK RAJZOLÁSA

Ismerkedjünk meg most egy program alapjai a Plus/4-es kiváló grafikai képességeivel. A szóban forgó programmal bármilyen függvényt ábrázolhatunk a képernyőn. Bár a program egy kicsit hosszú, mégis úgy gondoljuk, hogy megéri a beírást.



15.3. ábra Függvény rajzolása

A program kezeléséhez:

Az ábrázolandó függvényt az 540-es programsorba kell beírni.

540 DEFFNA(X) = függvény

A program beírásakor gondosan ügyeljünk a szintaxisra. A függvény által felveendő értékek miatt nem kell aggódnunk. A számítógép akkor sem szakítja meg a program futását, ha hibát talál (pl. osztás nullával), hanem folytatja a számolást. Ehhez ismét a TRAP utasítást használjuk.

Indítás után a program bekéri az X tengelyre vonatkozó maximális értéket. Erre nullánál nagyobb, pozitív számot adjunk meg, szögfüggvényeknél pl. 90-et, 180-at vagy 360-at. A számítógép most kiszámítja az Y maximális értékét, és ennek megfelelően osztja be a képernyőt úgy, hogy mindig optimálisan kihasználja a képernyő formátumát (Y irányban 200 pont).

```
10 REM FUGGVENYEK RAJZOLASA
20 REM A FUGGVENY AZ 540-ES SORBAN VAN
30 GOSUB540
40 REM HIBAKEZELES
50 TRAP520
60 PRINTCHR$(147):PRINT"            FUGGVENY RAJZOLASA"
70 PRINT"-----------------------------------------"
80 PRINT"FIGYELEM: A KEPERNYO NEHANY MASODPERCRE              ELSOTETUL"
90 PRINT:PRINT
100 INPUT"AZ X MAXIMALIS ERTEKE";XM
110 REM A KEPERNYO KIKAPCSOLASA
120 POKE65286,PEEK(65286)AND239
130 REM A KOORDINATATENGELYEK LATHATATLAN MEGRAJZOLASA
140 GRAPHIC1,1
150 DRAW1,0,100TO319,100
160 DRAW1,0,0TO0,199
170 FORZ=0TO319STEP80
180 DRAW1,Z,95TOZ,105
```

```
190 NEXTZ
200 FORZ=0TO199STEP25
210 DRAW1,0,ZTO5,Z
220 NEXTZ
230 REM AZ Y MAXIMALIS ERTEKENEK SZAMITASA
240 FORF=0TOXMSTEPXM/320
250 X=F/180*3.14
260 Y=FNA(X)
270 IFY>YMTHENYM=Y
280 NEXT
290 REM A KEPERNYO BEKAPCSOLASA
300 POKE65286,PEEK(65286)OR16
310 REM A GRAFIKUS KURZOR AZ X=0, Y=100 PONTRA ALL
320 LOCATE0,100
330 REM A FUGGVENY SZAMITASA
340 FORF=0TOXMSTEPXM/320
350 REM ATSZAMITAS I\MERTEKRE
360 X=F/180*3.14
370 Y=FNA(X):Y=100-Y*(100/YM)
380 DRAW TOF*(320/XM),Y
390 NEXT
400 REM ABRAFELIRATOK
410 CHAR1,38,11,"X"
420 CHAR1,18,13,STR$(INT(XM/2))
430 CHAR1,35,13,STR$(INT(XM))
440 CHAR1,1,1,"Y"
450 CHAR1,2,0,STR$(YM)
460 CHAR1,2,24,STR$(-(YM))
470 CHAR1,19,0,"BILLENTYUT LENYOMNI!",1
480 GETKEYA$
490 GRAPHIC0,1
500 LIST540
510 END
520 RESUME NEXT
530 REM ITT ALL A FUGGVENY
540 DEFFNA(X)=SIN(X)/SIN(2*X)
550 RETURN
```

A program ismertetése:

A program a 30-as sorról az 540-es sorra ugrik. Ebben a sorban van a függvényünk, amelyet a DEFFN utasítással az A névhez rendeltünk.

Az 50-es sorban levő TRAP utasítás hatására – számítási hiba fellépésekor – a program futása az 520-as sorban folytatódik. Az itt levő RESUME arra utasítja a számítógépet, hogy a hibát hagyja figyelmen kívül, és a feldolgozást azon a helyen folytassa, ahol a hiba bekövetkezett.

A 100-as sorban várja a program az X maximális értékét.

A 120-as programsorban egy különleges programozási fogást találunk. Az itt levő utasítás kikapcsolja a képernyőt. Erre a következő okok miatt van szükség: ahhoz, hogy megkapjuk az Y maximális értékét, előzőleg a programnak az összes számítást el kell végeznie. Ez természetesen meglehetősen sok időt vesz igénybe. Azért, hogy a számításokat némileg meggyorsítsuk, kikapcsoljuk a képernyőre írást. Ekkor a számítógép gyorsabban tud dolgozni, mert a megjelenítéssel összefüggő, időigényes szervezési munkák elmaradnak.

A 140–220 programsorok a koordináta-rendszer tengelyeit rajzolják meg. Mivel a képernyő ki van kapcsolva, ez természetesen láthatatlanul történik.

A 240–280-as sorokban a program az Y maximális értékét számítja ki.

A 300-as sorban levő utasítás ismét bekapcsolja a képernyőt. Ez úgy történik, hogy a TED egyik regiszterében, a 65286-os tárcímen levő rekeszben levő 4-es bit értékét 1-re állítjuk (ez a mondat most még valószínűleg érthetetlen; a magyarázatot a könyv további részén adjuk.)

A 410–470-es sorok végzik az ábra feliratozását, és írják ki az X és az Y maximális értékét.

A 480-as sorban a program egy billentyű lenyomására vár.

A 490-es sor ismét kikapcsolja a grafikát.

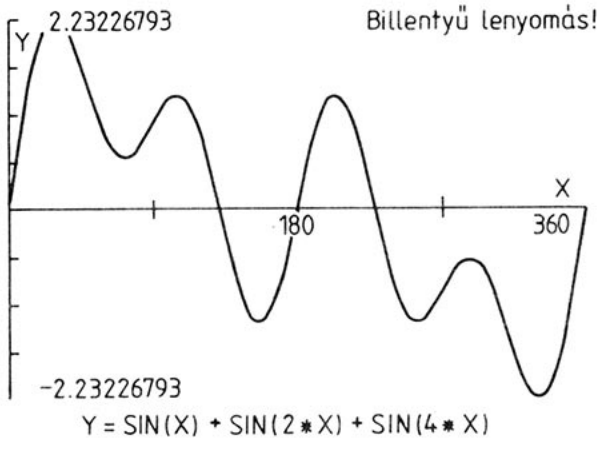
Az 500-as sorban levő utasítás hatására a program kilistázza a függvényt tartalmazó programsort. Ez nagyon hasznos dolog, hiszen a függvényt most azonnal meg is változtathatjuk. A LIST utasítás tehát programon belül is használható. A LIST utasítás hatására azonban a program futása nem folytatható. Ezért ez a sor a programból ki is hagyható.

Az 510-es sorban befejeződik a program futása (ez a sor is elhagyható).

Az 520-as sorban levő RESUME NEXT utasítás hatására a program hiba esetén is tovább fut.

Ha a programmal nem szögfüggvényeket akarunk ábrázolni, akkor a 250-es és a 360-as sorokban levő utasításokat ki kell hagyni. Ezek az utasítások az ívmértékre való átszámítást végzik, mivel a számítógép a szögek adatait ívmértékben várja. (Fordító megjegyzése: a két, átszámítást végző utasítás helyére az X = F értékadó utasítást kell mindkét programsorba beírni!).

Egy másik, szintén a programmal készített grafika:



15.4. ábra Függvénygörbe

A program természetesen továbbfejleszthető. Elképzelhető, hogy az X maximális értéke helyett a minimális értékének a megadása a célszerű. A Z-nek (ami a koordinátarendszer kezdőpontja) sem kell okvetlenül 0-nál kezdődnie. Így lehetővé válik, hogy egy függvénygörbének csak egy kisebb részletét ábrázoljuk.

15.13. A HÁROMDIMENZIÓS FÜGGVÉNY PÉLDAPROGRAMJA

Nézzünk egy másik lehetőséget a függvények grafikus ábrázolására. A függvényt három dimenzióban ábrázoljuk. Az X és Y tengelyhez hozzájön még tehát a Z tengely. A Z tengely adja a függvényábrázolás „mélységét".

A következő program alapját a 64'er nevű folyóirat 4/85-ös számának egyik cikke képezi. A programot átírtuk a Plus/4-es gépre. Ez a változat ugyan nem annyira kényelmes mint az eredeti program, viszont rövid, és így gyorsan beírható.

```
10 REM 3D-FUGGVENY
20 XA=50:YA=20:ZA=20
30 COLOR0,1
40 COLOR1,2
50 GRAPHIC1,1
60 FORXH=0TO10:FORYH=0TO10STEP.05
70 X=XH:Y=YH:GOSUB90:X=YH:Y=XH:GOSUB90:NEXTYH,XH
80 GETKEYA$:GRAPHIC0:END
90 Z=5+5*(SIN(X/10)*SIN(X/5)-SIN(Y/5))
100 D=XA-X:IFD=0THEN140
110 YS=15.7*(Y-X*(YA-Y)/D)+120:ZS=(-20)*(Z-X*(ZA-Z)/D)+150
120 IFYS<00RYS>319ORZS<00RZS>199THEN140
130 DRAW,YS,ZS
140 RETURN
```

Itt a program 90-es sorában van a függvény. Arra azért ügyelni kell, hogy a Z értékei nagyjából −5 és 5 között legyenek. Mivel a programban a számítások meglehetősen hosszú időt vesznek igénybe, egy függvény vizsgálatához a 60-as sorban a STEP után nagyobb érték is írható.

```
60 FORXH=0TO10:FORYH=0TO10STEP.5
```

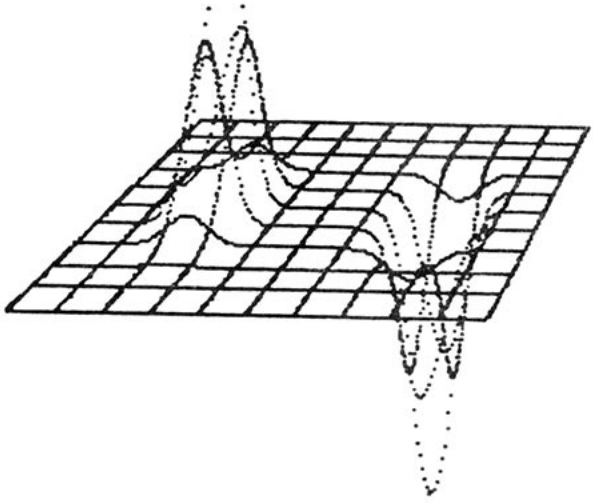
A 20-as sorban levő értékek a kívánt perspektívát adják meg. Ezek például ilyen értékek is lehetnek:

```
20 XA=50:YA=30:ZA=40
```

Három további változat a 90-es sorra:

```
90 Z=5+2*(SIN(X/5)-SIN(Y/2))
90 Z=5+5*(SIN(3.14/10*X)*SIN(6.28/10*Y))^11
90 Z=5-2*(SIN(3.14/10*X)*SIN(3.14/10*Y))^8
```

Bemutatunk két, a programmal készített háromdimenziós függvényt.



15.5. ábra Példa

15.6. ábra Példa

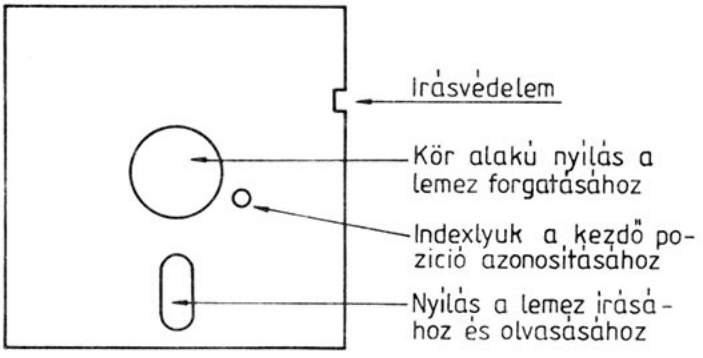
16. A lemezmeghajtó egység programozása

Szerencsés az az ember, aki a Plus/4-es számítógép mellett még egy lemezmeghajtó egység tulajdonosának is mondhatja magát. Mennyi tárolási és betöltési időt lehet így a kazettás egységhez képest megtakarítani! Akárhogy is legyen, mindenkinek javasoljuk, olvassa el ezt a fejezetet. Reméljük, hogy azok, akiknek már van ilyen lemezes készülékük, további hasznos ismereteket kapnak, a jövendőbeli tulajdonosok pedig kedvet ahhoz, hogy erre (is) takarékoskodjanak.

A lemezmeghajtó egység kezeléséről és programozásáról akár külön könyvet is lehetne írni. Ezért ez a fejezet egy kicsit hosszabb is a többinél. Sajnos elkerülhetetlen, hogy ennek során ne kelljen néhány szakkifejezéssel megismerkedni. A következőkben erről az érdekes és hasznos készülékről mondjuk el a legfontosabb tudnivalókat.

16.1. A HAJLÉKONY LEMEZEK

A hajlékony lemezekre széles körben elterjedt floppy disc elnevezés amerikai eredetű, és szó szerint „laza", a számítástechnikai gyakorlatban pedig „hajlékony" lemezt jelent. Ránézésre és az anyagára vonatkozóan is nagyon hasonlít egy hanglemezre: kör alakú, és műanyagból készült. Ami a hanglemezektől megkülönbözteti, az elsősorban az a mágnesréteg, amivel a lemez mindkét oldala be van vonva – és amire felvehető, illetve amiről lejátszható az információ –. A lemez impregnált műanyag tasakban van, amely a szennyeződésekkel és a külső mechanikai károsodásokkal szemben védi. A tasak egyúttal némi stabilitást is ad a lemeznek.



16.1. ábra Egy hajlékony lemez felépítése

A lemezborító egyik oldalán négyszög alakú bevágás van. Ha ez a bevágás le van ragasztva, akkor a lemezről csak olvasni lehet, ráírni viszont nem. Ez az úgynevezett írásvédelem. A lemez közepén található a forgatásához szükséges, kör alakú nyílás. Ahhoz, hogy a lemezen levő adatokat ill. programokat el lehessen olvasni, a mágneslemezt a műanyag borítón belül meg kell forgatni (akárcsak a hanglemezt a hangszedő tű alatt). A meghajtó egységben ezért egy motor is van. A középső, kör alakú nyílás gondoskodik arról, hogy a lemez mindig azonos középpont körül forogjon. A lemez ezen a nyíláson keresztül illeszkedik a meghajtó egység motorjához. Maga a lemez ezen a helyen van a legnagyobb mechanikai igénybevételnek kitéve, ezért lehetőség szerint olyat vásároljunk, amelynek ezen a belső nyílásán erősítő körgyűrű is van. Az „indexlyuk" csak bizonyos lemezmeghajtóknál jut szerephez. Az általánosan elterjedt Commodore 1541 vagy 1551 típusú meghajtó ezt nem használja. Ez a nyílás egyébként arra szolgál, hogy a számítógép megállapíthassa, hol „kezdődik" a lemez. A lemezen látható az úgynevezett író/olvasó fej nyílása. Ezen a nyíláson keresztül tud a lemezmeghajtó egység író/olvasó feje a lemezre írni, ill. onnan olvasni.

Megjegyzés: ilyen nyílás mindkét oldalon van, hogy az egy fejjel rendelkező gép (ilyen az 1541/1551-es) a lemez megfordítása után a másik oldalon is tudjon írni (ha engedélyezett).

Nagyon fontos: Soha ne érintsük kézzel a lemez mágneses bevonatát! Ezáltal ugyanis bepiszkolódhat a lemezmeghajtó író/olvasó feje, és a készülék máris hibázhat. A lemezeket mindig az eredeti tasakjukban, lehetőség szerint pormentes és száraz helyen tároljuk. Hallottuk azt is, hogy ezek a Commodore lemezmeghajtók nem megbízhatók. Ezzel szemben határozottan állítjuk, hogy aki csak egy kicsit is vigyáz a lemezeire, és megfelelően gondoskodik a meghajtó tisztaságáról, az éveken keresztül hibátlanul dolgozhat velük. Nekünk legalábbis ilyen problémáink nem voltak.

A meghajtó egység karbantartására, így pl. a készülék és az író/olvasó fej tisztítására, vagy a megvezető sínek kenésére vonatkozóan javasoljuk az idevonatkozó irodalom tanulmányozását.

A hajlékony lemezes meghajtó egység

Ahhoz, hogy a hajlékony lemezekre írni, ill. azokról olvasni tudjunk, természetesen egy lemezmeghajtó egységre is szükség van. Ebben az egységben van a lemezek forgatását végző meghajtó motor, valamint a síneken mozgathatóan felszerelt író/olvasó fej, amelyet egy léptető motor mozgat. A 1541-es és a 1551-es lemezmeghajtó elektronikája tulajdonképpen önálló számítógépnek is tekinthető, amelyik a motorokat és az adatok feldolgozását vezérli.

Csakúgy mint a Plus/4-es, a meghajtó egységben levő számítógép is tartalmaz ROM (csak olvasható) tárat. Ebben a tárban van az úgynevezett DOS (Disk Operating System), vagy magyar nevén a lemezoperációs rendszer. Ez az a program, amely a meghajtó egységet működteti.

1551-es lemezmeghajtó egység

Külön a Plus/4, a C16 és a C116 számítógépekhez fejlesztették ki a 1551-es lemezmeghajtó egységet. Ennek a típusnak a 1541-essel szemben több előnye is van. Egyrészt olcsóbb, másrészt pedig – és ez a legfőbb előnye – mintegy négyszer gyorsabb, mint a 1541-es. A 1551-es egységet a számítógép bővítő csatlakozójához kell kapcsolni. Az adatátvitel párhuzamos üzemmódban történik.

A 1551-esnek és a 1541-esnek majdnem azonos az utasításkészlete. A 1551-es két új, érdekes és hasznos utasítást ismer, ami azonban nem jelenti azt, hogy a különböző meghajtókkal felírt lemezek a másik meghajtóval nem lennének minden további nélkül olvashatók.

16.2. LEMEZES MEGHAJTÓ VAGY KAZETTÁS EGYSÉG?

A kazettás egységeknek a lemezes meghajtókkal szemben az a nagy előnyük, hogy lényegesen olcsóbbak és nagyobb mennyiségű adat tárolható rajta. Hátrányuk viszont több van.

A legnagyobb hátrány az adattárolás elvéből következik. A kazettán az adatokat csak sorban, egymás után (ún. szekvenciális módon) lehet tárolni. Könnyen belátható, hogy ilyenkor milyen körülményes lehet, ha az adatok egy bizonyos részéhez akarunk hozzájutni. Ehhez a szalagnak véletlenül éppen a kívánt helyen kell állnia, vagy a kívánt helyig előre kell tekerni, ill. a közbenső adatokat is át kell olvasni. Ez természetesen elég sok időt vehet igénybe. A lemezeknél egészen más a helyzet: bár az adatok tárolása bizonyos értelemben itt is sorban, egymás után történik, (szekvenciális file), az adatok bizonyos részéhez való hozzáférés azonban lényegesen egyszerűbb. Az író/olvasó fej egész egyszerűen a kívánt helyre áll, és elolvassa a keresett adatot.

Az ilyen fajta elérésre csak a lemezeknél van lehetőség, ahol a rendelkezésre álló tárterület teljes egészében „nyitva" van. Kazettáknál az adatok egy hosszú szalagon vannak, és ahhoz, hogy a kívánt adathoz hozzáférjünk, a szalagot előzőleg előre vagy vissza kell tekerni. Másik dolog, ami lehetővé teszi a lemezeken az adatok gyors elérését: mindegyik lemezen van egy tartalomjegyzék. Ennek köszönhetően a meghajtó egység mindig „tudja", hogy a kívánt adatot a lemez melyik részén keresse.

A lemezeknek a kazettákkal szembeni másik előnye a nagy adatátviteli sebesség. Ahhoz, hogy egy 10 kbyte nagyságú programot kazettán tároljunk, kb. 3 perc szükséges. Ugyanilyen hosszúságú programnak lemezre való írásához viszont csak 10–30 másodperc kell. A lemezeken levő adatok feldolgozási sebessége a kazettákéhoz képest kb. 10–40-szeres. (Ezért készítettek turbo programot a kazettás tárolás gyarapítására.)

Még egy előny: A lemezmeghajtókkal lehetőségünk van arra, hogy úgynevezett relatív állományokat hozzunk létre. Ezeknél az állományoknál az egyes összefüggő adatok (az úgynevezett rekordok) folyamatosan vannak sorszámozva. Összehasonlításként gondoljunk az indexes változókra. Bármikor elolvasható pl. az A(55) változó tartalma. Ez a helyzet a relatív állományoknál is. Ha tehát egy meghatározott

adatcsoportot (rekordot) akarunk olvasni, akkor elegendő, ha ennek csak a sorszámát közöljük a meghajtó egységgel. A meghajtó egység ekkor az író/olvasó fejet pontosan arra a helyre állítja, ahol a keresett rekordot a lemez tárolja, és máris elolvashatja. Ez a fajta adattárolás a kazettás egységen nem lehetséges.

Abban biztosak vagyunk, hogy aki a számítógépével komolyan akar foglalkozni, az előbb-utóbb mindenképpen vesz egy lemezes meghajtó egységet. Kezdetben azonban megteszi egy kazettás egység is.

16.3. AZ ÁLLOMÁNYOK TÍPUSAI

Ebben a fejezetben gyakran találkozunk az „állomány" fogalommal (eredeti, és széles körben elterjedt angolul a neve: file). Azért, hogy pontosan tudjuk, mit értünk alatta, álljon itt egy rövid magyarázat: állománynak nevezzük a tetszőleges számú, de egymással összefüggő adatok halmazát. Ha tehát írtunk egy programot a Plus/4-es számítógépbe, és ezt valamilyen adathordozón – kazettán vagy lemezen – tároljuk, akkor ez már egy állomány, mégpedig ebben az esetben programállomány. A lemez tartalomjegyzékében ezt az állományt a PRG betűk jelölik. A gépi kódú programok szintén programállományok. Vannak azonban más típusú állományok is. Ha egy programmal adatokat tárolunk vagy olvasunk, akkor ezek az adatok egy úgynevezett soros, vagy más néven szekvenciális állományba kerülnek. Az adatok a hajlékony lemezen úgynevezett „relatív állomány"-ba is szervezhetők.

A soros (szekvenciális) állomány csak egybefüggően, azaz az elejétől a végéig sorba tölthető a gépbe vagy írható ki valamilyen adathordozóra. Arra azért van lehetőség, hogy egy meglevő, soros állomány végére újabb adatokat írjunk. Ezeket az állományokat a lemez tartalomjegyzékében a SEQ rövidítés jelöli. A soros állományok pl. címjegyzékek készítésére használhatók.

A relatív állományok lehetővé teszik, hogy a lemezen levő állomány bármelyik adatát beolvassuk a gépbe, és hogy a lemezen levő állomány bármelyik adatát megváltoztassuk. A lemez tartalomjegyzékében ezeket az állományokat a REL rövidítés jelöli. Az így szervezett állományokban lényegesen rövidebb idő alatt lehet bizonyos adathoz hozzáférni, hiszen itt – ellentétben a soros állományokkal – nem kell a keresett adat előtt álló összes adatot végigolvasni. A relatív állományokkal kiválóan dolgozhatók fel pl. táblázatok. A relatív állomány tetszés szerint bővíthető, határt csak a lemez kapacitása szab. Így egy lemez egyik oldalán kb. 160 kbyte nagyságú állomány tárolható, és az állomány bármelyik pontja igen rövid idő alatt elérhető.

Egy állományt a lemezre íráskor vagy az onnan való olvasás előtt „meg kell nyitni". Ez azt jelenti, hogy a meghajtó egységet az adatátvitel előtt bizonyos utasítással elő kell készíteni. Ennek az utasításnak a neve az OPEN.

16.4. AZ OPEN UTASÍTÁS

Ez az utasítás a lemezes meghajtó vagy más adatbeviteli/kiviteli egység és a számítógép közötti átviteli csatornákat nyitja meg az adatforgalom számára.

Szintaxis:

OPEN fn,dn,sa

Az egyes paraméterek jelentése:

fn Az állomány száma. Értéke 1 és 127 közötti tetszőleges szám lehet. Az OPEN utasítást követően mindazokat az utasításokat, amelyek erre az állományra vonatkoznak, az itt megadott számmal kell jelölni.

dn A fizikai egység száma (device number). Egy állományban levő adatok nem csak kazettára vagy lemezre írhatók, hanem pl. a képernyőre is. Ez a paraméter a különböző fizikai egységek azonosítására szolgál.

sa Másodlagos cím (secondary address). (A lemezmeghajtó egységgel kapcsolatban e paraméter az adatforgalmat lebonyolító csatorna számát jelöli, ill. a nyomtató is használja.)

16.4.1. A fizikai egység száma

0 = billentyűzet
1 = kazettás egység
2 = RS-232 interface
3 = képernyő
4 = nyomtató
8 = lemezes meghajtóegység
9 = lemezes meghajtóegység

Amint látjuk, arra is van lehetőség, hogy az adatokat mágneslemez helyett pl. nyomtatóra írjuk ki. A billentyűzetet és a képernyőt az operációs rendszer automatikusan beviteli, ill. kiviteli egységnek tételezi fel.

Ettől függetlenül, programon belüli OPEN utasítással is kiírhatók a képernyőre az adatok. Lehet, hogy ez első olvasásra értelmetlennek tűnik, hiszen kézenfekvő, hogy a képernyőre a PRINT utasítással írjunk. Ha viszont egy programnak az adatokat felváltva, nyomtatóra vagy képernyőre kell küldenie, akkor ezt a kiírást egy és ugyanazon rutinnal lehet elvégezni, amennyiben az OPEN utasításban a fizikai egység azonosító számaként egyszer 3-at, egyszer pedig 4-et adunk meg.

16.4.2. A másodlagos cím (secondary address)

A másodlagos cím pl. a kazettás egységnél jut szerephez.

OPEN 1,1,0,"ÁLLOMÁNYNÉV"	Olvasásra nyit meg egy állományt.
OPEN 1,1,1,"ÁLLOMÁNYNÉV"	Írásra nyit meg egy állományt.
OPEN 1,1,2,"ÁLLOMÁNYNÉV"	Írásra nyit meg egy állományt, és az állomány végére elhelyezi a „szalag vége" (EOT = End of Tape) jelölést.

A „szalag vége" (EOT) jelölés azt jelenti, hogy a kazettán nincsenek további állományok. A számítógép tehát így „veszi észre", hogy a kazetta a végére ért. Lemezes meghajtó egység esetén ez a másodlagos cím (aminek az értéke 0 és 15 között lehet) az úgynevezett csatornaszám. A Plus/4-es számítógép operációs rendszere a 0 és az

l másodlagos címeket a LOAD és a SAVE utasításokhoz használja. A 15-ös másodlagos cím a lemezes meghajtó parancscsatornája számára van lefoglalva. Így a felhasználó a 2–14 közötti másodlagos címeket használhatja. A nyomtatónál a nyomtatás módja, szolgáltatásai állíthatók be.

Néhány példa:

OPEN 1,8,2,"TELEFON,S,W"	megnyit egy soros állományt írásra (S = szekvenciális, W = Write, írni)
OPEN 1,8,10,"LISTA,S,R"	megnyit egy soros állományt olvasásra (R = Read, olvasni)
OPEN 1,8,2,"TELEFON,S,A"	megnyitja a "TELEFON" nevű soros állományt az állomány bővítése céljából (A = Append, hozzáfűzni)

Most már tudjuk, hogy hogyan kell egy állományt megnyitni. A feldolgozás befejezésekor a megnyitott állományt ismét le kell zárni. Ezt a CLOSE utasítással végezzük.

16.5 A CLOSE UTASÍTÁS

A munka befejezésekor a megnyitott állományokat le kell zárni. Ez különösen akkor fontos, ha lemezmeghajtó egységgel dolgozunk. A nem megfelelően lezárt állományokat az operációs rendszer a lemez tartalomjegyzékében csillaggal jelöli meg. Mint minden, állományokkal dolgozó utasításnál, a CLOSE után is meg kell adni a lezárandó állomány számát, pl. CLOSE 5.

16.6. PRINT#, GET#, INPUT#, CMD

Legyen most egy megnyitott állományunk. Vajon hogyan lehet ebbe adatokat írni? Ugyanúgy, mint a képernyőre, tehát egy PRINT utasítással, és ezt követően az adatok megadásával. A PRINT utasításnak ehhez tartalmaznia kell az illető állomány számát is:

PRINT#f, adatok

Ez az utasítás nem azonos a normál PRINT utasítással. Ez például kérdőjellel (?) nem is rövidíthető. A # jel (létra, hashmark) az állomány jele. Ezért ezt az utasítást egyszerűen print-file-nak is nevezik.

```
10 OPEN1,8,2,"CIMEK,S,W"
20 INPUT"A CIM:";A$
30 PRINT#1,A$
40 INPUT"TOVABB <I/N>";IN$
50 IF IN$="I" THEN GOTO 20
60 CLOSE1
```

Vigyázat! a PRINT és a # jel közé nem tehető szóköz, mert nem érti meg.

Lássunk egy példát:

```
10 OPEN1,8,2,"CIMEK,S,R"
20 INPUT#1,A$
30 PRINTA$
40 IF ST=64 THEN CLOSE1:END
50 GOTO20
```

Ezzel a programmal címeket írhatunk egy soros állományba. Nézzük most azt a programot, amellyel ezeket az adatokat el tudjuk olvasni.

Az ST (status) rendszerváltozó értéke az összes adat elolvasása után 64 lesz. Ily módon egyszerűen megvizsgálható, hogy a programot befejezhetjük-e.

Az adatokat tehát az INPUT utasítással olvassuk a gépbe. A # jel itt sem hagyható el. A beolvasásra az INPUT# helyett a GET# utasítás is használható. Amint tudjuk, a GET utasítással csak egy karakter olvasható be. Ez vonatkozik a GET#-re is. A beolvasandó adatok természetesen több karakterből is állhatnak. A GET# használatakor tehát valamilyen módon fel kell ismernünk, hogy mikor érünk az adat utolsó karakteréhez. A megoldás nagyon egyszerű, mert az adatoknak a PRINT# utasítással való tárolásakor minden egyes adatbevitel végén egy úgynevezett kocsi vissza (CHR$(13)) jel is a lemezre kerül.

```
10 OPEN1,8,15,"CIMEK,S,R"
20 GET#1,X$
30 IF X$=CHR$(13)THENPRINTA$:A$="":GOTO20
40 A$=A$+X$
50 IF ST=64THENCLOSE1:END
60 GOTO 20
```

Itt minden egyes beolvasott karakter után megvizsgáljuk, hogy az a kocsi vissza karakter vagy nem. Ha az, akkor az előzőleg beolvasott jeleket kiíratjuk, a változó tartalmát töröljük, és újra kezdjük a beolvasást. Ha az ST rendszerváltozó értéke 64, akkor az utolsó adatot is beolvastuk. Az állományt lezárjuk, és a programot befejezzük.

A CMD utasítás

A CMD utasítással az adatok kiírását a képernyő helyett valamilyen más, az OPEN utasításban megadott fizikai egységre irányítjuk át.

Szintaxis:

CMD f

ahol az f annak az állománynak a száma, amelyet az OPEN és a CLOSE utasításban is használunk.

Példa:

```
OPEN1,4
CMD1
LIST
PRINT#1
CLOSE1
```

Ezzel az utasítássorozattal egy BASIC nyelvű program listáját a nyomtatóra írjuk. A CMD utasítást a PRINT# kapcsolja ki. A CMD utasítás megkönnyíti pl. szöveg nyomtatóra írását. Arra is van lehetőség, hogy az OPEN utasításban megadott egységszám megváltoztatásával ugyanazt az adatkiviteli rutint különböző egységeknél is felhasználjuk.

```
OPEN2,8,2,"PROGRAM,S,W"
CMD2
LIST
PRINT#2
CLOSE2
```

Ez az utasítássorozat egy program listáját soros állományként lemezre írja. Ezután ez az állomány pl. egy szövegfeldolgozó programmal visszaolvasható. (Ilyenkor nem kell a PRINT#, INPUT#.)

16.7. A JOKER KARAKTEREK

Két különleges karakter könnyíti a lemezműveleteket. Az egyik a kérdőjel, a másik pedig a csillag (*) karakter. Ezek az állománynevekben tetszőleges karaktert helyettesíthetnek. A kérdőjel egy, a csillag pedig tetszőlegesen sok karakter helyén állhat. Ezek használatával pl. könnyebben törölhető több, hasonló nevű állomány.

Nézzünk meg néhány példát:

A FILE? állománynév lehetne pl. FILE1 vagy FILE2 vagy FILEA stb. is.

??????JATEK lehetne pl. KARTYAJATEK vagy LABDA-JATEK.

Ha egy program betöltésekor a program nevében kérdőjelet használunk, akkor a gép az első olyan programot tölti be, amelynek a neve a megadott „mintával" megegyezik:

DLOAD"CIMEK?"

betölti az első olyan programot, amelynek a neve 6 karakterből áll, és az első öt betűje a CIMEK.

A JATEK* állhat pl. a JATEKPROGRAM vagy a JATEKOK vagy a JATEK-1 stb. helyett.

DLOAD"*"

betölti lemezről az első, tetszőleges nevű programot.

DLOAD"JATEK*"

betölti lemezről az első olyan programot, amelynek a neve "JATEK"-kal kezdődik.

A csillag és a kérdőjel egymással kombinálható is:

DLOAD"??????JATEK*"

betölti lemezről az első olyan programot, amelynek a neve hat tetszőleges betűvel kezdődik, ezután a "JATEK" betűk következnek, majd a megengedett maximális hosszig (16 karakter) ismét tetszőleges jelek. A joker karakterek különösen jól használhatók állományok törlésénél és a tartalomjegyzék listázásánál.

Az @ (at) karakter

E karakter segítségével lehet már meglevő állományokat felülírni. Tegyük fel, hogy a FUGGVENY nevű programunkat továbbfejlesztettük, és most ezt a javított programot szeretnénk változatlan néven ismét tárolni. Ehhez a következőt kell beírni:

```
DSAVE"@:FUGGVENY"
```

A név elé írt @ jel hatására az operációs rendszer a régi néven tárolja az új programot (a régi programot pedig törli). Soros állományok felülírásánál is használni kell ezt a karaktert.

Ajánlott irodalom: Említést tettünk az igen kényelmes adatkezelést lehetővé tevő relatív adatállományokról. Sajnos ezeknek a programozása nem túl egyszerű. Mindenesetre e könyv keretei közé már nem férne be. Hasonló a helyzet a lemez közvetlen elérését lehetővé tevő utasításokkal is. Ezekkel kapcsolatban utalunk a 1541-es lemezmeghajtó egység programozását ismertető szakkönyvekre.

16.8. A LEMEZUTASÍTÁSOK

A Plus/4-es számítógépen egy sereg BASIC utasítás könnyíti meg a meghajtó egységgel végzendő munkákat:

HEADER	DIRECTORY
DSAVE	DLOAD
BACKUP	COLLECT
COPY	SCRATCH
VERYFY	

A speciális lemezutasításokon túlmenően további utasítások is léteznek a lemezes állományok kezelésére:

OPEN	PRINT#
GET#	INPUT#
CMD	CLOSE

A lemezoperációs rendszernek (DOS) saját rendszerutasításai vannak:

NEW	SCRATCH
RENAME	COPY
INITIALIZE	VALIDATE

Ezeket az rendszerutasításokat nem szabad összetéveszteni a BASIC utasításokkal, még ha részben azonos is a hatásuk. Ezeket a rendszerutasításokat közvetlenül a lemezmeghajtó egységre kell küldeni. A rendszerutasítások kiadásához a BASIC OPEN utasítását kell használni.

Szintaxis:

OPEN fn,n,15,"utasítás:állomány neve...."

fn az állomány száma (1 és 127 közötti tetszőleges szám), amellyel az OPEN utasítás után mindig a 8-as egységszámú készülékre (lemezmeghajtó) hivatkozunk.

n a lemezmeghajtó egységszáma. Erre akkor lehet szükség, ha egyszerre két meghajtót csatlakoztatunk. A 1541-es és a 1551-es meghajtó egységszáma az alapértelmezés szerint 8.

15 A lemezmeghajtó egység úgynevezett parancscsatornája.

„utasítás: állomány neve...." A kívánt rendszerutasítás teljes egészében kiírva, vagy csak az első betűjével rövidítve. Az utasítás után egy kettőspontot kell írni, ha megadunk egy vagy több állománynevet.

Az egyes utasítások ismertetésénél minden esetben megadjuk az OPEN utasítás írásmódját is.

A HEADER utasítás

Amikor új lemezt vásárolunk, akkor ezt a felhasználást megelőzően elő kell készítenünk. A lemezen mindenekelőtt egy bizonyos alapmintát, nevet, azonosítót, a „formát" kell tárolni. Ezt a műveletet nevezik a lemez formázásának. Az így előkészített lemezek a formázott lemezek.

Az utasítás *szintaxisa:*

HEADER "lemez neve",Dx,Iy(,(ON)Un)

lemez neve Maximum 16 karakterből (betűkből és számokból) álló név.

x A lemezmeghajtó száma. Ha csak egy meghajtóval dolgozunk, akkor ez D0. Ha két meghajtó van csatlakoztatva, akkor az első száma a D0, a másodiké a D1.

y A lemezazonosító jele (IDentification). A lemezazonosító két, tetszőleges karakter lehet, ami a lemez formázásakor a név mellett szintén felíródik a lemezre.

n A lemezmeghajtó egységszáma általában a 8-as. Ha az egységszám ettől eltérő, akkor az itt adható meg. Ha nem adunk meg semmit, akkor a számítógép a 8-ast tételezi fel. Az utasításban az ON Un helyett csak Un is írható.

A HEADER utasítás tehát megformáz egy lemezt. Ez azt jelenti, hogy a lemezt sávokra, a sávokat pedig szektorokra osztja fel. A lemezen 35 sáv van, amelyek a lemez szélétől befelé haladva, 1-től 35-ig folyamatosan sorszámozva vannak. A sávok tehát az író/olvasó fej 35 lehetséges pozícióját jelölik. Mivel a lemez középpontja felé a körgyűrűk kisebbek, ezért a sávok kerülete is csökken. Ennek megfelelően a külső sávokon több szektor (a sáv egy meghatározott szakasza), míg befelé haladva egyre kevesebb szektor fér el. Az 1–17 sávokon sávonként 21 szektor, a 18–24 sávokon

19–19 szektor, a 25–30 sávokon 18–18 szektor, míg végül a 31–35 sávokon sávonként 17 szektor van.

A lemezünk összesen 683 szektorra van felosztva. A szektorokat más néven blokkoknak is nevezik. Az operációs rendszer a tartalomjegyzék számára lefoglal néhány blokkot, csakúgy, mint az úgynevezett BAM számára. Ez utóbbi a blokkfoglaltsági térkép, amely megmutatja a szabad, illetve a foglalt blokkokat. A programjaink számára a lemezen 664 blokk marad fenn. Egy-egy blokk 256 byte-nyi információt tartalmazhat. Ebből 2 byte-ot az az információ foglal el, amely megmutatja, hogy hol található a lemezen az illető blokkhoz tartozó következő, úgynevezett folytató blokk. Összesen tehát 664 blokk, blokkonként 254 byte, azaz 163000 byte áll a rendelkezésünkre.

A HEADER utasítás azonban nem csak a fenti szektorbeosztást készíti el, hanem létrehozza a lemez tartalomjegyzékét, az úgynevezett directory-t, vagy katalógust is. Ezenkívül a lemezen esetlegesen korábban tárolt információt teljes egészében letörli. Ez ennek az utasításnak nagyon fontos tulajdonsága. Az utasítás kiadása előtt tehát győződjünk meg arról, hogy tényleg a kívánt lemez van a meghajtóban. Ha egy már használt lemezt újra formázunk, akkor a rajta levő összes információ elveszik! Ennél az oknál fogva a gép a HEADER utasítást nem is hajtja azonnal végre, hanem előzőleg a képernyőn feltesz egy biztonsági kérdést:

ARE YOU SURE?

ami magyarul azt jelenti, hogy biztosak vagyunk a dolgunkban?

Ha a kérdésre igennel válaszolunk (Y, azaz YES), akkor a gép az utasítást végrehajtja.

Példák a HEADER utasításra:

HEADER"JATEKOK",I12,D0

Az utasítás megformáz egy lemezt, a lemez neve "JATEKOK", az ID-je pedig 12 lesz.

HEADER"PROGRAMOK",D0,U9

Megformáz egy lemezt a 0-ás számú, 9-es egységszámú lemezmeghajtóban. A lemez neve "PROGRAMOK" lesz. Mivel itt nem adtunk meg ID-t, marad a lemez korábbi ID-je. Tehát csak a lemez neve változik. Ez azonban csak már formázott lemezeknél használható. Új lemezek esetén mindig meg kell adni az ID-t is.

A HEADER egy BASIC utasítás. A formázáshoz azonban rendszerutasítás is használható. Ez így néz ki:

```
OPEN1,8,15,"N:JATEKOK,12"
CLOSE 1
```

Ennek ugyanaz a hatása, mint a HEADER-rel felírt első példánknak.

```
OPEN1,9,15,"N:PROGRAMOK"
CLOSE 1
```

Ez a második HEADER utasításnak felel meg. Vigyázzunk arra, hogy ne felejtsük el a CLOSE utasítást sem! Még néhány megjegyzés a lemezazonosítóval kapcsolatban:

Az ID lemezazonosító

Amint már mondtuk, az ID két, tetszőleges karakterből állhat. Lehetőség szerint azonban mindegyik lemeznek más-más ID-t célszerű adni. Ennek a következő az oka: a lemezmeghajtó egységben egy beépített RAM (írható és olvasható tár) is van. Az operációs rendszer ebbe a RAM-ba olvassa be az ún. BAM-ot, azaz a lemez blokkfoglaltsági térképét. Ebben a tárban módosítja is, illetve írja át a mindenkori helyzetnek megfelelően ezt a térképet, majd írja vissza a lemezre. Vannak azonban olyan lemezutasítások is, amelyek végrehajtása után az operációs rendszer nem tárolja azonnal vissza a lemezre a BAM-ot. Ha most a meghajtóban levő lemezt egy olyanra cseréljük ki, amelynek pontosan ugyanolyan az ID-je mint az előző lemezé, akkor a meghajtó az első lemez BAM-jával fog továbbra is dolgozni, ami nyilván hibás működést eredményez. Az operációs rendszer ugyanis csak a lemezek ID-jét figyeli, és ha ezek nem különböznek egymástól, akkor a cserét nem veszi észre.

A DIRECTORY utasítás

Ezzel az utasítással megnézhetjük a lemez tartalomjegyzékét. Ehhez elegendő, ha csak beírjuk magát a DIRECTORY utasításszót (vagy lenyomjuk az F3 billentyűt), anélkül, hogy egységszámot vagy bármilyen más paramétert meg kellene adnunk.

DIRECTORY (D0), (U8), (,"állomány neve")

Ha csak egy lemezmeghajtó egységgel dolgozunk, akkor a D0 és az U8 elhagyható. Ha a lemez teljes tartalomjegyzékére vagyunk kíváncsiak, akkor az állomány nevét sem kell megadni. A képernyőn ekkor megjelenik a lemez neve, az ID-je és az 1541-es, ill. 1551-es meghajtók azonosító jele, a 2A. Ezután íródik ki az egyes programok által elfoglalt blokkok (szektorok) száma, a programok (állományok) neve és típusa a képernyőre. Más lehetőség is van arra, hogy egy lemez tartalomjegyzékét beolvassuk. Ebben az esetben a DIRECTORY-tól eltérően a tartalomjegyzék a Plus/4-es tárába kerül ugyanúgy, mint egy program. Ekkor ez ki is listázható:

```
LOAD"$",8
LIST
```

(A 8-as a lemezmeghajtó egység azonosító száma). A DLOAD"$" utasítással a tartalomjegyzék nem tölthető be. Ne feledkezzünk meg arról, hogy a tartalomjegyzék betöltése következtében a tárban esetleg meglévő program felülíródik, és így törlődik is.

A lemez a tartalomjegyzékét tehát „$" néven tárolja. A lemezoperációs rendszer (DOS) a tartalomjegyzéket természetesen minden változáskor automatikusan helyesbíti a lemezen.

A DSAVE utasítás

A DSAVE utasítással programokat tárolhatunk lemezen.
Szintaxis:

DSAVE"állomány neve"(,D1)(,U9)

A D1 és az U9 itt is elhagyható, ha csak egy meghajtóval dolgozunk. Az állomány neve, azaz a program neve maximum 16 karakterből állhat. A DSAVE utasítással csak BASIC programok tárolhatók. Gépi kódú programok a gépi kódú monitorprogramból tárolhatók, vagy olyan, közvetett módon, hogy megnyitunk egy programállományt, és a program adatait byte-ról byte-ra átírjuk a lemezre.

A DLOAD utasítás

Szintaxis:

DLOAD"JATEKOK"(,D1)(,U11)

Ezzel az utasítással a "JATEKOK" nevű programot a 11-es egységszámú, 1-es meghajtóról a számítógépbe töltjük. Ha csak egy lemezmeghajtóval dolgozunk, akkor elég ennyit írni:

DLOAD"JATEKOK"

A DLOAD helyett használható ez az utasítás is:

LOAD"JATEKOK",8

Ha gépi kódú programot akarunk betölteni, akkor az utasítás:

LOAD"JATEKOK",8,1

A gépi kódú programok betöltése után ki kell adni a NEW utasítást. Ez az utasítás a nullás lapon levő egyes rekeszek (magyarázatot lásd a könyv további részein) tartalmát visszaállítja az eredeti értékükre. Ha viszont nem adjuk ki a NEW-t, akkor olyan hibaüzeneteket kaphatunk, amelyekkel nem tudunk mit kezdeni, és nem is javíthatók. Megjelenhet például az OUT OF MEMORY ERROR (=a tár végére értem) hibaüzenet. Ezt a következők okozhatják: programok betöltése után e programok végcíme egyidejűleg a BASIC-tár végcíme is lesz. Ha most pl. a lemezről beolvasunk egy grafikát, akkor a BASIC számára túl magas lesz a végcím (ha a grafikus képernyőt a GRAPHIC utasítással előzőleg lefoglaltuk). A LOAD és a DLOAD utasítás programból is használható. Ekkor a program betöltődik, és automatikusan elindul.

Ha a LOAD"programnév",8,1 utasítással gépi kódú programokat töltünk be, akkor ezek ugyan nem indulnak el automatikusan, viszont a BASIC program újra indul. Ezt az újbóli indulást nem lehet megakadályozni. A BASIC program tehát ismét a LOAD"programnév",8,1 utasításba ütközne, és a gépi kódú programot ismét betöltené. Ezt viszont már meg lehet akadályozni. A megoldás:

```
10 X=X+1
20 IFX=1 THEN LOAD"VALAMI",8,1
30 ....
```

Első pillantásra semmi különös sem látszik. A BASIC program indításakor ugyanis minden változó értéke nulla lesz. A mi példánkban az X kezdeti értékéhez hozzáadódik 1, és a 20-as sorban, amikor az X értéke 1, betöltődik a gépi kódú program. Azt is mondtuk, hogy a program betöltődése után újra indul a BASIC program. Ekkor tulajdonképpen az X változó értékének ismét zérusnak kellene lennie, amihez ismét hozzáadódna az 1, és kezdődne ismét a töltés.

Szerencsére ez azonban nem így van, mert ebben az esetben a változók tartalma nem törlődik. A gépi kódú program betöltése és a BASIC ismételt indulása után az X értéke továbbra is 1 marad. Most a 10-es sorban ehhez az értékhez hozzáadódik az 1, az X értéke tehát 2 lesz. Ekkor a 20-as sorban levő feltétel már nem teljesül, a program futása tehát a 30-as sorban folytatódhat.

A BACKUP utasítás

Ezzel az utasítással teljes lemezek másolhatók. Az is igaz viszont, hogy a másoláshoz két lemezmeghajtó egység szükséges. Ha valakinek csak egy készüléke van, akkor ezt az utasítást nem használhatja.

Szintaxis:

BACKUP Dx TO Dy(,(ON)Un)

x annak a meghajtónak a száma, amelyben az eredeti lemez van.
y a másik meghajtó száma.
n egységszám ha ez a 8-tól eltér. Beírható az ON Un is.

A másoláshoz egyébként az új lemeznek kell formázottnak lennie. Ha viszont már formázva volt, akkor a rajta lévő tartalomjegyzék, és minden, korábban esetleg felírt információ elvész! Legyünk ezért óvatosak ennek az utasításnak a kiadásakor. Győződjünk meg minden esetben arról, hogy a kívánt lemez a kívánt meghajtóban van!

A COLLECT utasítás

Ha egy lemezt gyakran használunk, akkor előfordulhat, hogy a tartalomjegyzékben nyilvántartott szabad és foglalt blokkok összege nem egyezik meg a maximálisan felhasználható blokkok számával, azaz a 664-gyel. Ezt nem megfelelően lezárt állományok is okozhatják. Ekkor a lemezt „rendbe kell rakni". Ennek az utasításnak a végrehajtása után a szabad és a foglalt blokkok számának összege ismét 664 lesz.

Szintaxis:

COLLECT (Dx) (,(ON)Un)

x A meghajtó száma, ami 0 vagy 1. Ha csak egy meghajtóval dolgozunk, akkor ezt nem kell megadni.
n Egységszám, alapesetben 8. Az Un vagy az ON Un paramétert csak akkor kell megadni, ha az egység száma nem 8.

A COPY utasítás

A COPY utasítással egyes állományok másolhatók. Ennek természetesen főleg akkor van értelme, ha egyik lemezről egy másik lemezre másoljuk az állományokat, amihez viszont ismét csak két meghajtó szükséges. Egyetlen meghajtó esetén az állományt ugyanazon a lemezen egyik helyről másik helyre csak megváltoztatott néven lehet átmásolni.

Szintaxis:

COPY(DX,)"forrás"TO(Dy,)"cél"(,(ON)Un)

x Annak a meghajtónak a száma, amelyben az állományt tartalmazó lemez van.
y A másik meghajtó száma.
n Egységszám, ha az nem 8.

Nézzünk erre néhány példát:

COPY D0,"NEVEK"TOD1,"NEVEK"

Az utasítás a "NEVEK" állományt a 0-ás meghajtóról az 1-es meghajtóra másolja át, az átmásolt állomány neve is "NEVEK" lesz, az egységszám 8.

COPY"GRAFIKA"TO"RAJZPROGRAM"

Az utasítás a „GRAFIKA" nevű állományról ugyanazon meghajtóban és lemezen másolatot készít "RAJZPROGRAM" néven.

COPY D0 TO D1

A 0-ás meghajtóról átmásolja az összes állományt az 1-es meghajtóra.

A SCRATCH utasítás

A SCRATCH utasítással állományok törölhetők a lemezről. A törölt állományok által elfoglalt blokkok ismét szabadok lesznek.

Szintaxis:

SCRATCH"név"(,Dx)(,(ON)Un)

„név" a törlendő állomány neve.
x A lemezmeghajtó száma. Ha csak egy 1541-es vagy 1551-es meghajtóval dolgozunk, akkor ez elhagyható.
n Egységszám, alapértelmezés szerint 8. Ha csak egy meghajtóval dolgozunk, akkor ennek az egységszáma általában 8. Ebben az esetben az Un vagy ON Un elhagyható.

A SCRATCH utasítás arra is alkalmas, hogy néhány másodperc alatt órák vagy napok munkáját is semmissé tegye. Ezért ezzel az utasítással nagyon óvatosan kell bánnunk. A számítógép az utasítás kiadása után – de még a végrehajtása előtt – ezért meg is kérdezi:

ARE YOU SURE?

azaz, hogy biztosan ezt akarjuk?

Ha a kérdésre az Y billentyűt (YES, tehát igen) nyomjuk le, akkor a gép az utasítást végrehajtja. Ajánlatos azonban, ha a válaszunkat kétszer is meggondoljuk.

Az igazsághoz tartozik azonban, hogy az ezzel az utasítással törölt állomány fizikailag nem törlődik a lemezről. Az utasítás csak a lemez tartalomjegyzékében változtat meg egy byte-ot, és a blokkfoglaltsági térképen a törölt állomány által elfoglalt blokkokat ismét szabadnak nyilvánítja. A tartalomjegyzék átírt byte következtében a tartalomjegyzék kilistázásakor ez az állomány már nem szerepel a jegyzékben. Ha azonban egy állományt a SCRATCH utasítással már töröltünk, akkor ezt csak egy speciális programmal (Pl. Disk Monitor) lehet újra "visszahozni".

A VERIFY utasítás

A verify angol szó magyarul nagyjából azt jelenti, hogy valaminek a valódiságát megvizsgálni. Ezzel az utasítással egy lemezen lévő program byte-ról byte-ra összehasonlítható a számítógép tárában lévő tartalommal. Így megállapítható, hogy egy programot valóban hibamentesen sikerült-e tárolnunk. Ha e vizsgálat során a számítógép különbséget észlel a tárában levő tartalom és a lemezen levő program között, akkor kiírja a VERIFYING ERROR hibaüzenetet.

Szintaxis:

VERIFY"név",n

n A lemezmeghajtó egységszáma, tehát általában 8.

Ha a vizsgálat nem talált eltérést, akkor a számítógép ezt írja ki:

VERIFYING OK

Javasoljuk ennek az utasításnak a használatát, még akkor is, ha ez sokáig is tart. A vizsgálat egyébként ugyanaddig tart, míg a program betöltése.

Ha egyszer mégis előfordulna ilyen hiba, akkor a következők szerint kell eljárni: tároljuk még egyszer a programot, de most más néven. Végezzük el a VERIFY utasítással az összehasonlítást. Ha ez a program hibátlan, akkor töröljük a lemezről az első változatot.

16.9. A LEMEZES MEGHAJTÓ EGYSÉG RENDSZERUTASÍTÁSAI

A NEW rendszerutasítás

Ennek az utasításnak ugyanaz a hatása, mint a BASIC HEADER utasításnak. A NEW tehát formázza a lemezeket.

Szintaxis:

OPEN fn,n,15,"NEW:nev,id"
CLOSE fn

vagy

```
OPEN fn,n,15
PRINT# fn,"NEW:nev,id"
CLOSE fn
```

fn az állomány logikai száma, amellyel az OPEN után a megfelelő utasításokkal (PRINT# fn, GET# fn stb.) mindig az n jelű egységre hivatkozunk. Az állomány logikai száma 1 és 127 közötti tetszőleges szám lehet.
n Egységszám, a 1541/1551-es meghajtó egységek esetén alapértelmezés szerint 8.
id A lemez azonosítója

Ha megadjuk az ID-t, akkor a meghajtó egység a lemezt teljesen újra formázza. Ha viszont elhagyjuk, akkor az utasítás csak a lemez tartalomjegyzékét törli (ez új, formázatlan lemezek esetén persze nem megy). A NEW utasítás az N betűvel rövidíthető.

A SCRATCH rendszerutasítás

Ennek az utasításnak ugyanaz a hatása, mint a BASIC SCRATCH utasításé. Az utasítással állományok törölhetők.
Szintaxis:

```
OPEN fn,n,15,"S:név1,név2..."
CLOSE fn
```

Az fn és n jelentése mint a NEW utasításnál.

S A SCRATCH rövidítése. Természetesen a teljes utasításszó is kiírható.
név1 A törlendő állomány neve.

Ha egyidejűleg több állományt akarunk törölni, akkor ezek neveit az utasításszó után, egymástól vesszővel elválasztva kell feltüntetni. Arra azonban ügyelni kell, hogy az idézőjelek közé 40-nél több karakter nem írható. Ha ez a hely nem lenne elegendő, akkor ki kell adni egy másik SCRATCH utasítást is. Ellenkező esetben maga az állomány nem törlődne, hanem csak a lemez tartalomjegyzéke és a blokkfoglaltsági térképe (BAM) változna meg.

A RENAME rendszerutasítás

A RENAME utasítás használatával a lemezen egy állomány neve megváltoztatható,
Szintaxis:

```
OPEN fn,n,15,"R:új név = régi név"
CLOSE fn
```

Az R betű helyett természetesen kiírható a teljes, RENAME utasításszó is. Először az állomány új nevét kell megadni, majd – egy egyenlőségjelet követően – az állomány régi nevét. A fenti utasítás így is írható:

```
OPEN fn,n,15
PRINT# fn,"R:új név = régi név"
CLOSE fn
```

A RENAME utasítást BASIC utasítások nem támogatják, tehát az állományok átnevezésére csak ez a módszer használható.

A COPY rendszerutasítás

Ezt az utasítást a BASIC lemezutasításainál már megismertük. A COPY utasítással állományok másolhatók.

Szintaxis:

```
OPEN fn,n,15,"C:új név = régi név"
CLOSE fn
```

Az utasítás hatására a lemezen az első (új) néven létrejön egy állomány, amely pontosan megegyezik a másodikként megadott állománnyal (régi).

Ennek az utasításnak a segítségével több állomány egy állománnyá is összefogható. Az utasítás lehetővé teszi például, hogy soros (szekvenciális) állományokat, tehát bizonyos módon egymással összefüggő adatokat egy új állományba összekapcsoljunk. Ehhez ezt kell beírni:

```
OPEN fn,n,15,"C:új név = régi név 1, régi név 2 ..."
CLOSE fn
```

vagy

```
OPEN fn,n,15
PRINT# fn,"C:új név = régi név"
CLOSE fn
```

Az INITIALIZE rendszerutasítás

Ezzel az utasítással arra utasítjuk a lemezmeghajtó egységet, hogy a lemez blokkfoglaltsági térképét – a BAM-ot – olvassa be az egység RAM tárába. Ezzel kapcsolatban utalunk a könyvnek a lemez ID-jére vonatkozó fejezetére (lemezazonosító).

Szintaxis:

```
OPEN fn,n,15, "INITIALIZE"
CLOSE fn
```

vagy

```
OPEN fn,n,15
PRINT# fn,"INITIALIZE" (vagy egyszerűen rövidítve, "I")
CLOSE fn
```

A VALIDATE rendszerutasítás

Ennek az utasításnak ugyanaz a hatása, mint a BASIC COLLECT utasításnak. Az utasítás a BAM-ban a foglaltként nyilvántartott, de nem rendesen lezárt állományokhoz, vagy egyetlen állományhoz sem tartozó blokkokat ismét szabaddá nyilvánítja. Ezt az utasítást csak abban az esetben kell használnunk, ha a tartalomjegyzékben megadott blokkok száma nem 664. Egyébként természetesen a BASIC COLLECT utasítását célszerű használni. A teljesség kedvéért azért ennek az utasításnak a szintaxisát is megadjuk:

Szintaxis:

```
OPEN fn,n,15,"V" (vagy kiírva:VALIDATE)
CLOSE fn
```

vagy:

```
OPEN fn,n,15
PRINT# fn,"V"
CLOSE fn
```

17. Közvetlen hozzáférés a tárhoz: a PEEK, POKE és a WAIT utasítás

Ezek az utasítások lehetőséget adnak arra, hogy a Plus/4-es számítógép tárának tartalmát megnézzük, illetve megváltoztassuk. A PEEK utasítás a tár tartalmának elolvasását végzi.

A PEEK utasítás

Szintaxis:

PEEK (tárcím)

tárcím az a tárcím, (rekesz), amelynek a tartalmát olvasni akarjuk. A tárcímet decimális számrendszerben kell megadni, értéke 0-65535 között lehet.

Példa:

PRINT PEEK (34)

eredménye 130 lesz. Az utasítással a 34-es tárcím tartalmát olvassuk, és írjuk a képernyőre. Ez a tárcím az úgynevezett nullás lapon (zero page, zp) van, amely a RAM-nak az a része, ahol a Plus/4-es a működéséhez fontos adatokat tárolja. Változtassuk meg most egy tárcím (rekesz) tartalmát. Ehhez a POKE utasítást kell használni.

A POKE utasítás

Szintaxis:

POKE tárcím, érték

tárcím az a tárcím decimális alakban, amelynek a tartalmát meg akarjuk változtatni.

érték az az érték – szintén decimális alakban – amelyet a fenti tárcímnek fel kell vennie.

Változtassuk meg most egy, a BASIC-RAM területen lévő tárcím tartalmát:

```
POKE 7000,215
```

Ha a PRINT PEEK(7000) utasítással meg akarjuk nézni a tárcím mostani (aktuális) tartalmát, akkor eredményül a 215-öt kapjuk. Ez tehát sikerült. Próbáljuk ki még a

következőt is (ajánljuk azonban, hogy ha a gép tárában valamilyen program van, akkor azt előzőleg tároljuk valahol):

```
POKE 48,0
DIM A(30)
```

Mi történt? Semmi más, mint hogy a számítógép „lemerevedett". Most sajnos a RESET gombbal sem sikerül működésre bírni. Nincs más hátra, a gépet ki kell kapcsolni, és újra kell indítani. Ezzel a példával csak azt akartuk bemutatni, hogy ez is „elérhető" egy POKE utasítással. Elegendő egy apróság, és a számítógép máris „fejreáll". Legyünk ezért nagyon óvatosak ezzel az utasítással!

Írjuk be az alábbi miniprogramot, és indítsuk el a RUN-nal (a példa csak akkor működik, ha előzőleg nem hívtunk meg grafikát, mert ellenkező esetben a BASIC-kezdet más címen van, és a POKE utasításnak nem lenne hatása a programra.)

```
10 PRINT 1000*5
```

(A programot pontosan úgy írjuk be, ahogyan itt látszik).

A programot futtatva természetesen az

```
5000
```

értéket kapjuk.

Írjuk be most a következő sort:

```
POKE 4106,173
```

Ha újra elindítjuk a programot, akkor meglepődve tapasztaljuk, hogy az eredmény egyszerre csak 200 lesz. A programot ki is listázhatjuk, és azt látjuk, hogy ez már nem 5000*5, hanem 5000/5 (azaz 5000 osztva 5-tel). A POKE utasítással tehát megváltoztattuk a programot. Persze ez meglehetősen szokatlan módja egy program megváltoztatásának, de gondoljunk csak arra, hogy egy program saját magát is meg tudja változtatni. Lehet, hogy ez is egy lépés a mesterséges intelligencia felé? A POKE utasítást nagyon jól lehet arra használni, hogy pl. a TED nevű integrált áramkör (chip) fontos regisztereinek tartalmát megváltoztassuk. Segítségével arra is van lehetőség, hogy az adatokat ne változókban tároljuk, hanem közvetlenül a számítógép tárába írjuk be. A változók tartalmát minden egyes RUN programindítás törli, ezzel szemben a tár tartalma érintetlen marad, feltéve, ha az adatok tárolására olyan tárcímeket használunk, amelyeket a számítógép operációs rendszere nem változtat meg. Ilyen adattárolási célra használhatók például az 1630–1771 közötti tárcímek, mivel ezt a tárterületet a számítógép nem veszi igénybe.

A PEEK és a POKE egymással kombinálható is. Erre akkor van szükség, ha egy tárcímen csak bizonyos, meghatározott biteket akarunk megváltoztatni. Ehhez először a tárcímen levő tartalmat ki kell olvasni, az értékét meg kell változtatni, majd a megváltoztatott tartalmat ugyanarra a címre vissza kell írni. Erre egy példa a képernyő kikapcsolása. Írjuk be:

```
POKE65286,PEEK(65286)AND239
```

A számítógép a látszat ellenére is működik. Az utasításkombinációval a 65286-os tárcím tartalma és a 239 között elvégeztük az AND (ÉS) logikai műveletet, amelynek eredményeként a 4-es bit értékét 0-ra állítottuk (az értéke 0 lett). Ez a bit azt jelzi, hogy a képernyő be van-e kapcsolva (1) vagy a képernyő ki van-e kapcsolva (0).

Írjuk be most „vakon":

```
POKE65286,PEEK(65286) OR 16
```

Ezzel a képernyőt ismét bekapcsoltuk, azaz a 4-es bit értékét ismét 1-re állítottuk. Egyébként a számítógép feldolgozási sebessége a képernyő kikapcsolása következtében mintegy 30%-kal felgyorsul. Ha olyan programmal lesz dolgunk, ahol lényeges a nagy sebesség, és a számítások közbenső eredményeit nem szükséges megjeleníteni a képernyőn, akkor ez az utasításkombináció jól használható.

Egy másik utasítás, amellyel a tárhoz közvetlenül hozzá lehet férni, a WAIT utasítás. Ezzel az utasítással a 6. fejezetben már foglalkoztunk, ezért itt csak röviden említjük:

A WAIT utasítás

Szintaxis:

WAIT tárcím,1.érték (,2 érték)

A WAIT utasítással lehetőségünk van arra, hogy egy program futását mindaddig felfüggesszük, amíg egy meghatározott tárcím egy meghatározott értéket fel nem vesz. Ezzel kapcsolatban javasoljuk a 6. fejezet ismételt átolvasását. Ezt az utasítást az Összehasonlító parancsok című fejezetben használtuk, mert ez is – akárcsak az IF ... THEN – összehasonlítást végez.

Figyelem! Mind a POKE, mind a WAIT utasítás használatánál fokozott óvatossággal kell eljárni! Helytelen használatukkal a programok nagyon könnyen „befagyhatnak". Ilyenkor a számítógépet csak a RESET gomb benyomásával, vagy csak a gép ki- és bekapcsolásával lehet ismét szólásra bírni. Írjuk be:

```
WAIT 134,0
```

Most csak a RESET gomb lenyomása segít!

18. READ, DATA, RESTORE

Ha egy programban sok változóhoz kell értékeket rendelni, akkor ehhez a címben szereplő három utasítás jelentős segítséget nyújt. Nézzünk mindjárt egy példát:

```
10 FOR I=1 TO 10
20 READ T(I)
30 NEXTI
40 DATA 1,2,3,4,5,6,7
60 FOR I=1 TO 10
70 PRINT T(I)
80 NEXT
110 DATA 8
120 DATA 9,10
```

Futtassuk a programot. A képernyőn megjelennek a számok 1-től 10-ig.

18.1. A READ UTASÍTÁS

Szintaxis:

READ változó név lista
pl. READ A
READ A$,B,C,D$...

Ha a számítógép a program végrehajtása során READ utasítást talál, akkor átnézi a programtárát felülről lefelé, és DATA sort keres. A mi példánkban elsőként a 100-as sorban fordul elő DATA. Most a gép elolvassa a DATA utasításszó után álló első elemet, és az értékét hozzárendeli a READ után álló változóhoz. A számítógép megjegyzi magának ezt a helyet, és a következő READ utasításkor már ettől a helytől kezdve keresi a további elemeket. Azt a tárrekeszt, amelyben egy tárcím található, mutatónak (pointer) is nevezzük. A READ utasítás tehát a DATA-mutatót mindig egy elemmel tovább állítja. Ha nem talál már újabb elemet, és DATA sor sincs több, akkor az OUT OF DATA (=elfogytak az adatok) hibaüzenetet kapjuk.

A READ utasítás után több változó is írható. A hozzárendelendő elemek számának és típusának meg kell felelni a változók számának és típusának. Ha a READ numerikus értéket vár, és a következő hozzárendelendő elem karakter típusú, akkor hibajelzést kapunk, (fordítva nem) és a program futása megszakad. Ügyeljünk arra, hogy a program mindig a megfelelő típusú elemet adja át.

18.2. A DATA UTASÍTÁS

A számítógép a DATA utasítást mindaddig figyelmen kívül hagyja, REM sornak tekinti, míg egy READ utasításra egy DATA elemet be nem kell olvasnia. Egyszerűbben fogalmazva: DATA sorok a program bármely részén lehetnek. A számítógép a program feldolgozásakor ezeket átugorja. A DATA utasításszó után álló elemeket egymástól vesszővel kell elválasztani. A karakterláncokat csak akkor kell idézőjelbe tenni, ha azok saját magukon belül vesszőt vagy kettőspontot tartalmaznak. Ezeket a jeleket a gép ugyanis elválasztó jeleknek tekintené.

18.3. A RESTORE UTASÍTÁS

Szintaxis:

RESTORE

vagy

RESTORE numerikus kifejezés

Ha egy programnak a DATA elemeket többször is el kell olvasnia, akkor a RESTORE utasítással a DATA mutatót a programon belül vissza lehet állítani. Ha csak a RESTORE-t adjuk ki paraméterek nélkül, akkor a mutató az első DATA elemre áll vissza.

A RESTORE sorszám utasítással a mutatót egy meghatározott programsorra állíthatjuk. Ezzel a kombinációval a mutatót a programban előre is lehet állítani. Arra ügyelni kell, hogy a megadott sorszám valóban létezzen. A sorszámot egy, a RESTORE után álló változó is tartalmazhatja.

Ezeket az utasításokat gyakran használják arra, hogy gépi kódban megírt programot POKE utasításokkal beírják a tárba. Ennek az az előnye, hogy azok is hasznosíthatják a gépi kódú programok gyorsaságát, akik magában a gépi kódú programozásban egyébként nem járatosak. Az eljárás lényege az, hogy a gépi kódokat DATA sorokban helyezzük el, majd ezeket egyetlen ciklussal beolvassuk a tárba. Hibák elkerülése végett célszerű, ha utolsó elemként valamilyen speciális értéket – pl. – 1-et,* választunk, és ezzel fejezzük be a ciklust. Ilyen ciklusok programozására a korábbi fejezetekben már mutattunk példákat.

19. Hibakezelés

Legkésőbb a hibakereséskor mindenkiről kiderül, hogy milyen a programozási stílusa. Aki kevés elágazást és sok REM sort tartalmazó programot írt, annak az eddigi többletmunkája most megtérül. Az ilyen programozást bizonyára gondos tervezés előzte meg, ami a későbbiekben a hibakeresését is megkönnyíti. Azt persze nem kell várni, hogy egy hosszabb program azonnal hibamentesen fut. A legtöbb hibaüzenetet kezdetben a SYNTAX ERROR. A HELP billentyűvel a hibát okozó sort a képernyőre írathatjuk. Sajnos villogva jelennek meg a karakterek, ami egy hosszabb programsor esetén a szemet fáraszthatja. Ilyenkor célszerű a sort a LIST sorszám utasítással a képernyőre írni.

A Plus/4-es BASIC nyelv számos hibaüzenettel segíti a felhasználó munkáját. Ezeknek az üzeneteknek a segítségével valamennyi beviteli hiba, sőt a logikai hibák egy része is kézben tartható. Így a programban elkövetett nagyobb baklövések igen gyorsan megtalálhatók és kijavíthatók. A hibaüzenetek áttekintését és az egyes üzenetek jelentését a könyv A és B függeléke tartalmazza. Vannak azonban olyan hibák, amelyeket a számítógép nem képes felismerni. Így pl. azt nem tudja megmondani, hogy a program által szolgáltatott érték megfelel-e a várakozásainknak. Ilyen jellegű hibákat okozhat az, hogy a programunkat előzőleg esetleg nem kifogástalanul terveztük meg. A következőkben arról lesz szó, hogy ezeknek hogyan juthatunk a nyomába.

A LIST utasítással írjuk a programunkat a képernyőre, majd állítsuk meg a listázást a CTRL/S billentyűkkel. Vizsgáljuk át sorról sorra a programot vagy a hibásnak feltételezett programrészt. Próbáljuk meg követni a program futását gondolatban. Ha így nem fedeztük fel a hibát, akkor ismételjük meg az egészet, vagy pedig járjunk el a következő pontban leírtak szerint. Ha valaki olyan szerencsés, hogy ki is nyomtathatja a programját, célszerű a listát ki is nyomtatni.

Mindenkinek ajánljuk azonban, hogy legalább a legfontosabb programrészeket, elágazásokat és változókat jegyezze fel magának. A – remélhetőleg elkészített – programterv alapján, különböző kiinduló adatokkal menjünk végig a program folyamatán. Vizsgáljuk meg, hogy a különböző programszakaszokban a várt eredményt kapjuk-e. Számos hibát már ebben a fázisban megtalálhatunk.

Ha még mindig maradt hiba a programban, akkor folytassuk a keresést. Egy-egy fontos programrész végére írjuk be a STOP utasítást. A program futása ezen a helyen meg fog szakadni. Most kiírathatjuk a legfontosabb változók értékeit, és megvizsgálhatjuk, hogy ezek helyesek-e. Természetesen ezeket a változókat magából a programból is ki lehet íratni. Ha a változók tartalma megfelelő, akkor a program futását a CONT utasítással folytathatjuk. Ha ily módon a program végére érünk, de a hibák okaira mégsem derült fény, akkor valahol a program logikája hibás (esetleg elágazási hibák). Ezek megkeresésére létezik egy speciális utasítás.

19.1. A TRON ÉS A TROFF

Szintaxis:

TRON

Ha kiadjuk ezt az utasítást, akkor a számítógép kiírja az éppen feldolgozás alatt levő sor számát. Ha egy sorban több, kettősponttal elválasztott utasítás vagy ciklus van, akkor a sorszámot annyiszor írja ki, ahányszor az adott sorban levő utasításokat hajtja végre. A gép a sorszámokat mindaddig kiírja, amíg egy TROFF utasítással nem találkozik.

Szintaxis:

TROFF

Ezzel az utasítással tehát kikapcsolható az aktuális sorszám kiírása.

Mindkét utasítás mind parancs, mind program módban, a program bármely részén kiadható. Az egyes programsorok feldolgozása azonban olyan gyors, hogy a sorszámok kiírását szemmel aligha lehet követni. A program futása a COMMODORE billentyűvel lassítható, illetve a CTRL/S lenyomásával megállítható.

19.2. HIBAKEZELÉS PROGRAMON BELÜL

Magán a programon belül is lehet reagálni a legkülönfélébb hibákra. Sőt, egy speciálisan programozott hibarutinban a fellépő hibát még semlegesíteni is lehet.

19.2.1. A TRAP és a RESUME utasítások

Szintaxis:

TRAP sorszám

Ezt az utasítást a programban ott kell kiadni, ahol hibákkal számolhatunk. Az utasításszó után egy sorszámnak kell állnia, amely egyben a hibakezelő rutin kezdő sorszáma. A TRAP nem ír ki hibaüzenetet, hanem a program futását elágaztatja a megadott sorszámra. Hasonlóan működik, mint a GOSUB. Egymásban ágyazott ugrások azonban nem használhatók, tehát magának a hibakezelő rutinnak hibátlannak kell lennie. A visszaugrás utasítása a RESUME.

Szintaxis:

RESUME

vagy

RESUME NEXT

vagy

RESUME sorszám

A RESUME hatására a program végrehajtása a hibát előidéző utasításra ugrik vissza.

A gép most ismét megpróbálja az utasítás végrehajtását. Ha ez a kísérlet sem sikerül, akkor ismét elágazik a hibakezelő rutinra, és minden kezdődik elölről.

A RESUME NEXT arra utasítja a számítógépet, hogy a program végrehajtását a hibaforrást követő utasítással folytassa. A hibaforrást nem veszi figyelembe.

A RESUME sorszám alakú utasítás azt a sorszámot adja meg, ahol a program futásának a hibakezelés után folytatódnia kell.

Hibaváltozók

Abból a célból, hogy a fellépő hibákra reagálni lehessen, a számítógépnek erre a célra lefoglalt változói vannak:

ER	Mindegyik hibaüzenethez tartozik egy szám. A hiba számát ez a változó tartalmazza, és a változóból kiolvasható.
EL	Ez a változó azt a sorszámot tartalmazza, amelyben a hiba fellépett. Ez is csak olvasható változó.
ERR$ (ER)	Ez a szöveges változó az ER által definiált hibaüzenetet tartalmazza. A hibaüzenet csak olvasható.

Nézzünk meg egy példát:

```
10 TRAP 100
20 PRINT "IRJON BE EGY 1 ES 9 KOZOTTI SZAMOT"
30 INPUT A
40 PRINT 100/A
50 GOTO 20
100 IF ER = 20 THEN PRINT" HIBAS ADAT ":RESUME 20
110 PRINT ERR$(ER) " HIBA. A HIBAS SOR SZAMA " EL
120 END
```

A 10-es sorban rögzítjük, hogy a programnak egy hiba bekövetkezésekor a 100-as sorra kell elágaznia. Ha a bekért számra 0-át adnánk meg, akkor normál esetben a DIVISION BY ZERO ERROR IN 40 hibaüzenetet kapnánk. A program azonban most a 100-as sorra ugrik. Mivel a hibaüzenet száma a 20, a képernyőre most a mi szövegünk íródik ki, és a végrehajtás visszakerül a 20-as sorra. A program futása tehát nem szakad meg.

Kövessünk el szándékosan egy szintaktikai hibát úgy, hogy a 20-as sorban levő PRINT helyett írjuk ezt: RINT. Indítsuk el a programot. Egy szintaktikai hibának más a hibaszáma, mint a 0-val való osztásé, ezért a programunk most a 110-es sorban kiírja a mi hibaüzenetünket: SYNTAX HIBA. A HIBAS SOR SZAMA 20. A 120-as sorban befejeződik a programunk. A TRAP...RESUME utasításpárral még a STOP billentyű hatása is kikapcsolható.

Bővítsük a programunkat:

```
105 IF ER = 30 THEN RESUME
```

Ha a program indítása után lenyomjuk a STOP billentyűt, akkor a program a 100-as sorra ágazik. A 105-ös sorban levő feltétel teljesül, tehát a RESUME hatására a program futása abban a sorban folytatódik, amelyben megszakítottuk. Na most aztán van egy örökmozgónk, ami soha sem állítható meg. Vagy mégis?

Nyomjuk le a RUN/STOP billentyűt, és tartsuk lenyomva. Nyomjuk be most eközben a RESET gombot is. Ezután előbb engedjük el a RESET gombot, majd

utána a másikat. A képernyő megváltozott. Most a gépi kódú monitorprogramban vagyunk. Ebből a programból az X beírásával és a RETURN lenyomásával lehet kilépni. A programunk még mindig a gépben van, a LIST utasítással erről meg is győződhetünk. Próbálja meg most mindenki saját maga, hogy a program a 22-es hibaszámra reagáljon. Az ennek megfelelő hibaüzenet a TYPE MISMATCH ERROR, amit akkor kapunk, ha numerikus érték helyett egy karaktert, vagy karakter helyett numerikus értéket írtunk be.

Az itt leírt utasításokkal és fogásokkal bizonyára felderíthető egy programban előforduló összes hiba. Akkor se essünk kétségbe, ha ez az első próbálkozásra nem is sikerülne. Keljünk fel a gép mellől és nyújtózzunk vagy járjunk egyet. Kis szünet után biztosan jobban fog menni.

20. A Plus/4-es gépi kódú programozása

A Plus/4-es tartalmaz gépi kódú monitorprogramot, amelyet a következőkben ismertetünk. Előbb azonban tisztázzuk azt, hogy mi is valójában az a gépi kód. A számítógép felépítéséről szóló fejezetben láttuk, hogy a számítógép csak nullákkal és egyesekkel tud dolgozni. A mikroprocesszor adatbusza ugyanis ettől eltérő kapcsolási állapotot nem tud felvenni. Hasonló a helyzet a mikroprocesszorra érkező utasításoknál is. Az egyes utasítások az adatbuszon különböző kapcsolási állapotokat hoznak létre. Mivel az adatbusz 8 bit széles – ami a gyakorlatban azt jelenti, hogy nyolc vezetékből áll, és mindegyik vezeték 1-1 bitet ábrázol – a mikroprocesszor elméletileg maximum $2^8 = 256$ utasítást tud megkülönböztetni, azaz pontosan annyit, ahány különböző kapcsolási állapot az adatbuszon létrejöhet. A valóságban azonban a mikroprocesszor csak kevesebb utasítást (úgynevezett műveleti kódot) ismer. Van olyan kód, amikor a mikroprocesszor „lemerevedik". Ezért vigyázni kell arra, hogy a mikroprocesszor csak olyan műveleti kódokat kaphasson, amelyeket az utasításkészletében ténylegesen tartalmaz.

Az utasítások felosztathatók adatmozgató, aritmetikai, logikai, összehasonlító, elágazó, ugró, eltoló utasításokra, az állapotregiszterre vonatkozó utasításokra, ezek kombinálására és néhány más utasításra. Ezeket az utasításokat az adatbuszon keresztül kell a mikroprocesszorral közölnünk. Ehhez elegendő lenne, ha az adatbusz mind a nyolc vezetékére egy-egy kapcsolót kötnénk, és ezek megfelelő állításával hoznánk létre a kívánt kapcsolási állapotokat. Ez persze rendkívül körülményes és lassú lenne. Az egyes vezetékeknek (azaz a biteknek) ezeket a kapcsolgatásait a memóriában levő programok végzik el helyettünk. Azaz a monitorprogram a memóriába ír, majd a vezérlést átadja a beírt programnak. A monitor program a beírást segíti, itt a számok hexadecimális alakját használjuk. A gépi kódú program tehát egy sor számból áll, amelyek egy része a parancsokat, más része pedig az adatokat jelenti. A gépi kód ugyan meglehetősen egyszerű és könnyen érthető utasításokat tartalmaz, de az egész együtt, mint komplett program, nagyon áttekinthetetlen. Ez ugyanis a BASIC-től eltérően nem világosan érthető szöveggel, hanem absztrakt számokkal dolgozik. Úgy is meg lehet fogalmazni, hogy míg a BASIC a felhasználót támogató programozási nyelv, addig a gépi kódú nyelv a mikroprocesszor képességeit használja ki. Ebből következően a gépi kódnak egy óriási előnye van a BASIC-kel szemben, nevezetesen a lényegesen nagyobb sebesség.

Nézzünk meg ehhez egy kis példaprogramot:

Legyen a feladat, hogy 255-től 0-ig visszafelé számolunk, és a mindenkori számot elhelyezzük a $0003 tárcímen. Ezt a következő gépi kódú program végzi el:

A9 FF 85 03 C6 03 D0 FC 00

Az egyes utasításoknak megfelelő absztrakt számokkal való helyettesítésére alkalmas

az úgynevezett assembler írásmód. Ennél az egyes utasításokat nem a mikroprocesszor által is érthető számokként, hanem az utasítások angol nyelvű megnevezésére utaló betűrövidítésekkel (az úgynevezett mnemonikokkal) kell megadni. Ez nagyjából összehasonlítható azzal, hogy a számítógép a BASIC nyelvű programokban levő BASIC utasításszavakat nem a beírásuknak megfelelően, hanem az úgynevezett BASIC-tokenjüknek megfelelő, kódolt formában tárolja a tárban.

A példaprogramunk így írható assemblerben:

```
LDA #$FF
STA $03
DEC $03
BNE $FC
BRK
```

Ez már egy kicsit jobban néz ki, mint az előző számsorunk! A programunk 9 byte hosszúságú, és a végrehajtáshoz pontosan 1802 ütemciklusra van szüksége, ami nagyjából 1,8 ezredmásodpercnek felel meg. A mikroprocesszor az ütembemenetén négyszögimpulzusokat kap. Mindegyik impulzus egy-egy ütemciklus, amely a mikroprocesszort a munkájában egy-egy ütemmel tovább lépteti. A gépi kódú utasítások végrehajtásához 2-7 ütemnyi „időre" van szükség. Mivel a példaprogramunkban a BASIC nyelv FOR ... NEXT utasításpárjához hasonló ciklust is használtunk, a program végrehajtási ideje 1802 ütem.

A példaprogramunk BASIC nyelven így írható:

```
10 FOR K=255 TO 0 STEP -1
20 POKE 3,K
30 NEXT K
```

Ez a program sem foglal el túl sok helyet a tárban. Gondoljuk meg azonban, hogy önmagában ez a BASIC program a kitűzött feladatot nem teljesíti. Ehhez szükség van még a BASIC interpreterre, ami néhány kbyte-ot foglal el a tárban. Ezzel az interpreterrel számos rutint kell feldolgozni ahhoz, hogy a rövid kis BASIC programunk valóban működjön. Ez az oka annak is, hogy a BASIC programnak több mint egy másodpercre van szüksége ahhoz, hogy a 3-as tárcímre beírja a 255–0 közötti értékeket. Ez a gépi kódú program végrehajtásához képest mintegy 500-szoros idő!

Ez az egyszerű példa bizonyára mindenkit meggyőz arról, hogy a gépi kódú programnyelv a kevésbé áttekinthetősége ellenére a végrehajtási sebességet illetően a BASIC nyelvhez képest lényegesen előnyösebb. A gépi kódok beírásához a gépi kódú monitorprogramot lehet használni, aminek a Plus/4-es számítógépben TEDMON a neve.

20.1. A TEDMON

Ez a program lehetővé teszi, hogy a számítógépbe gépi kódú programokat írjunk és ezeket tárolhassuk, vagy gépi kódú programokat betöltsünk, és ezeket indíthassuk. Magát a TEDMON monitorprogramot a

MONITOR

utasítás beírásával indíthatjuk. Az utasítás az M és a SHIFT/O beírásával rövidíthető.

A TEDMON számos utasítást tartalmaz a gépi kódú programozáshoz:

```
A (Assemble)     = assemblálás
C (Compare)      = összehasonlítás
D (Disassemble)  = disassemblálás
F (Fill)         = feltöltés
G (Go)           = indítás
H (Hunt)         = keresés
L (Load)         = betöltés
M (Memory)       = tártartalom kiírás
R (Regiszter)    = regisztertartalmak kiírása
S (Save)         = tárolás
T (Transfer)     = adatmozgatás
V (Verify)       = külső/belső tár összehasonlítása
X (eXit)         = kilépés a monitorprogramból
>
;
.
```

A TEDMON A (assemble) utasítása

Ez az utasítás a gépi kódú programok assembly nyelvű beírását teszi lehetővé.
Az A szintaxisa:

A cím utasítás (operandus)

cím	az a tárcím (hexadecimális alakban), amelyiken az utasítást tárolni kell.
utasítás	az utasítás assembler írásmódja.
operandus	a gépi kódú utasítások egy-, két- vagy hárombyte-osak lehetnek. Egybyte-os utasításoknál operandus nem adható meg. Kétbyte-os utasításnál egy operandust, hárombyte-os utasításnál egy kétbyte-os címet kell megadni. (Az utasítás első byte-ja mindig valamelyik műveleti kódot jelenti.)

Példák:

A 3000 JMP $3000

Ez az utasítás végtelen ciklust eredményez, mivel itt a JMP (JUMP = ugrás) utasítással ismét visszaugrunk a $3000-es címre. A JMP hárombyte-os utasítás.

A 3000 LDA $FF

A LDA kétbyte-os utasítás. Ez az utasítás a processzor egyik regiszterébe, az akkumulátorba a $FF értéket tölti.

A 3000 BRK

A BRK egybyte-os utasítás, melynek hatására a program futása megszakad (úgynevezett szoftver-interrupt). Az A utasítást egy ponttal (.) is lehet helyettesíteni. Egy assembly utasítássor bevitele után a képernyőn megjelenik az utasítás hexadecimális formában (tulajdonképpen ez a gépi kód), valamint a tárban következő, írható cím.

A TEDMON C (compare) utasítása

A C utasítással két tártartomány tartalma byte-ról byte-ra összehasonlítható.
A C szintaxisa:

C 1.cím 2.cím 3.cím

1. cím az összehasonlítandó tártartomány kezdőcíme.
2. cím ennek a tártartománynak a végcíme.
3. cím annak a tártartománynak a kezdőcíme, amellyel az első tartományt össze kell hasonlítani.

A TEDMON D (disassemble) utasítása

Ez az utasítás az A utasításnak pontosan az ellenkezőjét végzi. A tárban levő, gépi kódú programot visszafordítja assembly nyelvű programmá (ezt az eljárást nevezik disassemblálásnak vagy visszaassemblálásnak).
A D szintaxisa:

D cím (végcím)

cím a disassemblálandó tártartomány kezdőcíme.
végcím a disassemblálandó tártartomány végcíme.

Példa:

D 9000

Az utasítás hatására a monitorprogram a $9000-res címtől kezdi a disassemblálást. A képernyőre egyszerre maximum 21 sor íródik ki. A disassemblálás folytatható, ha csak a D utasítást írjuk be (cím nélkül). Ha viszont végcímet is megadunk, akkor a disassemblálás a kezdőcímtől a végcímig megtörténik.

A TEDMON F (fill) utasítása

Ezzel az utasítással lehet legegyszerűbb módon egész tártartományokat egy meghatározott értékkel feltölteni.
Az F szintaxisa:

F 1.cím 2.cím érték

1. cím A feltöltendő tártartomány kezdőcíme.
2. cím A tartomány végcíme.
érték Hexadecimális érték, amellyel a tartományt fel kell tölteni.

Példa:

F 2000 3000 F1

Az utasítás hatására a $2000 és a $3000 között tárcímek mindegyikének (a határokénak is) $F1 lesz a tartalma.

A TEDMON G (go) utasítása

Ezzel az utasítással lehet a gépi kódú programokat indítani. Ilyenkor az R-rel előhívható regiszterek értéke bemásolódik a processzor regisztereibe.

A G szintaxisa:

G tárcím

A G utasítással természetesen ROM-ban levő programok is indíthatók.

A TEDMON H (hunt) utasítása

Ezzel a nagyon hasznos utasítással egy tártartományban megkereshetők bizonyos adatok.

A H szintaxisa:

H kezdőcím végcím adatok

kezdő- és végcím	A vizsgálandó tártartomány kezdő- és végcíme.
adatok	A tartományban keresendő adatok. Ezek maximum 27 byte-ból álló byte-sorozatok lehetnek, de akár egy-egy byte is. Az egyes byte-okat szóközzel kell egymástól elválasztani.

Az adatok helyett karakterlánc is kereshető. A karakterlánc elé (amely maximum 32 karaktert tartalmazhat) egy aposztróf jelet kell tenni.

Példa:

H 8000 FFFF 20 DC A0

Ez az utasítás a $8000–$FFFF közötti tártartományban a $20 $DC $A0 byte-sorozatot keresi.

A számítógép azokat a kezdőcímeket írja ki, ahol a byte-sorozat első byte-ja áll:

A05C A066 A06C A072
H 8000 FFFF ' CBM

A számítógép most a megadott tartományban a CBM karakterláncot keresi. A következő tárcímeken találja:

8007 FC56

A TEDMON L (load) utasítása

Ezzel az utasítással tölthetők be programok kazettáról vagy hajlékony lemezről.

Az L szintaxisa:

L ("(d:) program neve" (,n))

d	A lemezmeghajtó száma, 0 vagy 1. Ha csak egy meghajtóval dolgozunk, akkor ezt nem szükséges megadni.
n	Egységszám, kazettás magnó esetén 1, lemezmeghajtó esetén 8. Ha a programot kazettáról töltjük be, akkor nem szükséges megadni.

Ha kazettáról a soron következő, legelső programot akarjuk betölteni, akkor elegendő csak az L beírása. (Ez a programot attól a tárcímtől kezdődően tölti be, ahonnan a kimentés is történt.)

Példa:

L "GRAFIKA",8

Lemezről betölti a "GRAFIKA" nevű programot.

L "JATEK"

Kazettáról betölti a "JATEK" nevű programot.

A TEDMON H (hunt) utasítása

Ezzel az utasítással a tár tetszőleges részét megjeleníthetjük a képernyőn. A képernyőre íródik az első byte címe és ezzel egy sorban nyolc byte, majd az ezeknek megfelelő CHR$-kódok (inverz alakban), illetve pont (.) nem látható a karakterek helyett.
Az M szintaxisa:

M (kezdőcím (végcím))

kezdőcím Ha csak ezt adjuk meg, akkor 96 byte íródik a képernyőre.
végcím Ha mindkét címet megadjuk, akkor a kettő közötti összes byte a képernyőre íródik.

A kiírás a COMMODORE billentyűvel lassítható, a CONTROL/S ideiglenesen, a RUN/STOP billentyűvel pedig végleg megállítható. Ha ezután csak az M utasítást írjuk be, akkor a kiírás folytatódik. Nézzük meg most a $8007-től kezdődő tárterületet. Amint a H utasításnál mutatott példánál láttuk, a $8007-es tárcímtől kezdődően a CBM karakterláncnak kell állnia. Írjuk be:

M 8007

A képernyőn most 12, egyenként nyolc-nyolc byte-ot tartalmazó sor jelenik meg. Az első három byte valóban a CMB karaktereknek felel meg. A kiírás folytatásához most elegendő csak az M betű beírása és a RETURN lenyomása. Most folyamatosan kiíródik a következő 12 sor. Az

M 8000 8100

utasítás hatására képernyőre íródik a $8000 és a $8100 közötti teljes tárterület tartalma.

Tárcímek tartalmának megváltoztatása

A > (nagyobb mint) jel és egy tárcím beírásával ettől a tárcímtől kezdődően egyszerre egy–nyolc tárcím tartalma is megváltoztatható. A megadott kezdőcím mellé egy sorban, egymástól szóközzel elválasztva be kell írni a címek új tartalmát. Az M utasítás beírása esetén a > jel automatikusan megjelenik. Ekkor a kurzorbillentyűkkel a módosítani kívánt tárcímnek megfelelő helyre kell állni, és az ott levő tartalmat az

új tartalommal át kell írni (csak a hexadecimális rész módosítható). A RETURN billentyű lenyomására a tárcím tartalma megváltozik, és végrehajtja az adott sorra az M utasítást (kiír egy sor hex + CHR-t)
Szintaxis:

> cím egy-nyolc byte

A TEDMON R (registers) utasítása

Ezzel az utasítással kiírathatók a processzor regiszterei és azok tartalma. Ehhez elegendő csak az R beírása. Ekkor a képernyőn megjelenik a PC programszámláló, az SR állapotregiszter, az AC akkumulátor, az XR X regiszter, az YR Y regiszter és az SP veremmutató regiszter tartalma hex alakban.

A regiszterek tartalmának változtatása

Az R utasítással kiírt processzorregiszterek tartalma meg is változtatható. Ehhez elegendő a ; (pontosvessző) beírása. A kurzorbillentyűkkel ekkor a kívánt helyre kell állni, és az ott levő tartalmat egyszerűen át kell írni az új tartalommal. Ekkor még nem változik meg a processzor tartalma. A h utasítás módosít.

A TEDMON S (save) utasítása

Ezzel az utasítással a számítógép tárának egy meghatározott része kazettára vagy lemezre írható.
Szintaxis:

S „program neve",n,kezdőcím,végcím + 1

Példa:

S "RUTINOK",1,2000,21FF

Ezzel az utasítással a számítógép $2000 és $21FE tárcímek közötti részét RUTINOK néven kazettára tároljuk.

S "ADATOK",8,1700,2400

Az utasítás a tár $1700 és $23FF címek közötti tartalmát ADATOK néven lemezre írja. A tárterület programállományként kerül a lemezre. Ne feledkezzünk meg arról, hogy a végcímhez 1-et hozzá kell adni, különben lemarad az utolsó byte.

A TEDMON T (transfer) utasítása

Ezzel az utasítással egy tárterület tartalma átmásolható a tár egy másik területére.
Szintaxis:
T 1.cím 2.cím 3.cím

1.cím az átmásolandó terület kezdőcíme
2.cím az átmásolandó terület végcíme
3.cím az új terület kezdőcíme

Példa:

T 3000 3100 1800

Az utasítás a $3000 és az $3100 közötti területet átmásolja a $1800 és az $1900 közötti területre.

A TEDMON V (verify) utasítása

Ilyen utasítással már találkoztunk. A V egy kazettán vagy lemezen tárolt állományt összehasonlít a számítógép megfelelő tárterületén levő tartalommal. Ha a két tartalom nem egyezik, hibaüzenetet kapunk.

Szintaxis:

V (("program neve",n))

n egységszám, kazetta esetén 1, lemez esetén 8.

Név, eszköz elhagyható kazetta esetén.

A TEDMON X (eXit) utasítása

Ezzel az utasítással kilépünk a monitorprogramból, és visszatérünk a BASIC-be. Az esetlegesen már előzőleg a tárban levő BASIC program nem változik, hacsak a monitorprogramból az általa elfoglalt tárat mi magunk nem változtattuk meg.

20.2. A SYS BASIC-UTASÍTÁS

A gépi kódú programok BASIC-ből például a SYS utasítással indíthatók. Az utasítás mind közvetlen, mind program módban használható. Lehetőségünk van tehát arra, hogy BASIC programokon belül gépi kódú rutinokat is futtassunk.

Szintaxis:

SYS cím

A cím a gépi kódú program tárbeli kezdőcíme decimális alakban. A címet természetesen pontosan kell megadni, mert ellenkező esetben a számítógép viselkedése ellenőrizhetetlenné válhat. A SYS utasítással BASIC- vagy rendszer ROM-beli programok is indíthatók. Nem indíthatók viszont olyan programok, amelyek a $8000–$FFFF közötti RAM-területen vannak. Erről részletesebben a 21. fejezetben (memórialapozás) lesz szó.

Írjuk be a SYS 52651 utasítást. A képernyőn négy név jelenik meg. Ők fejlesztették ki a gép BASIC nyelvét.

Ez az utasítás nem teszi lehetővé, hogy a gépi kódú programoknak változókat adjunk át. Van azonban egy másik lehetőség arra, hogy egy BASIC programon belül gépi kódú programokat használjunk. Ez az USR utasítás.

20.3. AZ USR BASIC-UTASÍTÁS

Ezzel az utasítással szintén indíthatók gépi kódú programok. Ezeknek a programoknak értékek is átadhatók, amelyekkel további számítások végezhetők.
Szintaxis:

USR (változó)

Amint látjuk, maga az utasítás nem tartalmaz kezdőcímet. Persze erre azért szükség van, ezért ezt már az utasítás kiadása előtt közölni kell a géppel. A kezdőcímet a (decimális) 1281-es és 1282-es tárcímekre kell beírni, a BASIC POKE utasításával. Mivel egy tárcím 16 bites, és egy címre csak 8 bites érték írható, a tárolandó címet két részre kell osztani.

Ez a következőképpen történik: A címet 256-tal osztjuk, és a hányadost kerekítjük. BASIC nyelven a számítás a

H = INT (cím/256)

képlettel végezhető. Az eredmény az úgynevezett felső, vagy magasabb helyi értékű (high) byte. Ezt az értéket kell az 1282-es tárcímre írni.

Még ki kell számítani az alsó, vagy alacsonyabb helyi értékű (low) byte értékét. A képlet:

L = cím − 256 * H

Ezt az értéket az 1281-es címre kell írni. Most már indítható a gépi kódú program a USR utasítással. Az utasítással átadott változó az úgynevezett lebegőpontos akkumulátorba (FAC) kerül, ahonnan tovább feldolgozható. Amikor a gépi kódú programrészből egy RTS utasítással visszatérünk a BASIC programba, a változóba az az érték kerül vissza, ami a lebegőpontos akkumulátorban áll.

Az USR utasítást ritkábban használják, mint a SYS-t. Ez utóbbival is adhatók át értékek a BASIC-ből gépi kódú programnak, bár ez egy kicsit körülményesebb. Nagy előnye viszont az, hogy több érték is átadható. Erre vonatkozó további részletek a következő fejezetekben, ill. a mikroprocesszor gépi kódú programozásával foglalkozó szakirodalomban találhatók.

20.4. BEVEZETÉS A GÉPI KÓDÚ PROGRAMOZÁSBA

A könyv eddigi részében csak a BASIC nyelvű programozással foglalkoztunk. Ebben a fejezetben már megismerkedtünk a gépi kódú monitorral. Ahhoz azonban, hogy ezt használni is tudjuk, meg kell tanulnunk a gépi kódú programozás alapelemeit.

A gépi kódú programozási nyelv olyan utasításokból áll, amelyeket a számítógépben levő mikroprocesszor közvetlenül megért. Tehát nem PRINT, hanem „Load Accu" vagy „Jump" és ehhez hasonlók. Idézzük emlékezetünkbe a 2. fejezetben a számítógép alapvető felépítéséről elmondottakat: ott azt mondtuk, hogy a számítógép a BASIC utasításokat csak egy értelmező program, az úgynevezett BASIC interpreter segítségével képes megérteni. Ennek az értelmező programnak az a feladata, hogy a BASIC utasításokat egy sereg gépi kódú utasítássá alakítsa át.

A beépített monitorprogram lehetővé teszi, hogy a gépi kódú utasításokat közvetlenül beírjuk a gépbe. Itt jegyezzük meg, hogy ez a témakör önmagában is könnyen megtölthet egy teljes könyvet. Ezért az egyes témákkal kénytelenek vagyunk nagyon tömören foglalkozni, ugyanakkor igérjük, hogy valamennyi lényeges kérdésre kitérünk. Először megismerkedünk a 7501-es mikroprocesszor felépítésével, ezt követően a különböző utasításokkal és címzési módokkal, majd a fejezet végén bemutatunk néhány mintapéldát. Lássuk tehát először a mikroprocesszor felépítését.

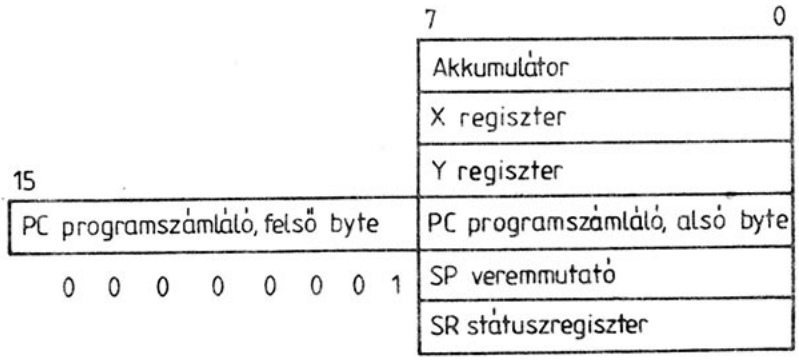
20.5. A 7501-ES MIKROPROCESSZOR FELÉPÍTÉSE

A Plus/4-es számítógépbe egy 7501-es mikroprocesszort építettek, amely szoftver oldalról kompatíbilis a jól ismert 6502-es mikroprocesszorral. A processzor egy 40 csatlakozós tokban helyezkedik el.

A 7501-es processzor különlegessége, hogy hét, programozható beviteli és kiviteli csatlakozóval rendelkezik. Ezeket a Plus/4-es számítógép a soros interface-hez és a kazettás egység interface-hez használja. Ennek a hét I/O (Input/Output) vezetéknek a jelölése: P0, P1, P2, P3, P4, P5 és P7. A P2 vezeték egy inverteren keresztül a User porthoz (felhasználói kapu) is hozzá van kötve, ahol a jelölése ATN.

A P0...P7 vezetékek a $0000 és a $0001 címeken keresztül programozhatók. A $0000 az adatirány regiszter, míg a $0001 az úgynevezett adatregiszter. Ha az adatirány regiszter valamelyik bitje magas (értéke 1), akkor az ennek megfelelő bemeneti/kimeneti (I/O) vezeték kimenetként működik. Ennek a vezetéknek az állapotát a $0001 címen levő adatregiszter megfelelő bitjének az értéke határozza meg. Ha ennek a bitnek az értéke 0, akkor az illető vezetéken levő jel szintje alacsony (low), ha 1, akkor magas (high). Ha az adatirány regiszter valamelyik bitje alacsony, akkor az ehhez tartozó vezeték bemenet, az állapotát pedig ugyancsak az adatregiszter megfelelő bitje határozza meg.

Lássuk most a mikroprocesszor regisztereit. A CPU-nak hat regisztere van. Vizsgáljuk meg sorra ezeket.



20.1. ábra A 7501-es mikroprocesszor regiszterei

Az akkumulátor egy 8 bites regiszter. Ez a CPU legfontosabb regisztere. A fejezetben még többször visszatérünk az akkumulátorra és a programozására. Az akkumulátor – többek között – annak köszönheti kitüntetett szerepét, hogy több művelet, így pl. a logikai és az aritmetikai műveletek csak ebben végezhetők el, és a műveletek eredményei is itt állnak rendelkezésre.

Az X és az Y regiszterek szintén 8 bites regiszterek. Ezekkel a regiszterekkel pl. indexelt címzések végezhetők. A későbbiekben még ezekre is visszatérünk.

A programszámláló két, 8 bites regiszterből áll. Ezekben a regiszterekben mindig az éppen végrehajtás alatt levő utasítás kezdőcíme van. Szigorúan véve a programszámláló egy utasítás végrehajtásakor már a következő utasítás címét tartalmazza, mert a CPU először beolvassa az utasítás- és az adatbyte-o(ka)t, majd ezeket értelmezi és végrehajtja. Eközben a programszámláló már a következő utasításbyte-ra mutat. A programszámlálóval egyébként – az elágazó utasításoktól eltekintve – nem kell törődnünk.

A következő, szintén 8 bites regiszter az úgynevezett veremmutató regiszter (SP, stack pointer). Ennek a regiszternek egészen különleges rendeltetése van. Ez a regiszter lehetővé teszi, hogy a CPU egy külön erre a célra szolgáló tárterületen adatokat tároljon, és azokat onnan visszaolvassa. Ez a speciális tárterület, amelynek veremtár a neve, a $0100-$01FF tárcímek között van. A 8 bites regiszterrel azonban csak $00–$FF közötti címek állíthatók elő, ezért ehhez a cím felső byte-ját (%00000001) a CPU automatikusan hozzáadja (lásd. 20.1 ábra). Jól tudjuk ugyanis, hogy egy tárcím 16 bites.

A veremmutató $FF-től kezdve, visszafelé számlál, azaz a számítógép bekapcsolásakor az értéke $FF. A veremtárba kerülő adatok tárolása tehát a $01FF címtől kezdődik. A veremtárból való olvasáskor viszont először az utoljára beírt adatok vehetők ki. Természetesen egy korábban beírt adat is kivehető, de ez egy kicsit körülményes. Ehhez a veremmutatót egy speciális utasítással át kell állítani, ami nagy elővigyázatot követel, mert a verem könnyen „túlcsordulhat", ill. a program „fejre állhat".

Lássuk most a következő regisztert, az állapotregisztert. Ez a regiszter a CPU aktuális állapotát írja le. Az állapotregiszterben levő minden egyes bitnek különkülön jelentése van. Ezeket a biteket jelzőbiteknek (flag) szokás nevezni, mert mindegyik valamilyen állapotot jelez. Ezek a bitek speciális utasításokkal törölhetők (0) ill. bekapcsolhatók (1). A jelzőbitek állását egyébként a legtöbb gépi kódú utasítás megváltoztatja.

Az egyes bitek jelentése.

0. bit átvitel bit (Carry-Flag). A bit értéke akkor 1, ha egy utasítás végrehajtásakor az illető regiszter 7. bitjén átvitel keletkezik, ha pl. egy összehasonlító utasítás végrehajtásakor a megcímzett rekesz tartalma a processzorregiszter tartalmánál nagyobb vagy azzal egyenlő.

1. bit zéró bit (Zero-Flag), értéke akkor 1, ha egy számítási művelet eredménye zérus, vagy ha az összehasonlító utasítás operandusai egyenlők.

2. bit megszakítás bit (Interrupt-Flag), azt mutatja, hogy a processzor elfogad megszakítás kérést (IRQ) vagy nem. Ha a bit értéke 0, akkor a processzor engedélyezi a megszakítást.

3. bit decimális jelzőbit (Decimal-Flag), azt mutatja, hogy az adatokat összeadáskor és kivonáskor kettes vagy tizes számrendszerbeli számokként kezeli. A bit 0 értéke esetén kettes számrendszerben számol.

4. bit Break–jelzőbit. Az mutatja meg, hogy egy program megszakítása szoftver oldalról, egy BRK utasítás hatására történt-e. Ha igen, akkor a bit értéke 1.

5. bit nincs szerepe, az értéke mindig 1.

6. bit túlcsordulás jelzőbit (oVerflow-Flag), értéke akkor 1, ha egy összeadásnál vagy kivonásnál az akkumulátorban a 6. bitről átvitel keletkezik a 7. bitre. A BIT utasítás is állítja ezt a jelzőbitet.

7. bit negatív jelzőbit (Negativ-Flag), értéke akkor 1, ha az eredmény negatív.

20.6. A NULLÁSLAP (ZERO-PAGE)

A nulláslap a számítógép tárában a $0000–$00FF közötti tárterületet jelenti. Ez a terület közvetlenül a veremtár alatt van. A nulláslapon levő tárcímekhez speciális utasításokkal, közvetlen módon lehet fordulni. Ennek az az előnye, hogy az utasításban elegendő a tárcímnek csak az alsó byte-ját megadni (hasonlóan a veremtár címzéséhez). A tárcím felső byte-ját (%00000000) a CPU ehhez automatikusan hozzáteszi. Ezáltal egyrészt tárhelyet takarítunk meg a gépi kódú programban, másrészt a végrehajtási idő is csökken. A nulláslapon levő tárcímek mind a 6502-es mikroprocesszorra épülő számítógépeknél, mind a Plus/4-esnél átmeneti tárhelyként szolgálnak. Emellett vannak olyan, gépi kódú utasítások, amelyek speciális címzési módokban csak a nulláslap tárcímeit használhatják.

20.7. A 7501-ES MIKROPROCESSZOR UTASÍTÁSKÉSZLETE

A 7501-es mikroprocesszor terjedelmes utasításkészletében levő utasítások funkcióik szerint a következő csoportokba oszthatók:

- adatmozgató utasítások,
- állapotregiszterre vonatkozó utasítások,
- aritmetikai utasítások,
- logikai utasítások,
- elágazó utasítások,
- ugró utasítások,
- összehasonlító utasítások,
- eltolási utasítások,
- egyéb utasítások.

Az utasítások írásához az úgynevezett assembler szintaxist használjuk. Ez azt jelenti, hogy az egyes utasításokat nem hexadecimális számok formájában, hanem a funkciójukra utaló, angol nyelvű kifejezésekből alkotott rövidítésekkel adjuk meg. Mindegyik rövidítés három betűből áll. Ezeket a rövidítéseket a számítástechnikai gyakorlatban mnemonikoknak nevezik. A következőkben felsoroljuk a mikroprocesszor

utasításait, a hozzájuk tartozó mnemonikokkal együtt. Zárójelben megadjuk a jelentésüket, valamint az utasítások által befolyásolt jelzőbiteket.

Az M függelék tartalmazza az összes utasítást a különböző címzésmódoknak megfelelő utasításkódokkal és a befolyásolt jelzőbitekkel együtt.

Adatmozgató utasítások

LDA	(érték betöltése az akkumulátorba, N,Z)
STA	(akkumulátor tartalmának áttöltése)
LDX	(érték betöltése az X regiszterbe, N,Z)
STX	(X regiszter tartalmának áttöltése)
LDY	(érték betöltése az Y regiszterbe, N,Z)
STY	(Y regiszter tartalmának áttöltése)
TAX	(akkumulátor tartalmának betöltése az X regiszterbe, N,Z)
TAY	(akkumulátor tartalmának betöltése az Y regiszterbe, N,Z)
TXA	(X regiszter tartalmának betöltése az akkumulátorba, N,Z)
TYA	(Y regiszter tartalmának betöltése az akkumulátorba, N,Z)
TXS	(X regiszter tartalmának betöltése a veremmutatóba)
TSX	(veremmutató tartalmának betöltése az X regiszterbe, N,Z)
PLA	(veremtár tartalmának betöltése az akkumulátorba, N,Z)
PHA	(akkumulátor tartalmának betöltése a veremtárba)
PLP	(veremtár tartalmának betöltése az állapotregiszterbe)
PHP	(állapotregiszter tartalmának betöltése a veremtárba)

Állapotregiszterre vonatkozó utasítások

CLC	(átvitel a jelzőbit törlése, C = 0)
SEC	(átvitel a jelzőbit beállítása, C = 1)
CLD	(decimális jelzőbit törlése, D = 0)
SED	(decimális jelzőbit beállítása, D = 1)
CLI	(megszakítás jelzőbit törlése, I = 0)
SEI	(megszakítás jelzőbit beállítása, I = 1)
CLV	(túlcsordulás jelzőbit törlése, V = 0)

Aritmetikai utasítások

ADC	(akku tartalmához érték és az átvitelbit hozzáadása, N,Z,C,V)
SBC	(akku tartalmából érték és az átvitelbit kivonása, N,Z,C,V)
INC	(adott tárcím tartalmának növelése 1-gyel, N,Z)
DEC	(adott tárcím tartalmának csökkentése 1-gyel, N,Z)
INX	(X regiszter tartalmának növelése 1-gyel, N,Z)
DEX	(X regiszter tartalmának csökkentése 1-gyel, N,Z)
INY	(Y regiszter tartalmának növelése 1-gyel, N,Z)
DEY	(Y regiszter tartalmának csökkentése 1-gyel, N,Z)

Logikai utasítások

AND (És művelet adott érték és az akku tartalma között, N,Z)
ORA (VAGY művelet adott érték és az akku tartalma között, N,Z)
EOR (KIZÁRÓ VAGY művelet adott érték és az akku tartalma között, N,Z)

Elágazó utasítások

BCC (elágazás, ha a C jelzőbit = 0)
BCS (elágazás, ha a C jelzőbit = 1)
BNE (elágazás, ha a Z jelzőbit = 0)
BEQ (elágazás, ha a Z jelzőbit = 1)
BPL (elágazás, ha az N jelzőbit = 0)
BMI (elágazás, ha az N jelzőbit = 1)
BVC (elágazás, ha a V jelzőbit = 0)
BVS (elágazás, ha a V jelzőbit = 1)

Ugró utasítások

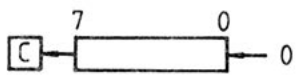
JMP (ugrás egy megadott tárcímre)
JSR (ugrás egy alprogramra)
RTS (visszatérés alprogramból)
RTI (visszatérés megszakításkezelő alprogramból)

Összehasonlító utasítások

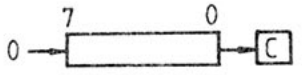
CMP (adott érték és az akku tartalmának összehasonlítása, N,Z,C)
CPX (adott érték és az X regiszter tartalmának összehasonlítása, N,Z,C)
CPY (adott érték és az Y regiszter tartalmának összehasonlítása, N,Z,C)
BIT (ÉS művelet adott érték és az akku tartalma között, az akku tartalmának megváltoztatása nélkül, Z az eredménytől függően változik: ha a művelet eredménye $00, akkor Z = 1,N-be az eredmény 7. bitje, V-be az eredmény 6. bitje kerül)

Az összehasonlító utasítások csak az állapotregiszterben levő jelzőbiteket állítják.

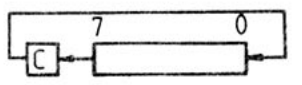
Eltolási utasítások



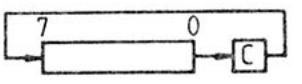
ASL (akku vagy tárcím tartalmát egy hellyel balra tolja, a 0. bit értéke 0 lesz, a 7. bit értéke a C bitbe kerül, N,Z,C)



LSR (akku vagy tárcím tartalmát egy hellyel jobbra tolja, a 7. bit értéke 0 lesz, a 0. bit értéke a C bitbe kerül, N,Z,C)



ROL (akku vagy tárcím tartalmát egy hellyel balra tolja, a 0. bitbe a C bit értéke, a C bitbe a 7. bit értéke kerül, N,Z,C)



ROR (akku vagy tárcím tartalmát egy hellyel jobbra tolja, a 7. bitbe a C bit értéke, a C bitbe a 0. bit értéke kerül, N,Z,C)

Egyéb utasítások

NOP (No Operation, üres utasítás, nincs semmilyen hatása)
BRK (úgynevezett szoftver megszakítás. Beállítja a megszakítás jelzőbitet és megszakítja a program futását. A megszakítás letiltását nem veszi figyelembe.)

20.8. CÍMZÉSMÓDOK

A címzésmódtól függően egy utasítás egy, kettő, vagy három byte-ból áll. Az egybyte-os utasítások természetesen csak magát az utasításbyte-ot tartalmazzák, a kétbyte-os utasítások az utasításbyte-ot és egy címbyte-ot (a nulláslapról) vagy egy operandust. Végül a hárombyte-os utasítások az utasításbyte-ból és két címbyte-ból (egy tárcím alsó és felső byte-jából) állnak.

Nem használható mindegyik utasítás mindegyik címzésmóddal. Az M függelékben áttekintést adunk a lehetséges utasításokról és a velük használható címzésmódokról.

Ismerkedjünk meg most a különböző címzési módokkal. Az egyes utasításokat itt is az assembler szintaxisuknak megfelelően jelöljük, azaz mindegyik utasítást három betűvel írunk le. A fejezetben bemutatásra kerülő példákat ki is próbálhatjuk a Plus/4-esen. Indítsuk el a monitorprogramot. Írjuk be a D utasítást, majd ezután a példában megadott tárcímet. A RETURN után a képernyőn megjelenik egy assembler programrészlet néhány sora. Az első programsor a példában tárgyalt utasítássor. Ennek az assembler szintaxisa a címzési módtól függ.

Közvetlen címzés

Kétbyte-os címzés. A második byte mindig egy adatbyte.

Példa:

.A02D A9 80 LDA #$80

Az utasítás az akkumulátorba a $80 értéket tölti.

Lehetséges utasítások: ADC AND CMP CPX CPY EOR LDA LDX LDY ORA SBC

Abszolút címzés

Hárombyte-os címzés. Az első byte az utasításbyte, amelyet egy kétbyte-os cím követ.

Példa:

.A035 20 66 A0 JSR $A066

Az utasítás hatására a program futása a $A066 címen kezdődő alprogrammal folytatódik (programelágazás).

Lehetséges utasítások: ADC AND ASL BIT CMP CPX CPY DEC EOR INC JMP JSR LDA LDX LDY LSR ORA ROL ROR SBC STA STX STY

Nulláslapos címzés

Kétbyte-os címzés. A második byte a nulláslapon levő címet adja meg.

Példa:

.A02F 85 61 STA $61

Az utasítás hatására az akku tartalma betöltődik a $0061 tárcímre.

Lehetséges utasítások: ADC AND ASL BIT CMP CPX CPY DEC EOR INC LDA LDX LDY LSR ORA ROL ROR SBC STA STX STY

Akkumulátoros címzés

Csak egybyte-os, tehát magát az utasításbyte-ot jelenti. Az utasítások csak az akkumulátor tartalmát változtatják meg.

Példa:

.A0AE 4A LSR

Ez az utasítás az akkumulátor tartalmát egy hellyel jobbra tolja. A 7. bit értéke 0, az átvitelbit értéke pedig a 0. bit értéke lesz.

Lehetséges utasítások: ASL LSR ROL ROR

Beleértett címzés

Szintén egybyte-os címzésmód. Ide tartoznak például a CPU-n belüli adatmozgató utasítások, valamint a veremtárra vonatkozó utasítások.

Példa:

.A0B1 A8 TAY

Ez az utasítás az akkumulátor tartalmát átmásolja az Y regiszterbe.

Lehetséges utasítások: BRK CLC CLD CLI CLV DEX DEY INX INY NOP PHA PHP PLA PLP RTI RTS SEC SED SEI TAY TSX TXA TXS TYA

Indexelt indirekt címzés (előzetes indexelés)

Ez már érdekesebb címzésmód. A címzés kétbyte-os, ahol a második byte egy nulláslapos cím. A keresett cím kiszámításának menete: az X regiszter tartalma hozzáadódik a megadott nulláslapos címhez. Az így kapott címen található a keresett cím alsó byte-ja, az 1-gyel nagyobb címen pedig a felső byte-ja. Így megkaptuk azt a tárcímet, amelyre az utasítás vonatkozik. Erre a meglehetősen bonyolult címzésmódra a Plus/4-es BASIC-je és az operációs rendszere nem tartalmaz példát. Találjunk ki hát egyet magunk:

81 A0 STA ($A0,X)

Az akku tartalma abba a rekeszbe íródik be, amelynek a címe abban a rekeszben található, amelynek a címét a $00A0 cím és az X regiszter tartalmának az összege adja meg. Ugye érthető?

Lehetséges utasítások: ADC AND CMP EOR LDA ORA SBC STA

Indirekt indexelt címzés (utólagos indexelés)

Szintén kétbyte-os címzésmód. Ennél a megadott nulláslapos cím tartalmához hozzáadódik az Y regiszter tartalma. Az eredmény adja a keresett cím alsó byte-ját. A felső byte-nak az utasításban megadott cím után következő címen kell lennie. Ha az összeadáskor átvitel keletkezik, akkor a felső byte-hoz 1-et hozzá kell adni.

Példa:

.A0E2 B1 22 LDA ($22),Y

A cím számításának menete: először a $0022 cím tartalmához hozzáadódik az Y regiszter tartalma. Az eredmény a keresett cím alsó byte-ját adja. A $0023 tárcím tartalma plusz egy esetleges átvitel adja a keresett cím felső byte-ját.

Lehetséges utasítások: ADC AND CMP EOR LDA ORA SBC STA

Nulláslapos címzés az X regiszterrel indexelve

Szintén kétbyte-os címzésmód. A megadott nulláslapos címhez hozzáadódik az X regiszter tartalma. A keresett cím felső byte-ja mindig 0.

Példa:

.A1C9 95 29 STA $29,X

Az akku tartalma abba a rekeszbe kerül, amelynek a címét a $0029 és az X regiszter összege adja meg.

Lehetséges utasítások: ADC AND ASL CMP DEC EOR INC LDA LDY LSR ORA ROL ROR SBC STA STY

Abszolút címzés az X vagy az Y regiszterrel indexelve

Ez ismét egy hárombyte-os címzésmód. A megadott címhez hozzáadódik az X vagy az Y regiszter tartalma, és az így kapott összeg adja meg a keresett címet.

Példák:

.9033 9D 00 02 STA $0200,X

Az akku tartalma abba a rekeszbe kerül, amelynek a címét a $0200 és az X regiszter tartalmának az összege adja meg.

.937F D9 53 84 CMP $8453,Y

Az utasítás a $8453 és az Y regiszter tartalmának összegéből adódó címen levő értéket összehasonlítja az akku tartalmával (az utasítás az N,Z és C jelzőbiteket állítja).

Lehetséges utasítások az abszolút-X esetén: ADC AND ASL CMP DEC EOR INC LDA LDY LSR ORA ROL ROR SBC STA

Lehetséges utasítások az abszolút-Y esetén: ADC AND CMP EOR LDA LDX ORA SBC STA

Relatív címzés

Eljutottunk az elágazó utasításokhoz. Ezeknél a relatív címzésmód használatos. Ezek kétbyte-os utasítások, amelyekkel a programszámláló állását változtatjuk. A program végrehajtása alapvetően az állapotregiszter állapotától függően ágazik el. Ez azt jelenti, hogy ha egy feltétel teljesül (hasonlóan a BASIC IF...THEN elágazásához), akkor az utasításban megadott cím hozzáadódik a programszámlálóban levő aktuális címhez. Az utasításban megadott cím úgynevezett 2-es komplemensként van ábrázolva. Tehát a 0–$7F közötti relatív cím a programban 0–127 címmel való előre ugrást, míg a $80–$FF közötti relatív cím visszafelé ugrást jelent. A $FF 1 lépésnyi, a $80 pedig 128 lépésnyi visszafelé ugrást jelent.

Ennek az egésznek egyszerűen az az oka: ahhoz, hogy a programban egyáltalán visszafelé is lehessen ugrani, negatív relatív címet is meg kell tudni adni. A relatív címbyte-unk ezért 7 címbitből és egy előjelbitből áll. A byte-ok ilyen, előjeles értelmezését nevezzük 2-es komplemensnek. Képezzünk egy ilyen számot.

Állítsuk elő a 100 2-es komplemensét. Ehhez először a számot fel kell írni bináris alakban, majd mindegyik bitet az ellenkező értékére kell változtatni (invertálás), és az így kapott eredményhez 1-et hozzá kell adni:

a számunk	=	100	$64	%01100100
negálva	=	155	$9B	%10011011
+1	=	1	$01	%00000001
eredmény	=	156	$9C	%10011100

Abban az elágazó utasításban tehát, amelynek hatására a programban 100 lépést kell visszafelé ugrani, a $9C relatív címet kell megadni.

Példák:

.9004 D0 03 BNE $9009

Ha a zéró jelzőbit 0, akkor a program három lépést előre ugrik. Ennek az utasításnak a feldolgozásakor a programszámláló már a $9006-os címre mutat!

.9021 30 D7 BMI $8FFA

Ez a program 41 lépést ugrik visszafelé, ha a negatív jelzőbit értéke 1. Vegyük figyelembe, hogy a programszámláló a $9023-ra mutat!

Lehetséges utasítások: BCC BCS BEQ BMI BNE BPL BVC BVS

Abszolút indirekt címzés

Ez a címzésmód csak egy utasításnál, az ugró utasításnál használható. Az utasítás hatására a program végrehajtása feltétel nélkül arra a címre ugrik, amelynek az alsó byte-ját az utasításban megadott cím, felső byte-ját pedig a rákövetkező cím tartalmazza.

Példa:

.8683 6C 00 03 JMP ($0300)

A $0300 címen a keresett cím alsó byte-ja, a $0301 címen a felső byte-ja van.

Lehetséges utasítás: JMP

Nulláslapos címzés az Y regiszterrel indexelve

Ez egy kétbyte-os címzésmód. A megadott nulláslapos címbyte-hoz hozzáadódik az Y regiszter tartalma. Az így kapott cím a keresett cím alsó byte-ja. A felső byte mindig $00.

Példa:

.97FF B6 A0 LDX $A0,Y

Az X regiszterbe annak a rekesznek a tartalma kerül, amelynek a címét az $A0 cím és az Y regiszter tartalmának az összege adja. Ez a rekesz mindig a nulláslapon van, mert a felső byte értéke mindig $00.

Lehetséges utasítások: LDX STX

20.9. A GÉPI KÓDÚ PROGRAMOZÁS

Az eddigiekben megismerkedtünk a 7501-es (6502) mikroprocesszor utasításaival, és ezzel tulajdonképpen már fel is készültünk arra, hogy saját magunk írjunk egyszerűbb programokat.

Vigyázat! A gépi kódú programok beírása igen nagy figyelmet követel. A legkisebb

hiba is elegendő ahhoz, hogy a számítógép „fejre álljon", vagyis hogy a mikroprocesszor ellenőrizhetetlen állapotba kerüljön. Igen könnyen előfordulhat, hogy a rendszer által használt fontos címekre hibás értékek íródnak be. Ekkor gyakran csak a RESET gomb benyomása segít. A legtöbb esetben azonban használható még egy trükk: nyomjuk le a RUN/STOP billentyűt, tartsuk lenyomva, és eközben nyomjuk be, majd engedjük el a RESET gombot. Ha most bejelentkezik a monitorprogram ismert alakja, akkor elengedhetjük a RUN/STOP billentyűt is. Most ismét a monitorprogramban vagyunk. Vannak azonban olyan helyzetek, amikor ez az eljárás sem segít.

A legjobb megoldás az, ha indítás előtt minden programot háttértárolóra kiviszünk. A gépi kódú program ugyanis a BASIC-től eltérően hiba esetén nem küld üzenetet.

Kezdjük el egy egyszerű programpéldával: adjunk össze két számot. A számaink legyenek $A0 és $6E. Az eredményt írjuk a $00D0 tárcímű rekeszbe.

```
.1000 A9 A0     LDA #$A0     ;akkuba $A0 erteket toltunk
.1002 18        CLC          ;C bitet toroljuk
.1003 69 6E     ADC #$6E     ;akkuhoz $6E-t es a C bitet
                              hozzaadjuk
.1005 85 D0     STA $D0      ;eredmenyt a $D0 cimre toltjuk
.1007 00        BRK          ;visszateres a monitorprogramba
```

A fenti programban látható az assembler szintaxis egy különlegessége, a megjegyzések beírása. Ezek a megjegyzések a program érthetőségét könnyítik meg. Hozzá kell tenni, hogy a gépbe beépített monitorprogram ilyen megjegyzések beírását nem teszi lehetővé. Ehhez ennél nagyobb teljesítményű assemblerre lenne szükség, amelyek általában még további szolgáltatásokat is nyújtanak (direktívák, szimbólumok használata stb.). Mivel a Plus/4-es csak egy úgynevezett direkt vagy azonnali assemblert tartalmaz, az előző (és a könyv további részében levő összes) programot a következőképpen kell beírni:

.1000 LDA#$A0 <RETURN billentyű>
CLC <RETURN billentyű>
ADC $6E <RETURN billentyű>
STA $D0 <RETURN billentyű>
BRK <RETURN billentyű>
<RETURN billentyű>

Ha valaki ezt túlságosan hosszúnak találja, akkor ezt is megteheti (soronként max. 8 byte írható be):

>1000 A9 A0 18 69 6E 85 D0 00 <RETURN billentyű>

Ezzel a program máris a tárban van. A program indítása:

G 1000 <RETURN billentyű>

Ennyit a programbeírásról. A következőkben a programok írásához alapvetően az assembler szintaxist használjuk. Úgy gondoljuk, hogy RETURN billentyű lenyomására sem kell már a figyelmet felhívni. Térjünk vissza a programunkhoz.

A program elindítása és lefutása után az eredmény $0E, ami a $00D0 tárcímű rekeszbe kerül. Ez az eredmény van egyébként még az akkuban is, amint ez a monitorprogram bejelentkezésekor látható is: a regiszterek kijelzésénél az AC alatt a $0E érték áll. Jogos viszont az az észrevétel, hogy ez az eredmény hamis, hiszen $A0

és $6E összege végül is $10E. Ez igaz, de mivel az akkuba csak egy 8 bites érték „fér el", ott az eredménynek csak az alsó 8 bitje, tehát az alsó byte-ja állhat. (a 9. bit a carryben van). Ahhoz, hogy a teljes eredményt megkapjuk, egy kicsit változtatni kell a programunkon. Álljon az eredmény a $D0 és a $D1 tárcímeken.

```
.1000 D8        CLD
.1001 A9 00     LDA #$00
.1003 85 D1     STA $D1
.1005 18        CLC
.1006 A9 A0     LDA #$A0
.1008 69 6E     ADC #$6E
.100A 90 02     BCC $100E
.100C E6 D1     INC $D1
.100E 85 D0     STA $D0
.1010 00        BRK
```

A programban először is töröltük a decimális jelzőbitet. Erre mindenképpen szükség van, mert most bináris számrendszerben számolunk. A programunk csak így működik helyesen. Most ugyanis az ADC utasítás végrehajtásakor a C bit értéke az eredménytől függően állítódik be. Emlékezzünk vissza: a C átvitelbit étéke akkor lesz 1, ha az összeadáskor az akku 7. bitjén átvitel keletkezik. Most éppen ez az eset áll fenn, mivel az eredmény nagyobb, mint $FF.

Ha a C átvitelbit értéke nem 1 lenne, akkor ugorjon a CPU a $100E címre (BCC $100E). Ha viszont a C bit magas, akkor legyen a $D0 tárcímen levő rekesz tartalma 1-gyel nagyobb (inkrementálódjon). Most a $D0 és a $D1 tárcímeken a helyes eredmény, nevezetesen a $010E áll.

Írjunk most egy olyan programot, amely lehetővé teszi, hogy a billentyűzetről meghatározott számú karaktert írhassunk a gépbe. Ha a megadott számot elértük, akkor a karakterek jelenjenek meg a képernyőn.

Ehhez néhány alapvető dologról kell említést tennünk: a Plus/4-esnek 64 kbyte ROM tára van. Ez fixen beírt, a felhasználók által megváltoztathatatlan programokat tartalmaz. Ebből a ROM-ból 32 kbyte területet a BASIC interpreter és az operációs rendszer foglal el. Ezek rengeteg, rövidebb-hosszabb rutint tartalmaznak, amelyeket a programozók is felhasználhatnak a saját programjuk részeiként. Miért kellene tehát saját magunknak egy billentyűzetlekérdező rutint kifejlesztenünk, ha ilyen már van a ROM-ban? Ugyanez a helyzet, ha pl. a képernyőre vagy valamilyen más, külső egységre egy jelet akarunk küldeni.

Ahhoz, hogy ezeket a rutinokat használni tudjuk, elegendő, ha ismerjük a belépési (kezdő) címüket, és gondoskodunk arról, hogy az általunk támasztott követelmények teljesüljenek (pl. a kívánt adat a kívánt regiszterben legyen). Ebben és a következő fejezetekben bemutatjuk az operációs rendszer legfontosabb rutinjait, a belépési címükkel együtt. Térjünk most vissza a beviteli/kiviteli programunkhoz.

20.9.1. Példaprogram: A billentyűzet lekérdezése

Az itt következő programmal öt karaktert olvasunk be a billentyűzetről, majd a karaktereket kiírjuk a képernyőre.

```
.1000 A2 05     LDX #$05    ;szamlalo
.1002 20 CF FF  JSR $FFCF   ;BASIN, karakter bevitele
.1005 95 D0     STA $D0,X   ;karakter tarolasa
.1007 CA        DEX         ;szamlalo csokkentese 1-gyel
.1008 D0 F8     BNE $1002   ;szamlalo meg nem zerus
.100A A2 05     LDX #$05    ;szamlalo a kiirashoz
.100C B5 D0     LDA $D0,X   ;karakter az akkuba
.100E 20 D2 FF  JSR $FFD2   ;BSOUT, egy karakter kiirasa
.1011 CA        DEX         ;szamlalo csokkentese 1-gyel
.1012 D0 F8     BNE $100C   ;szamlalo meg nem zerus
.1014 00        BRK         ;vissza a TEDMON-hoz
```

Ebben a programban az operációs rendszer két rutinját használjuk: a beviteli rutint (BASIN $FFCF) és a kiíró rutint (BSOUT $FFD2). Az első rutin egy karaktert betölt az akkuba, a második pedig az akkuban levő karaktert a képernyőre írja.

Térjünk vissza a programunkhoz. Ezzel a programmal ötnél több karaktert is beírhatunk, de ezek közül csak az első ötöt tárolja. Emiatt aztán a képernyőn nagy lehet a zűrzavar.

A billentyűzet lekérdezéséhez tehát más módszert kell választanunk. Ehhez gondoljuk végig, hogy milyen folyamatok zajlanak le a számítógépben. A számítógép ugyanis a megszakítás programmal folyamatosan figyeli a billentyűzetet. Ha lenyomunk egy billentyűt, akkor annak a kódját a $C6 című rekeszben tárolja. A (utolsó RETURN-től számított) lenyomott billentyűk száma az $EF címre kerül, míg maguk a billentyűk kódjai az úgynevezett billentyűzet pufferbe ($0527–$0530) íródnak. A lenyomott billentyűk számának megállapításához tehát a $EF tárcím tartalmát kell lekérdezni. Ha az értéke eléri az 5-öt, akkor az első öt karaktert egyszerűen kiolvassuk a billentyűzet pufferből. A programunk ekkor így néz ki:

```
.1000 A2 05     LDX #$05    mar az otodik lenyomott gomb?
.1002 E4 EF     CPX $EF
.1004 D0 FC     BNE $1002   ;nem
.1006 A2 00     LDX #$00    ;igen, a karakter kiirasa
.1008 BD 27 05  LDA $0527,X
.100B 20 D2 FF  JSR $FFD2   ;BSOUT
.100E E8        INX
.100F E0 05     CPX #$05    ;mar az otodik kiirt karakter?
.1011 D0 F5     BNE $1008   ;nem
.1013 00        BRK
```

20.9.2. Példaprogram: Grafikus minta

Rajzoljunk a képernyőre egy mintát a finomfelbontású grafikus üzemmódban. Írjunk ehhez egy olyan programot, amelyet BASIC-ből SYS utasítással indíthatunk. Azért, hogy különböző mintákat lehessen kirajzolni, a SYS utasítás után megadunk egy tetszőleges számot. Lássuk először a programot. A program kezdőcíme $065E.

A $065E–$06EB közötti területre szabadon írható gépi kódú program anélkül, hogy ez a BASIC területen levő programot megzavarná. A példaprogramunkban a nullás-lapon levő tárcímekre is szükségünk van. Erről a lapról a $D0–$E8 közötti címek használhatók.

```
.065E 20 DE 9D    JSR $9DDE     ;atvesz egy erteket BASIC-bol
                                ;(max. 65535)
.0661 A0 00       LDY #$00
.0663 A2 20       LDX #$20
.0665 84 D0       STY $D0       ;grafikus tar kezdocim also
                                 byte-ja
.0667 86 D1       STX #D1       ;grafikus tar kezdocim felso
                                 byte-ja
.0669 B1 D0       LDA ($D0),Y ;grafikus tar es a $14
                                 tartalma kozott
.066B 45 14       EOR $14       ;EOR muvelet elvegzese
.066D 91 D0       STA ($D0),Y ;az eredmeny visszairasa
.066F A5 14       LDA $14       ;$14 rekesz tartalmanak
.0671 49 FF       EOR $FF       ;negalasa
.0673 85 14       STA $14
.0675 E6 D0       INC $D0       ;$D0 tartalma plusz 1
.0677 D0 F0       BNE $0669     ;elagazas, ha nem #$00
.0679 E6 D1       INC $D1       ;$D1 tartalma plusz 1
.067B A6 D1       LDX $D1       ;$D1 tartalmanak osszE-
.067D E0 40       CPX #$40      ;hasonlitasa #$40-nel
.067F D0 E8       BNE $0669     ;ha kisebb, elagazas $0669-re
.0681 60          RTS           ;vissza a BASIC-be
```

Ebben a programban is használunk egy érdekes rendszerrutint, nevezetesen a GETADR nevű rutint (belépési címe $9DDE). Ez a rutin átvesz a BASIC programból egy max. 65535 értéket, és ezt elhelyezi a $14 (alsó byte) és a $15 (felső byte) címeken. A programunkban most csak az alsó byte-ot használjuk. Ezt a byte-ot írjuk be a grafikus tárba. Előzőleg azonban a byte és a grafikus tár minden egyes byte-ja között elvégezzük a KIZÁRÓ VAGY (EOR) műveletet, hogy a grafikus tár esetleges tartalmát ne változtassuk meg. A grafikus tár byte-jainak címe a $D0 és $D1 címeken van. Ezt a címet a grafikus tár utolsó címéig, $3FFF-ig növeljük. Ezután a program visszatér a BASIC-be. Azért, hogy a minta egy kicsit érdekesebb legyen, a $14 címen levő érték és a #$FF között elvégzünk egy KIZÁRÓ VAGY műveletet, aminek következtében az egyes bitek értéke az ellenkezőre változik (negálás). A gépi kódú program indítása BASIC-ből:

SYS 1630,X

ahol X egy 0–255 közötti tetszőleges szám.

Példa:

```
10 GRAPHIC1,1
20 CIRCLE,100,100,100
30 FOR X=0 TO 255
40 SYS 1630,X
50 GETKEY A$
60 NEXT
```

Ha a grafikus képernyőt kétszer egymás után ugyanazzal az értékkel állítjuk elő, akkor visszakapjuk az eredeti képernyőt. A gépi kódú programmal természetesen még tovább lehet kísérletezgetni. A $14 címen levő értéket pl. nem kell minden alkalommal az ellenkezőjére változtatni. Ez egyszerűen kiiktatható, ha a $0671 és a $0672 címekre egy-egy NOP utasítást írunk be. Az ábránk így egy kicsit plasztikusabb lesz. A $0668 címre ORA $14 vagy AND $14 utasítást is írhatunk. Kísérletezgessünk egy kicsit. Ez a legjobb módszer a gépi kódú programozás megtanulásához.

20.9.3. Példaprogram: Grafika áttárolása

Tároljuk a grafikus képernyőt a RAM-ban, a $DD00 és a $FCFF tárcímek közötti területen. Ezzel egyidejűleg az ott levő tártartalmat töltsük át a grafikus tárba (tehát a $2000–$3FFF címek közötti területre). A feladatunk tehát az, hogy két tárterület tartalmát egymással felcseréljük.

```
.065E A9 40     LDA #$40    ;1.tar vegcim + 1 felso
                             byte-ja
.0660 85 D4     STA $D4
.0662 A0 00     LDY #$00    ;1. es 2.tar kezdocim also
                             byte-ja
.0664 A2 20     LDX #$20    ;1.tar kezdocim felso byte-ja
.0666 84 D0     STY $D0
.0668 86 D1     STX $D1
.066A A2 D0     LDX #$D0    ;2.tar kezdocim felso byte
.066C 84 D2     STY $D2
.066E 86 D3     STX $D3
.0670 78        SEI
.0671 8D 3F FF  STA $FF3F   ;a RAM bekapcsolasa
.0674 B1 D2     LDA ($D2),Y ;2.tarbol egy byte olvasasa
.0676 48        PHA         ;es a verembe helyezese
.0677 B1 D0     LDA ($D0),Y ;1.tarbol egy byte olvasasa
.0679 91 D2     STA ($D2),Y ;es beirasa a 2.tarba
.067B 68        PLA         ;byte elovetele a verembol
.067C 91 D0     STA ($D0),Y ;es beirasa az 1.tarba
.067E C8        INY         ;Y regiszter + 1
.067F D0 F3     BNE $0674   ;nincs atvitel
.0681 E6 D1     INC $D1     ;ha van atvitel, akkor
                             kezdocim + 1
.0683 E6 D3     INC $D3
.0685 A5 D1     LDA $D1     ;grafikus tar vege?
.0687 C5 D4     CMP $D4
.0689 D0 E9     BNE $0674   ;meg nem
.068B 8D 3E FF  STA $FF3E   ;igen, ROM bekapcsolasa
.068E 58        CLI
.068F 60        RTS         ;vissza a BASIC-hez
```

Ebben a programban magasra állítottuk a megszakítás (Interrupt) jelzőbitet. Erre feltétlenül szükség van, mert az operációs rendszer ROM-ot ki kell kapcsolnunk. Emlékezzünk vissza a Plus/4-es tárterületeinek felosztására: a BASIC és az operációs rendszer ROM alatt helyezkedik el a RAM-tár. Ehhez a RAM-hoz a $FF3F tárcímen keresztül fordulhatunk. Most viszont nem következhet be megszakítás, mert azt a ROM-ot, ahol a megszakítás rutin található, kikapcsoltuk. A megszakítások kezelésével a 21. fejezetben foglalkozunk.

A programunkban ezúttal két mutatót használunk: az egyik a $DO $D1 címen, a másik a $D2 $D3 címen van. A $D4 címen a grafikus tár végcíménél 1-gyel nagyobb cím felső byte-ja van. Először a céltartományból ($DD00-tól kezdődően) betöltünk egy byte-ot az akkumulátorba, majd ezt az értéket beírjuk a veremtárba (PHA utasítás). Ezután a grafikus tár megfelelő címéről töltünk az akkuba egy értéket, majd ezt beírjuk a céltartomány megfelelő helyére. Végül a veremből elővesszük az oda eltett értéket (PLA), és beírjuk a grafikus tárba. Rendre végrehajtva ezeket a cseréket a $2000–$3FFF és a $DD00–$FCFF címek közötti tárterületek tartalma egymással felcserélődik.

Mielőtt a programot elindítanánk, feltétlenül gondoskodnunk kell arról, hogy a $DD00-tól kezdődő tárterület a BASIC elől el legyen zárva. Ezt még a BASIC program beírása előtt el kell végezni. Írjuk be:

```
POKE 56,221:CLR
```

Ezután még mindig kereken 52500 byte-nyi terület áll a programjaink rendelkezésére, és a SYS 1630 utasítással két grafikus képernyő között tudunk ide-oda kapcsolni.

20.9.4. Példaprogram: OLD rutin

Bizonyára mindenkivel előfordult már, hogy a NEW parancsal kitörölt a gépből egy olyan programot, amelyre még szüksége lett volna. Netán előzőleg nem is tárolta kazettán vagy lemezen.

Az is lehet, hogy valaki úgy fejezte be a programot, hogy benyomta a RESET gombot, de közben nem tartotta lenyomva a RUN/STOP billentyűt. Ekkor pedig a programnak lőttek!

Csak semmi pánik! Szigorúan véve ezzel ugyanis még nem töröltük a programot, csupán csak a BASIC tár első három byte-ját állítottuk nullára. Maga a teljes program még a tárban van, csak éppen nem listázható és nem indítható. Nosza rajta, mentsük meg a programot! Erre két lehetőséget mutatunk be.

Az egyszerűbb megoldás:

Írjuk be közvetlen üzemmódban a következőket:

```
POKE 4097,1:RENUMBER
```

A programunk most megint teljes egészében a tárban van. Arra kell csak ügyelni, hogy a fenti utasítás beírása előtt a BASIC tár kezdete ugyanott van, ahol a RESET gomb benyomása előtt is volt. Ha pl. korábban bekapcsoltuk a grafikát, akkor azt most még egyszer be, majd ki kell kapcsolni. A fenti parancsot csak ezután szabad kiadni.

Lássuk most a másik módszert, amellyel egy törölt programot vissza lehet hozni. Azt is elmagyarázzuk, hogy mi ennek a módszernek a mechanizmusa. Írjuk be egy RESET vagy egy NEW után a következőt:

```
POKE PEEK(43)+PEEK(44)*256,1:SYS34840:SYS34891:CLR
```

Most a program ismét teljes egészében a tárban van. Listázható, futtatható és tárolható. Ha viszont a program a próbafutása során saját magát tette tönkre (valamilyen POKE-okkal), akkor természetesen most is a hibás változat van a tárban. Ezt az esetet azonban – amely minden programozóra a hideglelést hozza – ezúttal hagyjuk figyelmen kívül.

Ha valaki ezt a parancssort nem akarja minden alkalommal újra beírni, akkor gépi kódú rutinként kazettán vagy lemezen is tárolhatja, ahonnan szükség esetén visszatöltheti, és egy SYS-szel elindíthatja.

Ezt a rutint OLD-rutinnak neveztük el. Vannak ugyanis olyan BASIC interpreterek, amelyek ismerik az OLD utasítást, amellyel egy „törölt" programot vissza lehet hozni. A programunkat ismét a $065E tárcímtől kezdve írjuk meg:

```
.065E A0 00     LDY #$00     ;csatolocim szimulalasa
.0660 A9 01     LDA #$01     ;es a
.0662 91 2B     STA ($2B),Y  ;BASIC tarba irasa
.0664 20 18 88  JSR $8818    ;csatolocimek ujraszamitasa
.0667 20 4B 88  JSR $884B    ;valtozoterulet kezdet
                              ujraszamitasa
.066A 4C 9A 8A  JMP $8A9A    ;BASIC CLR utasitas
```

Mit csinál ez az OLD-rutin? Képzeljük el, hogy a tárban van egy BASIC programunk, és megváltoztatjuk ennek az első sorát. Ekkor a számítógép az összes csatolócímet újra számítja. Pontosan ezt tesszük itt mi is. Elhitetjük a számítógéppel, hogy változás történt a BASIC programban, és a $8818-as címen kezdődő rutinnal újra számítjuk az összes csatolócímet. Ez mindaddig tart, míg a tárban két, egymást követő 00 byte-hoz nem érünk. Ez jelenti ugyanis a program végét.

Most már bizonyára érthető lesz, hogy miként működik a rutinunk. A NEW utasítás a BASIC tár elején, tehát az $1001 és $1002 címeken levő rekeszek tartalmát nullázza. Egy LIST vagy RUN kiadásakor a gép tehát két, egymást követő 00 byte-ot talál, és ezt a BASIC program végeként értelmezi. Most a rutinunkkal egy fiktív csatolócímet „szimulálunk" úgy, hogy az $1001-es címre – tehát amelyikre a $2B $2C mutat – beírunk #$01-et. Ezzel azt mondjuk a számítógépnek, hogy itt még nincs vége a programnak, tehát számolja újra a további csatolócímeket mindaddig, amíg a tárban két, egymást követő 00 byte-ot nem talál. (Szigorúan véve három darab 00 byte-ot, melyek közül az első a programsor végét, a második kettő pedig magának a programnak a végét jelenti.)

A csatolócímek újraszámítása után még ki kell számítani azt a mutatót, amely a változók területének a kezdőcímére mutat, majd meg kell hívni a CLR rutint, amely a többi mutató értékét is helyreállítja.

Írjuk most a gépi kódú programunkat a monitorral "OLD SYS 1630" néven valamilyen háttértárolóra, ahonnan a

LOAD"OLD SYS 1630",8,1 utasítással (lemezről) vagy
LOAD"OLD SYS 1630",1,1 utasítással (kazettáról)

bármikor visszatölthetjük a gépbe.

A program indítása:

SYS 1630

Ha egy BASIC program törlése előtt a finomfelbontású grafika is be volt kapcsolva, és így nyomtuk be a RESET gombot, akkor is az OLD rutin meghívása előtt a grafikát ismételten be vagy ki kell kapcsolni:

GRAPHIC 1:GRAPHIC 0

Ennek hatására a BASIC-tár kezdete visszaáll a kiinduló értékére (ha be van kapcsol-

va a grafika, akkor a BASIC programtár nem a $1000-es, hanem a $4000-es címen kezdődik).

Ennyit bevezetésként a gépi kódú programozásba. Ugye nem is annyira bonyolult, mint amennyire az első pillantásra látszik.

A következő fejezetben is a gépi kódú programozással foglalkozunk, de most már egy kicsit mélyebben. Ezért a fejezetnek *A Plus/4 gépi kódú programozása haladóknak* címet adtuk. Bemutatjuk a megszakítások programozását, aztán az úgynevezett memórialapozást. Látunk majd néhány programpéldát, és megismerkedünk az operációs rendszer néhány újabb, fontos rutinjával. A saját fejlesztésű gépi kódú programok írásához feltétlenül javasoljuk a függelékben összefoglalt utasítások tanulmányozását.

21. A Plus/4-es gépi kódú programozása haladóknak

Ebben a fejezetben még jobban elmélyedünk a számítógép működésének rejtelmeiben. Az a szándékunk, hogy bemutassuk a rendszer programozásának további lehetőségeit. Többek között megismerkedünk a Plus/4-es tárkezelésével is. Ez a későbbi programok megértéséhez nélkülözhetetlen. Az ehhez szükséges alapismereteket igyekszünk a lehető legegyszerűbb módon elmagyarázni. Lássunk hát neki.

21.1. A MEGSZAKÍTÁSOK PROGRAMOZÁSA

Kétségkívül az egyik legérdekesebb programozási technika a megszakítások programozása. Ebben a fejezetben megtanuljuk azt, hogy tulajdonképpen mi is az a megszakítás, hogy hogyan jön létre, mi a hatása, milyen megszakítási lehetőségek vannak a Plus/4-esnél és végül, de nem utolsósorban, hogy miként lehet ezeket programozni. (Megjegyzés: a számítástechnikai gyakorlatban sűrűn használatos a megszakítás eredeti, angol nyelvű megfelelője, az interrupt).

21.1.1. A megszakítások

A számítógép operációs rendszere a gépben futó programot folyamatosan és rendszeres időközönként megszakítja. Amikor bekapcsoljuk a számítógépet, a képernyőn megjelenik egy szöveg, és villog a kurzor. A gép ezzel jelzi, hogy egy utasítás vagy valamilyen program beírására vár. Ez a várakozás azonban nem „semmittevés", hanem az operációs rendszer által folyamatosan végrehajtott várakozási ciklus. A gép processzora tehát a látszat ellenére állandóan dolgozik.

Erről nagyon egyszerűen meggyőződhetünk, hiszen ha lenyomunk egy billentyűt, akkor a neki megfelelő karakter azonnal megjelenik a képernyőn. Aki viszont azt gondolja, hogy ezt a műveletet a várakozási ciklus hajtja végre, az bizony téves nyomon jár.

A várakozási ciklus másodpercenként kb. 100-szor szakad meg. Egy megszakítás közben a processzor az operációs rendszer egy másik részén végzi a munkáját. Ezen a helyen figyeli – többek között – a billentyűzetet is. Ha azt érzékeli, hogy egy ilyen megszakítás idején valamelyik billentyű éppen le van nyomva, akkor megkeresi az ehhez tartozó kódot. Ez azonban csak egyike annak a számos feladatnak, amelyeket a processzornak a megszakítás ideje alatt el kell végeznie. Ugyanez történik akkor is, amikor a számítógép egy – akár BASIC, akár gépi kódú – programot hajt végre. A megszakítások mindig pontosan, és ami még fontosabb, folyamatosan hajtódnak végre.

Számunkra ez teszi különösen érdekessé a megszakítások programozását. Ha sikerül megoldanunk, hogy a megszakítási fázisban a saját programjaink (is) fussanak, akkor sok dolgot egyszerűbben is programozhatunk. A gépi kódú programok futásának gyorsasága következtében olyan hatást érhetünk el, mintha a gépen egyidejűleg két program futna. A fejezet végén bemutatunk egy példaprogramot, amely egy billentyű lenyomásakor hangjelet is kivált.

Úgy gondoljuk, hogy ezzel a magyarázattal választ adtunk az első kérdésre, nevezetesen arra, hogy mi is az a megszakítás. Jöhet hát a következő kérdés: vajon honnan tudja a processzor, hogy éppen egy másik programot kell végrehajtania?

21.1.2. A megszakítások előállítása

A megszakítások a keletkezésük szempontjából alapvetően két csoportba oszthatók: vannak a hardver által kiválasztott, vagy hardver oldali megszakítások és vannak szoftver által kiváltott, vagy szoftver oldali megszakítások. Vizsgáljuk meg először a hardver által kiváltott megszakításokat.

Ehhez azonban még néhány, a hardverre vonatkozó ismeretre van szükség:

A Plus/4-es – akárcsak más számítógépek is – nem csak processzorból és a RAM és a ROM áramkörökből állnak, hanem további integrált áramköröket, úgynevezett chipeket is tartalmaznak. E chipek egyike különösen fontos a számítógép kifogástalan működése szempontjából. Ennek a neve TED, amely az angol TExt Display kifejezésből adódik. A számítógépen belül a TED a felelős az olyan feladatok elvégzéséért, mint a tárkezelés, a képernyő felépítése, a billentyűzet és a botkormányok figyelése, továbbá a hardver oldali megszakítás kiváltása. Van néhány olyan, RAM tárcíme is, amelyekhez ugyanúgy lehet fordulni, mint a „közönséges" RAM címekhez, azaz ezekre értékek írhatók, ill. a bennük levő értékek olvashatók. Ezek a címek a $FF00–$FF3F közötti területen vannak.

Egy olyan, sok feladatot ellátó chipnek, mint a TED, számos csatlakozója van. Ezek közül a nyolcas számú csatlakozónak a neve IRQ, ami az angol Interrupt ReQuest rövidítése, és magyarul megszakítás kérést jelent. Ha a Plus/4-es működése közben bizonyos feltételek teljesülnek, akkor ennek következtében megváltozik az ezen a csatlakozón levő feszültség. A későbbiekben majd arra is kitérünk, hogy melyek ezek a feltételek.

A processzorunknak is van egy IRQ csatlakozója, mégpedig a hármas számú. Ez a két csatlakozó egymással össze van kötve, tehát a TED által kiváltott feszültségváltozást a processzor is érzékeli. Ezt a feszültségváltozást megszakítás kérésnek is nevezzük. Van még egy másik olyan chip is, amely megszakítás kérést küldhet a processzornak, ez pedig az Input/Output, azaz a bemeneti/kimeneti egység. Ebben a fejezetben csak a TED-re vonatkozó megszakításokkal foglalkozunk, ezért erre az utóbbi chipre nem térünk ki.

Azért, hogy egy beérkező megszakítás kérésre a processzor megfelelőképpen tudjon reagálni, egy saját, beépített programot tartalmaz. Amikor a processzorhoz egy megszakítás kérés érkezik, akkor a következő lépéseket teszi: Az éppen feldolgozás alatt álló utasítást végrehajtja, majd megvizsgálja az állapotregiszterben levő megszakítás bitet. Ha ez a bit magas, azaz az értéke 1, akkor a programmegszakításra vonatkozó kérést figyelmen kívül hagyja, és megkezdi a következő utasítás feldolgozását. A legtöbb esetben azonban a megszakítás jelzőbit nem magas, tehát engedélyezi a megszakítást. Ekkor a processzor először a programszámláló (angol rövidítése PC)

tartalmát a verembe helyezi. Erre azért van szükség, hogy a processzor a megszakítási rutin végrehajtása után tudja azt, hogy az eredeti, megszakított program végrehajtását melyik címen kell folytatnia. Ahhoz, hogy az eredeti programot a megszakítás előtti feltételek mellett folytathassa, az állapotregiszter tartalmát is a verembe kell mentenie. Ez is, akárcsak a megszakítás bit magasra állítása, automatikusan megy végbe. Most valami különleges történik:

A programszámlálóba betöltődik a $FFFE és a $FFFF címeken levő tartalom. Ezek a címek egyébként azok a legmagasabb címek, amelyeket a processzor még éppen meg tud címezni. Mivel a Plus/4-es különböző memóriaterületeket tud kiválasztani, a következőkben induljunk ki abból, hogy az operációs rendszer tárterülete van bekapcsolva. A fenti címek (amelyek tartalma a Plus/4-esnél $B3 és $FC) egy mutatót tartalmaznak, amely a feldolgozandó megszakítási rutinra mutat. A Plus/4-esnél tehát a program végrehajtása a $FCB3 címre ágazik el, és ott kezdődik el a megszakítási program feldolgozása. Mivel ez a megszakítási kérés egy vezetéken érkezik a processzorhoz, ezt a fajta megszakítást hardver oldali megszakításnak nevezik.

Nézzük most a szoftver oldali megszakítást:

Ezt a fajta megszakítást a processzor BRK utasítása váltja ki. Amikor a processzornak ezt az utasítást kell feldolgoznia, akkor majdnem minden úgy történik, mint az előző esetben. Van azonban két sajátossága: Mivel a BRK utasítást többnyire gépi kódú programok tesztelésére használják, ahol ezt gyakran egy 2 byte-os utasítás helyett kell beírni, a verembe a programszámláló 2-vel megnövelt értéke kerül. A megszakítási rutinból való visszatérés után a program végrehajtása azzal az utasítással folytatódik, amely a BRK utasítás utáni, második címen áll. A program a BRK megszakítás esetén is arra a címre ágazik, amelyikre az $FFFE és az $FFFF címek mutatnak. Azért, hogy ezt a fajta megszakítást a hardver oldali megszakítástól meg lehessen különböztetni, az állapotregiszterben a megszakítás jelzőbit (IRQ) mellett a BREAK bit (B bit) is magasra állítódik.

A mi processzorunk a megszakításokat illetően némileg különbözik a 6502-es típusúaktól. Ez utóbbiaknak ugyanis van még egy NMI vezetékük is. Az NMI az angol Non-Maskable-Interrupt rövidítése, és magyarul nem maszkolható megszakítást jelent, tehát olyan megszakítást, amelyet nem lehet letiltani. Ha a processzorhoz ilyen megszakítás érkezik, akkor az azt minden körülmények között végrehajtja. Ez a lehetőség a Plus/4-es számítógép 7501-es mikroprocesszorába nincs beépítve.

Ha esetleg valaki számára az itt leírtak nem teljesen világosak, az sem nagy baj. A fejezet további részeit e nélkül is meg lehet érteni. A szándékunk csak az volt, hogy némi bepillantást adjunk a hardver működésébe is.

21.1.3. A megszakítások programozása

Ebben a fejezetben kizárólag a hardver oldali megszakításokkal foglalkozunk. Ezek egyúttal a legrugalmasabb megszakítások is. Mielőtt folytatnánk az olvasást, kapcsoljuk be a gépet, és indítsuk el a monitorprogramot. Írjuk be:

```
D FCB3 FCBB
```

A képernyőn most látható rutin a regisztertartalmakat a verembe menti. Először az akkumulátort, aztán az X regisztert, majd az Y regisztert. Az STA $FDD0 utasítás

most a rendszer ROM-ot kapcsolja be. A JMP $CE00 utasítással ágazunk el a tulajdonképpeni megszakítási rutinra. Hajtsuk végre mi is ezt az elágazást:

D CE00 CE0B

Először az AKKU-ba kerül az állapotregiszternek a megszakítási rutinra való elágazás előtti aktuális tartalma, amely most a veremben van. A rutin most e tartalom és a #$10 érték között végrehajt egy AND műveletet (B bit vizsgálata). Ha a megszakítást egy BRK utasítás váltotta ki, akkor az állapotregiszterben levő 4-es bit értéke 1. A logikai művelet eredménye ekkor nullától eltérő, és a program elágazik a $CE0B címre. Ha viszont az eredmény nulla, akkor a program a $CE08 címtől folytatódik. Ezen a címen egy indirekt ugrás van arra a címre, amely a tár $0314 és $0315 címén található.

Ezen a ponton tudunk ehhez a rutinhoz a saját rutinjainkkal csatlakozni. Nézzük meg, mi van ezeken a címeken:

M 0314 0315

Az itt található cím: $CE0E. Mivel a $0314 és a $0315 címek RAM területen vannak, ezek tartalmát könnyedén meg tudjuk változtatni úgy, hogy ezek a saját rutinunkra mutassanak. Bemutatjuk, hogy mindez hogyan történik. Írjuk be a monitorprogrammal az alábbi sorokat:

```
.065E SEI
LDA #$6C
STA $0314
LDA #$06
STA $0315
CLI
RTS
```

Ezekkel az utasításokkal a megszakítás ugrásvektorát a $066C címre „térítettük" el. Ehhez először a megszakítás jelzőbitet az SEI utasítással magasra kell állítani. Emlékezzünk csak vissza, hogy ha ez a bit magas, akkor a megszakítás nincs engedélyezve. A megszakítás letiltása itt nagyon fontos, mert ha az „eltérítés" közben érkezne a processzorhoz egy megszakítás kérés, akkor a program egy teljesen meghatározhatatlan címre ágazna el, és a gép egészen bizonyosan „fejre állna".

Miután a mutatót „eltérítettük", a CLI-vel ismét engedélyezzük a megszakításokat, és egy RTS-sel befejezzük a kis programunkat. Ezzel a néhány programsorral most lehetővé vált, hogy a $066C-től kezdődő saját rutinjainkat bekapcsoljuk egy megszakításba. Mivel azonban ezeken a címeken még semmi értelmes program sincs, indításkor a számítógép is csak valami vad dolgot művelne. Írjuk be ezt:

```
.066C INC $FF19
JMP $CE0E
```

Miután beírtuk ezeket a sorokat is, lépjünk ki az X-szel a monitorból. A programunk indítása: SYS 1630. A képernyő most egy kicsit nyugtalankodni kezd. A programunk ugyanis minden egyes megszakításkor megváltoztatja a képernyő keretének színét. A változást a $FF19 tárcím tartalmának a folyamatos növelése idézi elő. A TED chip-ben ezen a tárcímen a keretszín regisztere van. A JMP $CE0E utasítással újra belépünk a tulajdonképpeni megszakító rutinba. A számítógép minden más funkciója változatlan marad. Legyen még tarkább? Egy RESET után módosítsuk a programot:

```
.066F INC $FF15
JMP $CE0E
```

Egy SYS 1630 után most a háttér is villog. A TED-ben a $FF15 regiszter tartalmazza a háttér színét. A sor beírásával szinte láthatóvá válnak a megszakítások. Azt is megfigyelhetjük, hogy a képernyő alsó ötödének mindig más a színe, mint a felső négyötödének. Ennek az oka a Plus/4-es rasztermegszakítási mechanizmusában keresendő.

21.1.4. A TED megszakító regiszterei

Amint a bevezetőben már említettük, a TED-nek különböző RAM regiszterei vannak, amelyek írhatók is és olvashatók is. A $FF15 és a $FF19 regiszterekről éppen az előbb volt szó, amikor a példánkban ezek tartalmát tetszőlegesen változtattuk. A következőkben azokkal a regiszterekkel foglalkozunk, amelyeknek közvetlen közük van a megszakítások előállításához.

Elsőnek vizsgáljuk meg a megszakítás érzékelő regisztert (Interrupt Request Regiszter). Ebben a regiszterben nyolc bit van, amelyek közül a 0. és az 5. bit kivételével mindegyiknek valamilyen különleges feladata van.

$FF09 megszakítás érzékelő regiszter

Bit	7	6	5	4	3	2	1	0
	megszakítás	3. időzítő	– – –	2. időzítő	1. időzítő	fényceruza	raszter	– – –

Nézzük meg az egyes bitek jelentését és a feladatukat.

0. bit: nincs feladata, az értéke közömbös
1. bit: rasztermegszakítást jelez
2. bit: fényceruza által kiváltott megszakítást jelezne, de mivel a Plus/4-esnek ilyen bemenete nincs, a bit értéke közömbös.
3. bit: a megszakítást az 1. időzítő alulcsordulása váltotta ki.
4. bit: a megszakítást a 2. időzítő alulcsordulása váltotta ki.
5. bit: nincs szerepe.
6. bit: a 3. időzítő alulcsordulása.
7. bit: jelzi, hogy megszakítás történt.

Nézzük mindjárt a következő regisztert, amelyet megszakítás engedélyező regiszternek nevezünk (Interrupt Mask Regiszter). Ez a regiszter is nyolc bitet tartalmaz.

$FF0A megszakítás engedélyező regiszter

Bit	7	6	5	4	3	2	1	0
	– – –	3. időzítő	– – –	2. időzítő	1. időzítő	fényceruza	raszter	9. bit

0. bit: ez a raszterregiszter 9. bitje
1. bit: engedélyezi a rasztermegszakítást
2. bit: engedélyezné a fényceruza általi megszakítást

3. bit: ha magas, akkor az 1. időzítő az alulcsorduláskor megszakítást vált ki.
4. bit: mint a 3. bit, de a 2. időzítőre vonatkoztatva
5. bit: nincs szerepe
6. bit: megszakítás a 3. időzítő alulcsordulásakor
7. bit: jelzi, hogy megszakítás történt

A TED-ben három, egyenként 2–2 byte hosszúságú időzítő van. Az időzítőkben levő értékeket a rendszer ütemjele (1 MHz) folyamatosan csökkenti. A regisztereik olvashatók, ill. ezekbe új indulási érték is tölthető. A regiszterek címeit a TED regisztereiről a függelékben adott áttekintés tartalmazza. Ha pl. az 1. időzítőbe 60000-et töltünk, akkor ez az érték a másodperc egymilliomod része alatt 1-gyel csökken, és 60 milliszekundum (ms) múlva 0 lesz. A következő rendszerütem során az értéke 65535 lesz, ami azt jelenti, hogy egy alulcsordulás következett be. Ha ebben az időpontban a megszakítás engedélyező regiszter hármas bitje magas, akkor a megszakítás érzékelő regiszter hármas és hetes bitjei is magasak lesznek. Ennek hatására egy megszakítás kérés érkezik a processzorhoz.

A hetes bit mindig akkor lesz magas, ha a megszakítás érzékelő regiszter többi bitjei közül egy vagy több magas. A megszakító rutinban most nagyon egyszerűen megállapítható, hogy melyik egység, és milyen körülmények között váltotta ki a megszakítást. A megszakító rutin befejeződése előtt a megszakítás érzékelő regiszterben a magasra állított bitet még törölni kell. Ez pl. úgy történhet, hogy a regiszter tartalmát elolvassuk, majd visszaírjuk a regiszterbe:

```
LDA $FF09
STA $FF09
```

Az operációs rendszer részéről a körülményektől függően az időzítők vagy a rasztersugár válthatnak ki megszakítást. A raszterregiszter 9 bitből áll. A kilencedik bit a megszakítás engedélyező regiszter 0. bitje. Mivel a képernyőn megjelenő képet is a TED állítja elő, ezért mindig tudja, hogy a rasztersugár a képernyőnek éppen melyik sorát rajzolja. Ha ez a sor megegyezik a $FF0B regiszter bitjei és a $FF0A regiszter nulladik bitje által meghatározott értékkel, akkor ez – feltéve, hogy a megszakítás engedélyező regiszter 1. bitje magas – megszakítást vált ki. Ha más készülékeket nem csatlakoztattunk a géphez, akkor ez a megszakítás mindig aktív.

Úgy gondoljuk, hogy most már eleget tudunk a megszakításokról és ezek programozásáról. Ennyi száraz elmélet után jöjjön valami színesebb, aminek gyakorlati hasznát is vehetjük. Az itt következő program minden egyes billentyű lenyomásakor hangjelzést ad. Mivel ez a program csak a megszakítások alatt fut, a számítógépet minden tekintetben a megszokott módon használhatjuk. Írjuk be előbb a BASIC betöltő programot. A magyarázat utána következik.

```
10 FOR I=DEC("065E") TO DEC("06AA")
20 READ A$
30 POKE I,DEC(A$)
40 B = B+DEC(A$)
50 NEXT
60 IF B<>10644 THEN PRINT"HIBA A DATA SOROKBAN"
100 DATA 78,A9,73,8D,14,03,A9,06,8D
101 DATA 15,03,A5,CD,85,D0,A5,CA,85
102 DATA D1,58,60,A5,CA,C5,D1,F0,04
103 DATA 85,D1,D0,08,A5,CD,C5,D0,F0
104 DATA 19,85,D0,A5,C6,C9,40,F0,11
105 DATA A9,FF,8D,0E,FF,A9,1F,0D,11
```

```
106 DATA FF,8D,11,FF,A9,04,85,D2,C6
107 DATA D2,D0,08,A9,E0,2D,11,FF,8D
108 DATA 11,FF,4C,0E,CE
```

Indítás előtt tároljuk a programot kazettára vagy lemezre. Ez után indíthatjuk RUN-nal. Ha ekkor HIBA A DATA SOROKBAN üzenetet kapunk, akkor valamit elírtunk. Ellenőrizzük a vizsgáló összeget és a DATA sorokat. Javítás után ismét tároljuk a programot. A gépi kódú program most a tárban van, és SYS 1630 utasítással indítható. A READY megjelenése után most CTRL és a SHIFT kivételével minden billentyű lenyomásakor egy hangnak is meg kell szólalnia. Így akár vakon is írhatunk a gépen, nem kell folyton a képernyőt nézni.

Mivel a SOUND utasítás a programból csak korlátozottan használható, a program működését az assembler listán magyarázzuk el. Ez lehetővé teszi, hogy mindenki a saját igényei szerint alakíthassa.

```
065E 78          SEI
065F A9 73       LDA  #$73
0661 8D 14 03    STA  $0314
0664 A9 06       LDA  #$06
0666 8D 15 03    STA  $0315
0669 A5 CD       LDA  $CD
066B 85 D0       STA  $D0
066D A5 CA       LDA  $CA
066F 85 D1       STA  $D1
0671 58          CLI
0672 60          RTS
```

A programnak ebben a részében „eltérítjük" a megszakítás vektort. A vektor ezután a $0673 címen kezdődő, saját programunkra mutat. Először azonban egy SEI utasítással feltétlenül meg kell akadályozni a megszakítást. Ha ugyanis az eltérítés közben érkezne egy megszakítás, akkor a számítógép azonnal „lemerevedne". Azért, hogy a jel kiírása és a hang megszólalása szinkronban legyen, a kurzor éppen aktuális pozícióját beírjuk a $D0 és $D1 címekre. Ezután a CLI-vel ismét engedélyezzük a megszakítást, és az RTS-sel visszatérünk a BASIC-be. Ezzel az előkészületi lépéseket, vagy más, gyakran használt szakkifejezéssel mondva, az inicializálást elvégeztük. Ha most bekövetkezik egy megszakítás, akkor az először a mi programunkat hajtja végre.

```
0673 A5 CA       LDA  $CA
0675 C5 D1       CMP  $D1
0677 F0 04       BEQ  $067D
0679 85 D1       STA  $D1
067B D0 08       BNE  $0685
067D A5 CD       LDA  $CD
067F C5 D0       CMP  $D0
0681 F0 19       BEQ  $069C
0683 85 D0       STA  $D0
```

A programunk a billentyűzetet nem figyeli. Ezt a feladatot továbbra is az operációs rendszer végzi a megszakító rutin további részében. Ezért a hangkeltésre mindig csak a megszakítást követően kerül sor. Először azt vizsgáluk, hogy változott-e a képernyőn a kurzor pozíciója. Ha igen, akkor az új értéket beírjuk a $D0, ill. $D1 címekre.

```
0685 A5 C6          LDA  $C6
0687 C9 40          CMP  #$40
0689 F0 11          BEQ  $069C
```

Mivel a kurzor pozíciója a LIST, PRINT és más utasítások hatására is változik, és ennek megfelelően a hang folyamatosan szólna, ebben a programrészben azt vizsgáljuk, hogy a kurzor pozíciója egy billentyű lenyomása miatt változott-e meg. A Plus/4-es a következő megszakításig az utoljára lenyomott billentyű kódját megjegyzi a $C6 címen.

```
068B A9 FF          LDA  #$FF
068D 8D 0E FF       STA  $FF0E
0690 A9 1F          LDA  #$1F
0692 0D 11 FF       ORA  $FF11
0695 8D 11 FF       STA  $FF11
```

Ezekben a sorokban szólaltatjuk meg a hangot. A $FF0E címre beírjuk a frekvenciát. A $FF11-ben bekapcsoljuk a TED 1. számú hangját, és a hangerőt maximumra állítjuk.

```
0698 A9 04          LDA  #$04
069A 85 D2          STA  $D2
069C C6 D2          DEC  $D2
069E D0 08          BNE  $06A8
```

Ezekkel az utasításokkal egyszerű módon rögzítjük a hang időtartamát. A hang hossza három megszakítási ciklus lesz, tehát a harmadik megszakítás kérés után kikapcsolódik.

```
06A0 A9 E0          LDA  #$E0
06A2 2D 11 FF       AND  $FF11
06A5 8D 11 FF       STA  $FF11
06A8 4C 0E CE       JMP  $CE0E
```

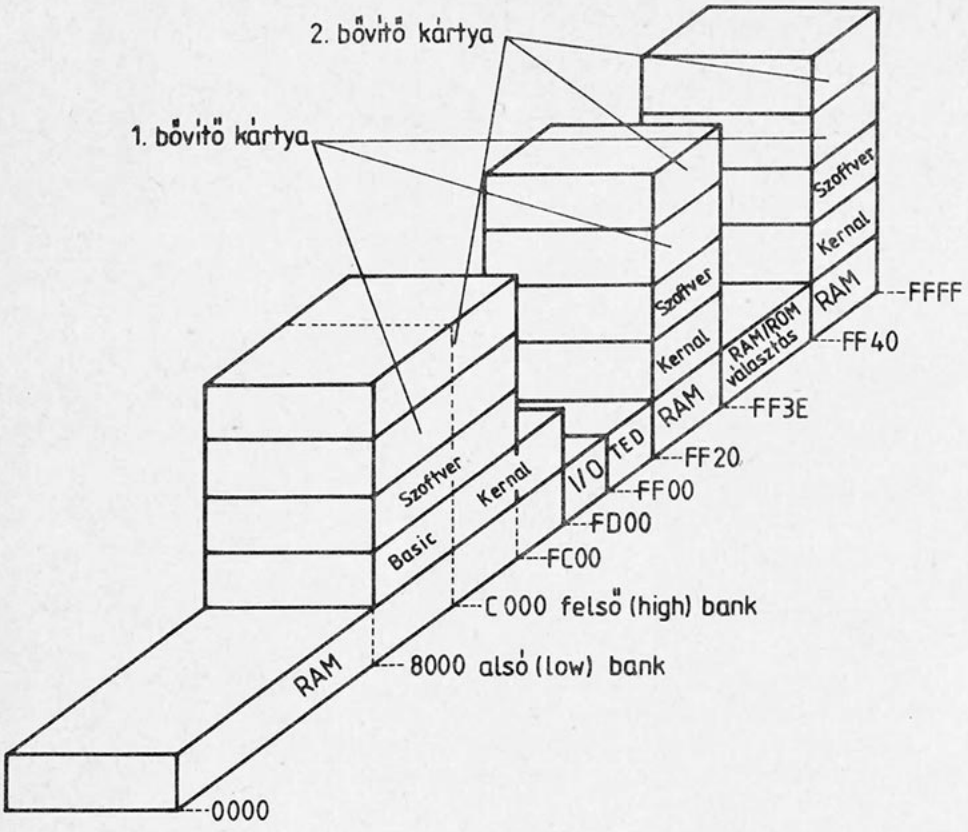
A kikapcsoláshoz a hangerőt nullára állítjuk, és ezzel egyidejűleg a $FF11 regiszter 4. bitjét, amely a hang be-, ill. kikapcsolását végzi, töröljük. Mivel a gépünknek a továbbiakban a szokott módon kell a feladatát ellátnia, visszatérünk az operációs rendszer tulajdonképpeni megszakító rutinjába.

Ezzel a megszakítások programozásával kapcsolatos mondanivalónk végére értünk. Reméljük, hogy a Plus/4-esnek ezzel a sokak számára még ismeretlen világába tett kis kirándulásunkkal sikerült felkeltenünk az érdeklődést. Mindenesetre javasoljuk a további kísérletezgetést, hiszen itt csak töredékét tudjuk bemutatni azoknak a lehetőségeknek, amelyekre a számítógép képes. A következő fejezetben, amely a memórialapozásról szól, ismét találkozunk ezzel a témával.

21.2. A MEMÓRIALAPOZÁS

A mikroelektronikai alkatrészek árainak csökkenése egyre nagyobb tárkapacitású számítógépek építését tette lehetővé. Ennek során a fejlesztők hamar beleütköztek a 16 bites címbuszok által felállított korlátokba. Már a 2. és a 3. fejezetekben említettük,

hogy 16 biten maximum 64 kbyte tárterület címezhető meg. Nos, a Plus/4-esben már 64 kbyte RAM van. Ehhez jön még 64 kbyte ROM. Természetesen jogos kérdés, hogy hogyan lehetséges ez? A varázsige, amely ezt a problémát megoldja, az úgynevezett memórialapozás (bankswitching). Nézzük meg ehhez a 21.1. ábrát.



21.1. ábra A Plus/4-es tárának felépítése

Természetesen 16 címvezetéken „egy darabban" csak 64 kbyte tárterület címezhető meg. Az azonban megoldható, hogy a tár (pl. a RAM) egy részét kikapcsoljuk (32768-tól 65535-ig) és a helyére ROM területet kapcsoljunk. Ezt a műveletet nevezzük memórialapozásnak. Az ábrán látszik, hogy az egyes tárak egyszerűen egymás fölött, egymással párhuzamosan helyezkednek el. Így pl. a 32768-as ($8000) cím lehet egyszer RAM tárcím, de lehetne még a BASIC ROM-ban, akárcsak a beépített felhasználói program alsó ROM-jában, és még két, külső, bővítő ROM-ban is.

Ezek között a tárak között tetszés szerint lehet ide-oda kapcsolni. Ezt értjük a memórialapozás (bankswitching) alatt. A számítógép természetesen összesen mindig csak 64 kbyte tárat lát. Az viszont teljesen mindegy a számára, hogy ez a 64 kbyte milyen tárrészekből áll, és hogy ezek között hogyan lapozunk. Csak az a fontos, hogy a CPU-hoz mindig logikus és végrehajtható utasítások érkezzenek.

Ebben a fejezetben a memórialapozással és az ezzel kapcsolatos problémákkal ismerkedünk meg.

Az egyes tárrészek közötti átkapcsolás természetesen nem valami kapcsolóval, hanem szoftver útján történik, mégpedig, úgy, hogy bizonyos tárcímekre kell valamit beírnunk. Ezeket a tárcímeket tehát „kapcsolók"-nak tekinthetjük.

Először is alapvetően a RAM és ROM között lapozhatunk: az írandó tárcímek $FF3E és $FF3F.

STA $FF3E a ROM-ot kapcsolja be (azaz lapozza felülre).
.STA $FF3F a RAM-ot kapcsolja be (azaz lapozza felülre).

Ha a RAM van bekapcsolva, akkor ez a $0000–$FCFF, az $FF20–$FF3D és a $FF40–$FFFF közötti tárterületeket foglalja el. A többi címen a TED és az I/O egységek vannak.

Vigyázat! Ha a ROM-ot akarjuk bekapcsolni, akkor előzőleg a megszakítást feltétlenül le kell tiltani. Ehhez a megszakítás jelzőbitet magasra kell állítani. (SEI, $78).

A $FD00–$FEFF közötti tárcímeket mindig csak az I/O egységek használhatják. A $FF00–$FF1F közötti címeket, valamint a $FF3E és a $FF3F címeket mindig a TED foglalja le. A ROM és a RAM közötti átkapcsoláskor egyébként az akkumulátor tartalma nem változik.

Ha most az STA $FF3E utasítással átkapcsolunk ROM-ba, akkor az éppen aktuális ROM kapcsolódik be. A $FD00–$FEFF, a $FF00–$FF1F közötti, valamint a $FF3E és a $FF3F címek azonban most is le vannak foglalva az I/O egységek és a TED számára. Ezeket a területeket tehát soha sem foglalhatja el a RAM vagy a ROM.

Az előbb az „éppen aktuális" ROM-ról beszéltünk. Amint az ábrából látszik, négy különböző ROM közötti átkapcsolásra van lehetőség. Ezen túlmenően a tár még alsó (low) bankra és felső (high) bankra is feloszthatő. Az alsó bank a $8000–$BFFF, a felső bank a $C000–$FFFF közötti tárcímeken van.

A Plus/4-es tartalmaz egy három, 74LS175, 74LS27 és 74LS139 típusjelű chipből álló átkapcsoló logikai áramkört. Ez az első pillantásra bonyolultnak tűnő kapcsolás lehetővé teszi, hogy a tár egyes elemeit 16 különböző konfiguráció szerint kapcsoljuk össze. Így lehetséges pl. az, hogy a $8000–$BFFF között bekapcsoljuk a BASIC ROM-ot, és ezzel egyidejűleg a $C000–$FFFF területen pedig egy, a bővítő csatlakozóba helyezett külső ROM-ot vagy RAM-ot. A 16 különböző lehetőség között a $FDD0–$FDDFcímek írásával kapcsolhatunk át. Az itt következő táblázatban megadjuk, hogy melyik tárcím, melyik konfigurációt kapcsolja be (A KERNAL ROM a számítógép operációs rendszerét jelenti).

21.1. táblázat:
A tár lehetséges konfigurációi és a bekapcsolási címük

Utasítás	Alsó bank $8000–$BFFF	Felső bank $C000–$FFFF
.STA $FDD0	BASIC ROM	KERNAL ROM
.STA $FDD1	beépített felh. program, alsó	KERNAL ROM
.STA $FDD2	1. bővítő kártya, alsó	KERNAL ROM
.STA $FDD3	2. bővítő kártya, alsó	KERNAL ROM
.STA $FDD4	BASIC ROM	beép. felh. program, felső

Utasítás	Alsó bank $8000–$BFFF	Felső bank $C000–$FFFF
.STA $FDD5	beép. felh. program, alsó	beép. felh. program, felső
.STA $FDD6	1. bővítő kártya, alsó	beép. felh. program, felső
.STA $FDD7	2. bővítő kártya, alsó	beép. felh. program, felső
.STA $FDD8	BASIC ROM	1. bővítő kártya, felső
.STA $FDD9	beép. felh. program, alsó	1. bővítő kártya, felső
.STA $FDDA	1. bővítő kártya, alsó	1. bővítő kártya, felső
.STA $FDDB	2. bővítő kártya, alsó	1. bővítő kártya, felső
.STA $FDDC	BASIC ROM	2. bővítő kártya, felső
.STA $FDDD	beép. felh. program, alsó	2. bővítő kártya, felső
.STA $FDDE	1. bővítő kártya, alsó	2. bővítő kártya, felső
.STA $FDDF	2. bővítő kártya, alsó	2. bővítő kártya, felső

Lássunk egy példát:

```
.1000 A2 05     LDX #$05        ; a konfiguráció száma
.1002 9D D0 FD  STA $FDD0,X     ; a konfiguráció bekapcsolása
.1005 86 FB     STX $FB         ; a szám $FB-be (megszakításhoz)
.1007 4C 03 80  JMP $8003       ; a program indítása
```

Ezzel a programmal a táblázatban szereplő ötödik lehetőséget kapcsoltuk be. Ennél a konfigurációnál a $8000–$BFFF területen a Plus/4-esbe beépített felhasználói program alsó része, a $C000–$FFFF területen pedig a program felső része helyezkedik el. Ezzel a példaprogrammal tehát a Plus/4-esbe beépített, ROM-ban levő felhasználói programot kapcsoljuk be. A program kezdőcíme $8003.

A példánkban a kiválasztott konfigurációnak a számát beírtuk a $FB címre. Ennek a célja a következő: mint tudjuk, a processzor másodpercenként kb. 100-szor végrehajt egy megszakítási rutint. Ebben a rutinban kérdezi le pl. a billentyűzetet is. A megszakító rutin után azt a konfigurációt kapcsolja be, amelynek a száma a $FB címen áll. Alapesetben a $FB értéke $00. A táblázatból láthatjuk, hogy ez a BASIC ROM-ot és a KERNAL ROM-ot jelenti. Ha tehát a példaprogramunkban a $FB értékét nem változtattuk volna meg, akkor a következő megszakítás ismét csak a régi konfigurációt kapcsolná be.

Most valaki megkérdezhetné, hogy miért nem írjuk be a kívánt értéket közvetlenül a $FB-re? Az ötlet ugyan nem rossz, de a gyakorlatban mégsem valósítható meg. A kívánt tárkonfiguráció ugyan ténylegesen létrejönne, csak azt nem tudjuk, hogy mikor jön a következő megszakítás. Ha ugyanis előzőleg már át is ugrottunk a ROM-ba, ez még mindig csak a BASIC ROM. Ezután jönne valamikor a következő megszakítás, amelyik a beépített program ROM-ját kapcsolja be. Az, hogy most a CPU ebben a ROM-ban éppen egy értelmezhető utasítást kapna-e el, tisztán szerencse kérdése, mindenesetre a program kifogástalan indítása nem lenne biztosítható.

A ROM-ok bekapcsolásához és indításához a Plus/4-es operációs rendszere némi segítséget ad. Így például a $FCC9 címen kezdődő rutin azt a modult kapcsolja be, amelynek a száma az X regiszterben van. A rutin ezután arra a címre ugrik, amelynek az alsó byte-ja a $02FE címen, felső byte-ja pedig a $02FF címen van. Lássunk erre is egy példát:

```
.1000 A9 03      LDA #$03
.1002 A0 80      LDY #$80
.1004 8D FE 02   STA $02FE    ;kezdocim also byte-ja
.1007 8C FF 02   STY $02FF    ;kezdocim felso byte-ja
.100A A2 05      LDX #$05     ;5-os modul, lasd a szoveget
.100C 86 FB      STX $FB      ;modul szama
.100E 4C C9 FC   JMP $FCC9    ;modul bekapcsolasa es
                               inditasa
```

Ez a rutin a $8003-as címtől kezdve elindítja a ROM-ba beépített felhasználói programot.

Az előbb egy új kifejezést használtunk, a modult. Ez a fogalom a fejezet és a könyv további részében még többször előfordul, ezért mindjárt itt tisztázzuk, hogy a modul alatt mindig valamelyik lehetséges tárkonfigurációt értjük. Ha tehát a táblázatunkból az 5-ös lehetőséget keressük ki, akkor ezen az 5-ös modult értjük.

A Plus/4-esen bekapcsolás után a 0-ás modul az aktív (BASIC ROM és az operációs rendszer).

Ha még egy pillantást vetünk a 21.1. ábrára, ekkor egy további érdekességet fedezhetünk fel: a $FC00–$FCFF területen vagy a RAM, vagy az operációs rendszer ROM-ja van bekapcsolva, de sohasem valamilyen más ROM. Ennek megvan a maga oka: az operációs rendszer ROM-jának ezen a részén ugyanis olyan, fontos rutinok vannak, mint pl. a modul-reset, modul-start, és a megszakító rutin egy része. Ezekre a rutinokra állandóan szükség van, ezért vannak a gyakorlatilag mindig rendelkezésre álló tárterületen. Ez egyben azt is jelenti, hogy ez a $FC00–$FCFF közötti terület más ROM-okra vagy külső kártyákra nincs hatással, és fordítva, ezek sem érhetik el ezt a területet. Lehet, hogy az elmondottak egy kicsit bonyolultak, ezért foglaljuk össze:

A RAM és a ROM között úgy kapcsolhatunk át, hogy egy értéket írunk a $FF3E és a $FF3F címekre (ford. megj.: teljesen mindegy, hogy mekkora ez az érték).

```
.STA $FF3E   bekapcsolja a ROM-ot.
.STA $FF3F   bekapcsolja a RAM-ot.
```

Ezenkívül még a 21.1. táblázat szerinti 16 különböző RAM konfiguráció között lehet átkapcsolni. A konfigurációk mindegyikét modulnak nevezzük. Az egyes moduloknak számuk van, és ezt a számot a bekapcsolási címük és a $FDD0 különbsége adja. A kívánt modul a $FDD0–$FDDF közötti tárcím írásával kapcsolható be. Ezt követően a megszakító rutin számára a bekapcsolt modul számát be kell írni a $FB tárcímre.

21.3. A MEMÓRIALAPOZÓ RUTINOK

A könyv H függelékében megadtuk a nulláslap és a rendszerváltozók tárkiosztását a $0800-as címig. A $0494–$04DC és a $05EC–$06EB címeknél a memórialapozás megjegyzés szerepel. Nézzük meg ezeket a rutinokat a monitorprogram segítségével. Írjuk be a D 0494 utasítást. A képernyőn ekkor ezt látjuk:

```
.0494 8D 9C 04    STA $049C
.0497 78          SEI
.0498 8D 3F FF    STA $FF3F
.049B B1 00       LDA ($00),Y
.049D 8D 3E FF    STA $FF3E
.04A0 58          CLI
.04A1 60          RTS
```

Ebben a rutinban a program saját magát változtatja, mégpedig úgy, hogy az akkumulátor tartalmát beírja a $049C címre, és ezáltal módosítja a $049B címen álló, indirekt, indexelt utasítást. Ezzel a rutinnal a RAM tetszőleges tárcíme olvasható. A rutinban a megszakítás jelzőbit magasra állításával a megszakítást letiltjuk. Erre feltétlenül szükség van, mert ha akkor érkezne megszakítás, amikor a RAM van bekapcsolva, a számítógép egészen bizonyosan lemerevedne.

Legyen az akkuban pl. #$D0 és az Y regiszterben #$10. Ha most elindítjuk a rutint, akkor az akkuba annak a RAM címnek a tartalma töltődik, amelynek az alsó byte-ját a $00D0 tartalmának és az Y regiszter tartalmának az összege adja meg. A felső byte a $00D1 címen van, és az értéke mindannyiszor 1-gyel növekszik, amikor a $00D0 és az Y tartalmainak összeadásakor átvitel keletkezik.

Példa:

```
.1000 A2 00       LDX #$00
.1002 86 D0       STX $D0
.1004 A2 A0       LDX #$A0
.1006 86 D1       STX $D1
.1008 A0 10       LDY #$10
.100A A9 D0       LDA #$D0
.100C 20 94 04    JSR $0494
.100F 00          BRK
```

Ez a rutin a RAM-ban levő $A010 tárcím tartalmát olvassa. A memórialapozó rutinokat főként az operációs rendszer, a BASIC interpreter és természetesen a beépített programok használják.

A $0494–$04E6 közötti rutinok a következő feladatokat végzik:

Kezdőcím	Feladat
$0494	Egy RAM cím olvasása. Indirekt címzés, az Y regiszterrel indexelve. Az akkuban annak a nulláslapos címnek kell lennie, amelyik a keresett cím alsó byte-ját tartalmazza.
$04A5	Egy RAM cím olvasása. A cím alsó byte-ját a $003B, felső byte-ját a $003C tartalmazza. Indirekt címzés, az Y regiszterrel indexelve.
$04B0	Mint a $04A5, de az alsó byte a $0022 címen, felső byte a $0023 címen.
$04BB	Mint a $04A5, de az alsó byte a $0024 címen, felső byte a $0025 címen.
$04C6	Mint a $04A5, de az alsó byte a $006F címen, felső byte a $0070 címen.
$04D1	Mint a $04A5, de az alsó byte a $005F címen, a felső byte a $0060 címen,
$045C	Mint a $04A5, de az alsó byte a $0064 címen, felső byte a $0065 címen.

A Plus/4-es bekapcsolása után a $05EC–$06EB területen csak egyetlen rutin van.

A kezdőcíme $05F5. Ezzel a rutinnal modulok kapcsolhatók be, és ezek tetszőleges címtől indíthatók. Ha ezt a rutint közvetlenül a gép bekapcsolása után indítjuk, akkor ez a beépített felhasználói programot indítaná el. A rendszerváltozók tárkiosztási listáján látható, hogy a $05F0–$05F4 címek mellett a Long Jump megnevezés áll. Ez a Plus/4-es esetén különböző modulok közötti ugrást jelent.

Nézzük meg monitorprogrammal a $05F5 címen kezdődő rutint. Könnyen felismerhető, hogy itt mi történik. A rutin először betöli az X regiszterbe a modul számát (itt a #$05-öt, ami a beépített felhasználói programot jelenti), majd meghatározza a kezdőcímet, és azt beírja a $05F0 és $05F1 címekre. Ezután elindítja a $FCFA-an kezdődő rutint, amely bekapcsolja az X regiszter szerinti modult és végrehajt egy indirekt ugrást a $05F0 címre.

Az operációs rendszer ROM még más rutinokat is tartalmaz, amelyek a memórialapozáshoz használhatók:

$FCF7 Azt a modult kapcsolja be, amelynek a száma az X regiszterben van. A régi modul számának az akkuban kell lennie. A rutin ezután azt a rekeszt olvassa, amelynek a címe $00BE-en (alsó byte) és $00BF-en (felső byte) van. A címzésmód ismét indirekt, az Y regiszterrel indexelve. (($BE),Y). Ezután a rutin visszakapcsolja a korábbi modult. Ily módon azokból a modulokból is kiolvashatók adatok, amelyek ki vannak kapcsolva.

$FC89 Azt a modult kapcsolja be, amelynek a száma az X regiszterben van. A régi modul számának most is akkuban kell lennie. Ezután a modulon belül elindítja azt a programot, amelynek a kezdőcíme a $00F0 és $00F1 címeken van. (Ez tulajdonképpen szubrutinhívás egy másik modulból ford. megj.). Amikor a CPU ezen a programon belül találkozik egy RTS-sel, akkor ismét a régi modul kapcsolódik be, a program végrehajtása pedig az eredeti programnak azon a részén folytatódik, ahonnan ez a $FC89-es rutint meghívta (visszatérés szubrutinból).

$FCB3 A rutin neve PULS. Ezt a rutint mindegyik megszakítás aktiválja. Ez a rutin bekapcsolja a nullásmodult (BASIC ROM, operációs rendszer ROM), és elindítja a megszakító rutint. A PULS rutin nagyon jól használható a bővítőcsatlakozóba helyezett, saját szoftvermodulunkhoz. A megszakító rutin után ismét a régi modul kapcsolódik be.

$FCC9 Azt a modult kapcsolja be, amelynek a száma a $FB címen van. Először bekapcsolja a modult, majd indirekt ugrást hajt végre a $02FE címre. A kezdőcím alsó byte-ja a $02FE címen, a felső byte-ja a $02FF címen van.

21.4. MODUL-RESET

A Plus/4-es bekapcsolásakor, ill. a RESET gomb benyomásakor a gép végrehajtja a 7. fejezetben, a funkcióbillentyűkkel kapcsolatban már említett modul-reset rutint. Ha valakinek egyszer az az ötlete támad, hogy saját EPROM-okat égessen, és ezeket a bővítő csatlakozóba helyezve működtesse, akkor fontos tudnia, hogy mi történik ebben a reset-rutinban.

Ez a rutin azt is lehetővé teszi, hogy a modulok automatikusan induljanak. Ha pl. valaki a Plus/4-es számítógépet meghatározott feladatokra akarja használni, és az ehhez szükséges programok EPROM-ban vannak, akkor nagyon hasznos lehet, hogy ezek a programok a számítógép bekapcsolásakor automatikusan elinduljanak.

Ez modul-reset rutin az operációs rendszer ROM-jában, a $FC1E címen kezdődik. Ez a rutin a következőket végzi:

Egymás után bekapcsolja a 15-ös, 10-es, 5-ös és nullás modulokat. Az éppen aktív modulban a $8007–$8009 címeken keresi a CBM azonosító jelet. Ha megtalálja ezt a jelet, akkor az illető modulból elolvassa a $8006 cím tartalmát. Az így kapott érték a modul jelzőszáma. A BASIC ROM jelzőszáma 0, a beépített felhasználói programé $0C.

Ezt a jelzőszámot egy táblázatban tárolja (a $05EC–$05EF közötti címeken). Ha ennek a jelzőszámnak az értéke 1, akkor a modult elindítja. Ennek ismeretében tehát lehetőség van arra, hogy a rutin a saját modulunkat is automatikusan indítsa.

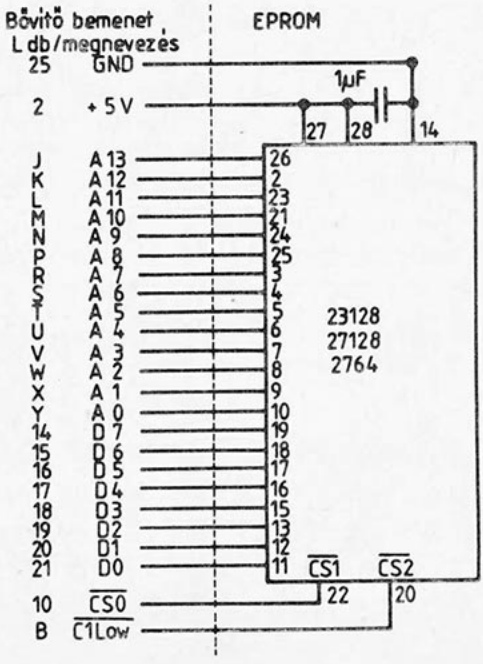
Nézzük meg egy automatikusan induló modul elejét (memoria dump):

```
>8000 4C 0A 80 00 00 00 01 43
>8008 42 4D ...
```

A $8000-es címen van az ugró utasítás a tulajdonképpeni modulprogramba, a $8006-os címen az 1 érték az automatikus indításhoz, a $8007–$8009 címeken pedig a CBM ASCII-kódjai. A program a $800A címen kezdődik.

21.5. EGY EPROM CSATLAKOZTATÁSA A BŐVÍTŐ BEMENETHEZ

A 21.2. ábrán bemutatjuk, hogy hogyan csatlakoztatható egy EPROM a bővítő bemenethez.



21.2. ábra

A példánkban az EPROM a $8000-től kezdődő címeket foglalja el. Ekkor egy 23128 vagy 27128 típusjelű EPROM a $8000–$BFFF területet veszi igénybe. Csatlakoztatható azonban egy 2764 típusú EPROM is. Ezeknél a típusoknál a 13-as címvezetékre nincs szükség, mert az EPROM csak 8 kbyte-os. Ekkor a 26-os lábat a +5 V-ra kell kötni. A 2674-es tehát a $8000–$9FFF területet foglalja el.

Természetesen az EPROM címtartománya a $C000 címtől is kezdődhet. Ehhez az EPROM 20-as lábát a C1 High-ra (bővítő 6-os csatlakozója), a 22-es lábát pedig a CS1-re (bővítő 9-es csatlakozója) kell kötni. Egyszerűbb a dolog azonban, ha az EPROM 22-es lábát a földre kötjük, a kapcsolás ekkor is kifogástalanul működik. Ha a $8000–$FFFF közötti, teljes címtartományt ki akarjuk használni, akkor 2 db, 27128 vagy 23128 típusú EPROM-ot kell csatlakoztatni. Az első EPROM 20-as lábát a C1 Low-ra, a második EPROM 20-as lábát pedig a C1 High-ra kell kötni. A C1 Low és a C1 High helyett használható a C2 Low és a C2 High is. Természetesen négy EPROM is csatlakoztatható, ha mind a négy kiválasztó csatlakozást igénybe vesszük. Az EPROM-ok és a számítógép címtartományainak egymáshoz rendelésére és az EPROM-okban levő programok indítására vonatkozóan javasoljuk a 21.1. táblázat, és az ismertetett példák tanulmányozását.

21.6. DATA SOROK ELŐÁLLÍTÁSA

Az előző fejezetekben már tekintélyes mennyiségű ismeretet szereztünk a számítógépünkről. Bizonyára van, aki már rövidebb gépi kódú programokat is írt. Ha most ezeket a rutinokat már meglevő BASIC programokba szeretnénk beilleszteni, akkor újabb nehézségekkel kell szembenéznünk. Vagy utólag kell ezeket a rutinokat betölteni, ami meglehetősen körülményes, vagy a monitorprogram segítségével DATA sorokat kell előállítani. Ez az eljárás viszont nagyon időigényes. Ebben a fejezetben bemutatunk egy lényegesen gyorsabb és kényelmesebb eljárást. Megírunk egy programot, amelynek segítségével DATA sorokat lehet előállítani.

Magát a programozást és a programot nagyon részletesen ismertetjük, hogy a gépi kódú nyelv megértését még könnyebbé tegyük. Azért, hogy a program beírása egyszerűbb legyen, megadjuk a BASIC nyelvű betöltő programot. A program működését természetesen az assembler lista alapján magyarázzuk el.

A megoldandó feladat:

Egy olyan, gépi kódú programot kell írnunk, amely lehetővé teszi, hogy a tárban lévő gépi kódú programokat DATA sorokban ábrázolhassuk. Az így előállított adatokat egy BASIC nyelvű betöltő programmal ismét be kell tudnunk írni a tárba. Már most látszik az első probléma. Ahhoz, hogy egy BASIC programsort egy programmal elő tudjunk állítani, először is azt kell tudnunk, hogy hogyan épül fel egy ilyen sor. Kezdjük hát akkor ezzel.

21.6.1. Egy BASIC programsor felépítése

Továbbolvasás előtt hajtsunk végre egy RESET-et. Ezután írjuk be:

```
10 PRINT "PLUS4"
```

Kapcsoljuk be a monitorprogramot, majd írjuk be:

```
M 1000 1010
```

Az eredmény:

```
>1000 00 0F 10 0A 00 99 20 22
>1008 50 4C 55 53 34 22 00 00
>1010 00
```

A harmadik sort nem írtuk végig, mert a többi byte már érdektelen, és az értékük véletlenszerű lehet.

Nézzük meg közelebbről a kapott eredményt. Az első érték nulla. Ez a byte mindig nulla, mert ez jelzi a BASIC interpreter számára, hogy itt kezdődik a programtár. A következő két byte értéke 0F és 10. Ez a két byte együtt egy címet képez. Mivel a 7501-es mikroprocesszornál a címek ábrázolása úgy történik, hogy a processzor a tárban először a cím alsó (low) byte-ját, és utána a felső (high) byte-ját helyezi el, ahhoz, hogy a tényleges címet megkapjuk, ezt a két byte-ot meg kell cserélni. Az így kapott cím $100F. Ezt a címet kapcsoló- vagy csatolócímnek nevezzük. Ez mondja meg az interpreternek, hogy a tárban hol kezdődik a következő programsor.

A következő két byte értéke 0A és 00. Ez nem újabb cím, hanem a BASIC sorunk sorszáma, ismét a fordított – alsó byte/felső byte – sorrendben tárolva. Ha a két byte sorrendjét megcseréljük, akkor az eredmény $000A, ami decimális számrendszerben valóban 10-nek felel meg.

A következő byte értéke 99. Ha ezt átszámoljuk decimális számrendszerbe, akkor 153-at kapunk. Nézzük meg a függelékben a BASIC tokenek táblázatát. A 153 mellett a PRINT utasítást találjuk. Tehát az interpreter a programsor értelmezésekor az utasításszót átalakítja egy számmá, és csak ezt a számot tárolja a tár megfelelő címén. Könnyű belátni, hogy ezáltal igen sok tárhely takarítható meg. A PRINT karakterenkénti tárolásához öt byte-ra lenne szükség, így viszont elegendő egy is. Nézzük most a következő byte-ot.

Ennek a byte-nak az értéke 20. Reméljük, hogy mindenkinél az, akinél ugyanis nem, az a programsor beírásakor elfelejtett szóközt írni a PRINT és az idézőjel közé. Ezt a hibát természetesen már korábban is észre kellett vennünk, hiszen így a csatolócímnek $100E-nek kellett lennie. Ha így lenne, akkor javítsuk ki a hibát, hogy a magyarázatunk a továbbiakban is követhető legyen. Térjünk vissza a $20-hoz. Ennek decimális rendszerbeli megfelelője a 32. Ez a szám a szóköz (közismert angol nevén SPACE) ASCII-kódja.

A többi már meglehetősen egyszerű: a 22 az idézőjel kódja, az 50 a P, a 4C az L, az 55 az U, az 53 az S, és végül a 34 a 4 kódja. A következő byte ismét az idézőjel ASCII-kódja. Vegyük szemügyre most az utolsó három byte-ot. Mindegyik értéke $00. Amikor az interpreter egy $00 értékű byte-tal találkozik, akkor erről tudja meg, hogy az éppen feldolgozás alatt lévő BASIC sor végére ért. Most előveszi az előbb elolvasott csatolócímet, és megnézi, hogy a program hol folytatódik. A mi csatolócímünk a $100F címre mutat. Az ezen, valamint a következő címen levő két byte-nak kellene tartalmaznia a következő sor csatolócímét. Ezt azonban most nem teszik,

mivel mindkét byte értéke $00. Erről a két, $00 byte-ról veszi észre az interpreter, hogy vége a programnak, és kiírja a READY üzenetet. Az elmondottak alátámasztására, a monitorból való kilépés után írjuk be:

```
20 PRINT
```

Térjünk vissza a monitorba, és írjuk be:

```
M 1000 1010
```

A sorok most így alakulnak:

```
> 1000 00 0F 10 0A 00 99 20 22
> 1008 50 4C 55 53 34 22 00 15
> 1010 10 14 00 99 00 00 00
```

Az utolsó byte-ot az előzőekben említett okok miatt itt sem tüntettük fel. A tár tartalmában csak a második sor utolsó byte-ja és a harmadik sor változott. Számítsuk ki ezek alapján a következő csatolócímet. Nézzük ehhez a második sor utolsó és a harmadik sor első byte-ját. Ezek a 1015-ös címet alkotják. Ezen a címen ismét csak két $00 értékű byte van, ami tehát azt jelenti az interpreter számára, hogy befejezheti a program feldolgozását, mert itt van a program vége.

Foglaljuk össze:

Mindegyik programsorhoz öt sorkezelő byte-ra van szükség. Egy programsor első és második byte-ja egy csatolócímet képez. Ez a cím a programtárban annak a byte-nak a címe, amellyel a következő feldolgozandó programsor kezdődik. Ha az ezen és a következő címen levő byte értéke $00, akkor ez a program végét jelenti. A következő két byte (a harmadik és a negyedik) az éppen feldolgozás alatt álló programsor sorszámát jelenti. A programsor utolsó byte-ja $00. Az interpreter erről állapítja meg, hogy vége a sornak.

Ha figyelmesen olvassuk a fenti összefoglalást, akkor bizonyára mindenki számára világos lesz, hogy ha több programsort egy sorba fogunk össze, akkor ezzel tárhelyet takaríthatunk meg. A Plus/4-es nagy tárkapacitásának köszönhetően erre azonban aligha lesz szükségünk.

Legyen a következő feladatunk az, hogy a monitorprogram segítségével beírunk egy komplett DATA sort csatolócímmel, BASIC tokennel és minden szükséges tartozékkal. Lépjünk ki először a monitorból, és írjuk be a NEW-t. Most vissza a monitorba, és írjuk ezt:

```
M 1001 1010.
```

Tartalmazzon előkészítendő programsor két DATA elemet, a sorszáma pedig legyen 256.

A első két byte-ot egyelőre hagyjuk figyelmen kívül. Számítsuk át a sorszámot hexadecimális számrendszerbe. Az eredmény $0100. Ahhoz, hogy megkapjuk a harmadik és a negyedik byte-ot, a sorrendet meg kell változtatni (alsó byte, felső byte), tehát: $00 $01. A DATA BASIC tokenje 131, ami hexadecimálisan $83. Ez lesz tehát az ötödik byte. Írjuk be most a hatodik, hetedik és a nyolcadik byte-ba rendre a $30, $2C és a $30 értékeket. Ezek a nulla, vessző, és megint a nulla megfelelő ASCII-kódjai. A sort ezzel beírtuk úgy, hogy egy $00 byte következik a sor, majd további két $00 byte a program lezárásaként.

Térjünk vissza most a csatolócímre, amelynek tudvalevően a sor első két byte-ján kell állnia. A sorunk utolsó előtti $00 byte-jának a címe $100A. Az első két byte értékének tehát $0A és $10-nek kell lennie. Írjuk át ezeket a byte-okat ennek megfelelően. A végeredmény most tehát ez:

```
>1001 0A 10 00 01 83 30 2C 30
>1009 00 00 00
```

Lépjünk ki a monitorból, és adjuk ki a LIST utasítást. Az eredmény:

```
256 DATA0,0
```

Amit látunk, az ugyan nagyon szép, de a dolog azért nem ennyire egyszerű. Írjuk be:

```
257 PRINT
```

A RETURN lenyomása után a rendszer lemerevedik. Ennek az oka néhány mutató, amelyeket a sor beírása után nem módosítottunk. Azt, hogy mit jelent a mutató és hogy mi a szerepe, a következő fejezetből tudjuk meg. Addig is keltsük életre a számítógépünket a RESET gomb benyomásával.

21.6.2. A BASIC programtár felépítése

Az előbbi fejezetben egy kis kudarc ért bennünket, amelyet most helyre kell hoznunk. A hiba azért csúszott be, mert még nem rendelkezünk néhány, a tárkezelésre vonatkozó alapvető ismerettel. Pótoljuk be hát ezeket a hiányosságokat.

A BASIC programtárba nem csak a program kerül bele, hanem az összes változó, tömbváltozó, karakterlánc és karakteres változó is. A tár kezelése dinamikus, ami azt jelenti, hogy egyes részei mindig az adott körülményeknek megfelelően alakulnak. Egy NEW után írjuk be a következőt:

```
PRINT FRE(X)
```

Az eredmény:

```
60669
```

Ez azt jelenti, hogy 60669 byte áll a BASIC programunk rendelkezésére. Jöjjön most ez:

```
DIM A 1000
PRINT FRE(X)
```

A még szabad tárterület 55657 byte-ra zsugorodott. A BASIC interpreter, amely egyben a tárkezelésért is felelős, megváltoztatott egy mutatót, és így 1000 elem tárolásához szükséges helyet lefoglalt. Erre a lefoglalt helyre most nincs szükségünk, ezért írjuk be:

```
CLR
```

Most ismét a teljes terület a rendelkezésünkre áll. Nézzük meg ezt az egészet monitorprogramban is.

```
M 2B 3A
>002B 01 10 03 10 03 10 03 10
>0033 00 FD FE FC 00 FD 00 FF
```

A képernyőn látható számsor néhány mutatót képez, amelyeket most részletesebben szemügyre veszünk.

Egy mutató mindig két byte-ból áll, és amint a nevéből is látszik, valahová mutat. Az első két tárcímen álló byte értéke $01 és $10. A mutatókat alkotó byte-ok tárolási sorrendje is alsó byte, felső byte. Cseréljük fel tehát a byte-okat, és ekkor $1001-et kapunk. Ennek persze semmi köze a híres mesekönyvhöz, hanem azt mondja nekünk és természetesen az interpreternek, hogy ezen a tárcímen kezdődik a BASIC tár. Igazából ez ugyan a $1000 címen kezdődik, de a tár első byte-jának mindig nullának kell lennie. Ezért ez a mutató a $1001-re az első csatolócím alsó byte-jára mutat.

A következő két byte értéke már a helyes sorrendben írva $1003. Vajon miért pont ez az érték? – kérdezhetnék többen. A válasz egyszerű: a programtár $1001-en kezdődik. Az előző fejezetben megállapítottuk, hogy ott egy két byte-os csatolócím áll. Nos, $1001 plusz $2 = $1003. Ez a mutató a változóterület kezdetére mutat. Egy újabb programsor beírásakor az interpreter ennek az értékét folyamatosan módosítja úgy, hogy az mindig az utolsó csatolócím utáni első byte-ra mutat. Ha a tárat egy felhőkarcolónak képzeljük el, ahol az egyes címeknek az egyes emeletek felelnek meg, akkor azt mondhatjuk, hogy ez a mutató a program növekedésével egyre felsőbb emeletekre vándorol.

A következő két byte által alkotott mutató értéke is $1003, és szintén felfelé vándorol. Ez a tömbváltozók (indexelt változók) területének kezdetére és egyúttal a változóterület végére mutat.

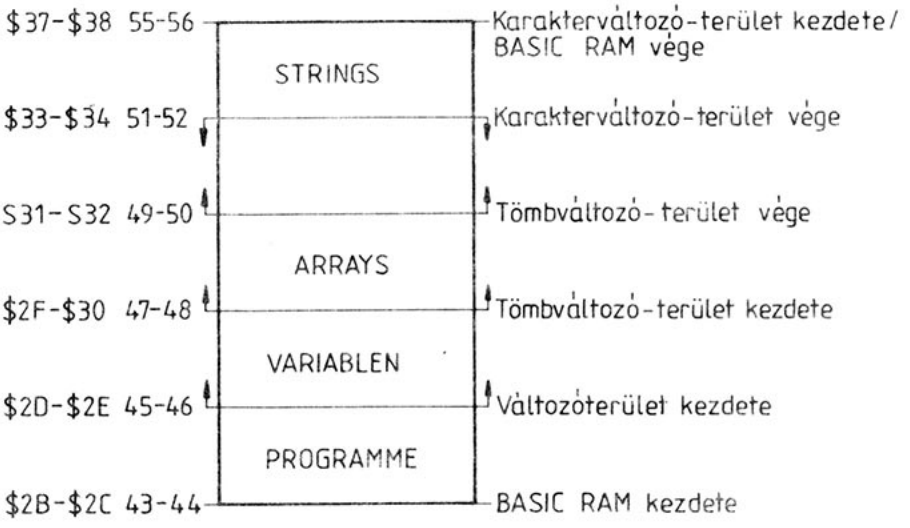
A most következő mutatókat legjobb egymással összefüggésben vizsgálni. Az első itt is $1003, a második $FD00. Az első mutató a tömbváltozók területének végére, a második pedig a karaktervváltozók területének végére mutat. Mivel a tömbváltozók kezdete mutató felfelé vándorol, logikus, hogy a tömbváltozók vége változó is ezt teszi. A programunkban tömböt még nem dimenzionáltunk, ezért a tömbváltozók kezdete és vége mutató értéke azonos.

Mi a helyzet a másik mutatóval? Nos, ez lefelé vándorol. Amikor ez a két mutató, tehát a tömbváltozók területének vége mutató és a karaktervváltozók területének vége mutató azonos értékű lesz, akkor kapjuk a híres/hírhedt OUT OF MEMORY ERROR hibaüzenetet.

Ennyi ismeret birtokában már biztosan mindenki kitalálja, hogy mi az oka a bekapcsoláskor megjelenő 60671 BYTES FREE üzenetben, és az utána azonnal kiadott PRINT FRE(X), 60669 válaszban levő byte-számok közötti eltérésnek. Az esetleg hibának vélt eltérést az okozza, hogy a változóterület kezdete mutató két byte-tal magasabb címre mutat, mint a BASIC-tár kezdete mutató.

Vizsgáljuk meg azt, hogy hol kezdődik a karaktervváltozók tárterülete. Ehhez hagyjuk figyelmen kívül a következő két byte-ot és vegyük szemügyre az ezután következőket. A következő kettő tartalma $00 és $FD. Az ezekből képzett mutató tehát a $FD00 címre mutat. Ez a cím egyrészt a BASIC-tár vége, másrészt a karaktervváltozók területének kezdete. A rendszer ezt a mutatót erre a fix értékre állítja be, ami azt jelenti, hogy ezt az interpreter nem változtatja meg.

Az elmondottak könnyebb megértése céljából ezekről egy kis rajzot is készítettünk.



21.3. ábra BASIC programtár felépítése

Na, sejtjük már, hogy miért történt az előbbi baleset? Elfelejtettük a BASIC program vége, és így persze a változóterület kezdete mutatót átállítani. Tulajdonképpen a tömbváltozók kezdete és vége mutatókat is át kell állítani, de ezt a munkát most már rábízzuk a számítógépre. Elvégre már úgyis régen dolgoztattuk. Először azonban újra be kell írni a programsorunkat a monitoron keresztül:

```
>1001 0A 10 00 01 83 30 2C 30
>1009 00 00 00
```

Ennek megfelelően állítsuk most feljebb a változóterület kezdete mutatót:

```
>002D 0C 10
```

Lépjünk ki a monitorból és írjunk be egy LIST-et. Úgy tűnik, hogy minden rendben van. Ez azonban még csak a látszat, hiszen a többi mutató értékét is megfelelőképpen módosítani kell. Ezt a BASIC CLR utasítása végzi el. Írjuk tehát be:

```
CLR
```

Most már végre nyugodtan beírhatunk egy újabb sort:

```
257 PRINT "PLUS/4"
```

A programot elindítva meggyőződhetünk arról, hogy a művelet sikerült. A CLR helyett RUN-t is írhattunk volna, mert ez is a helyükre állítja a mutatókat.

Ezt a meglehetősen unalmas procedúrát végzi majd el helyettünk a megírandó gépi kódú program. Mivel ez már egy kicsit hosszabb, sajnos nem fér el a $065E vagy decimálisan az 1630-as címen kezdődő szabad területen. A következő fejezetben arról lesz szó, hogy miként lehet a BASIC programtárban gépi kódú programok számára helyet foglalni úgy, hogy ezek a tárban levő BASIC programokat ne írják felül.

21.6.3. Tárhely foglalása

A Plus/4-esben a mutatók előbb vázolt állíthatósága nem csak az interpreter, hanem a programozó munkáját is segíti.

Az előzőekben már láttuk, hogy a BASIC-tár kezdete és vége mutató mindig fix értékre mutat. Ez tulajdonképpen azt jelenti, hogy ezeket az értékeket az interpreter nem változtatja meg. Egy kivétel azonban van. Ha bekapcsoljuk a Plus/4-es grafikáját, akkor a programtár kezdete mutató magasabb címre fog mutatni, és ennek megfelelően módosul a többi mutató is. Amikor a grafikát kikapcsoljuk, akkor a mutató ismét visszaáll a kiinduló értékre (természetesen vele együtt a többi is), tehát ugyanazokra a tárcímekre fognak mutatni, mint a grafika meghívása előtt.

Kapcsoljuk be ismét a monitort.

```
M 2B 32
>002B 01 10 03 10 03 10 03 10
```

Próbáljunk meg 256 byte hosszúságú tárterületet lefoglalni. Ehhez a következőt kell tennünk:

```
>002B 01 11 03 11
>2000 00
```

Lépjünk ki a monitorból és írjuk be a CLR-t. Most pedig: PRINT FRE(X). A tár aktuális nagysága már csak 60413 byte. Látszólag tehát 256 byte hiányzik. Ezek természetesen nem tűntek el, csak lefoglaltuk őket, és a BASIC-ből már csak PEEK vagy POKE segítségével olvashatók és írhatók. Lássuk a magyarázatot:

Csak a BASIC kezdet mutatót állítottuk át, amennyiben a $2B és a $2C címeken levő mutatót magasabb címre irányítottuk. Ez most a $1101 címre mutat, tehát ez a BASIC-tár új kezdőcíme. Még a változó terület kezdete mutatót is – csakúgy mint korábban – át kell állítani. A következő lépésnek is egyszerű a magyarázata. Jól tudjuk, hogy a programtár első byte-jának mindig nullának kell lennie. Ezt végeztük el a második sorban. A többi mutató módosításához szükség van még a CLR parancsra, és ezáltal a tár területe lényegesen kisebb lett. A tár jelenlegi mérete egy RESET-ig változatlan marad, még a RUN/STOP RESET sem változtatja meg.

Az így lefoglalt és üres területen most elhelyezhetők a gépi kódú programok vagy bármilyen más érték. Ha most bekapcsoljuk a grafikát, akkor ez elfoglalja a $1800–$1FFF közötti területet, és a BASIC tár a $2000-es címen kezdődik (mutató a $2001-re). Ha tehát grafikával is dolgoznunk kell, akkor ezzel az eljárással csak a $1000–$17FF közötti terület foglalható le.

A BASIC programtár kezdetének magasabbra állítása más vonatkozásban is kedvezőtlen. Ha pl. már beírtunk egy BASIC programot a tárba, és ezután az alsó tartományba még egy gépi kódú programot is be kell írnunk, akkor ez utóbbi a BASIC tár elejét felülírhatja, aminek következtében a gépünk szinte bizonyosan „fejre áll". Ezt az alsó terület tehát a gép bekapcsolása vagy egy RESET után a mutatók átírásával azonnal le kell foglalni.

Ennél jobb módszer a BASIC RAM vége/karaktervváltozó-terület kezdete mutató alacsonyabb címre való állítása. Ezt az eljárást is bemutatjuk. Egy RESET és a monitor bekapcsolása után írjuk be ezt:

```
>0033 00 FC
>0037 00 FC
```

A BASIC-be való visszatérés, és a PRINT FRE(X) kiadása után megállapíthatjuk, hogy ugyanazt az eredményt kaptuk, mint az előbb. Viszont nem kellett beírnunk CLR-t, nem kellett egy byte-ot nullára állítani, tehát kevesebb a ráfordítás. Mindezt azzal értük el, hogy a BASIC a vége/karaktervältozó-terület eleje mutatót lejjebb állítottuk. Az ebben a fejezetben használt tárcímek egyébként a könyv H függelékében is megtalálhatók. A tárfoglaláshoz ennek a módszernek a másik előnye, hogy jóval nagyobb területet zárhatunk le a BASIC nyelvű programok elől, és hogy a gépi kódú programokat a BASIC programból gyakorlatilag problémamentesen betölthetjük.

Nos, ennyi kiegészítő ismeret után most már valóban nekiláthatunk a tervezett DATA sorokat előállító programunk megírásához.

21.6.4. A DATA sorokat előállító program betöltő programja

A fő programot a tár felső részén, a $FB00 címtől kezdődően helyezzük el. Mivel ez azonban az operációs rendszer rutinjait is használja, saját memórialapozó rutinokat is kell írni, amelyek a $065E címen kezdődnek. A program betöltésével kapcsolatos munka csökkentése érdekében ezen a területen egy betöltő program is van, amely a program második részének, a tulajdonképpeni DATA sorokat előállító programnak a betöltését végzi. A későbbiekben tehát csak az itt következő betöltő programot kell betöltenünk, amely indítás után automatikusan betölti a program második részét.

A betöltő program beírásával kapcsolatos megjegyzéseink:

A következőkben megadjuk a betöltő program BASIC és assembly nyelvű változatát is. Beíráshoz kényelmesebb a BASIC betöltő program használata. A beírás után és az első indítás előtt a biztonság kedvéért célszerű a programot háttértárolón tárolni. Ha a program a későbbi futtatáskor a HIBA A DATA SOROKBAN üzenetet írja ki, akkor a megadott vizsgáló összeg nem egyezik meg a DATA sorokban levő számok összegével. A beírt programot meg kell vizsgálni, és a hibát ki kell javítani. Az újabb indítás előtt megint vigyük ki a háttértárolóra.

A lemezmeghajtó tulajdonosok a következők szerint változtassák meg a 110-es sort:

```
110 DATA 85,38,85,34,A9,08,85,AE
```

Ennek megfelelően a vizsgáló összeg értékét is 12592-re kell átírni. A magnótulajdonosoknak a programon nem kell változtatniuk.

A BASIC betöltő:

```
10 FOR T=1630 TO 1744
20 READ A$: B=B+DEC(A$)
30 POKE T,DEC(A$)
40 NEXT
50 IF B<>12592 THEN PRINT"HIBA A DATA SOROKBAN"
100 DATA A9,00,85,37,85,33,A9,FB
110 DATA 85,38,85,34,A9,08,85,AE
120 DATA A9,01,85,AD,A0,0D,84,AB
130 DATA 88,A9,5E,85,AF,A9,02,85
140 DATA B0,B9,C4,06,91,AF,88,10
150 DATA F8,A9,00,85,9A,20,D5,FF
```

```
160 DATA 90,19,60,78,8D,3F,FF,20
170 DATA 00,FB,8D,3E,FF,58,60,8D
180 DATA 3E,FF,58,20,00,00,78,8D
190 DATA 3F,FF,60,A0,17,B9,91,06
200 DATA 99,5E,06,88,10,F7,60,A9
210 DATA 01,A0,00,91,2B,20,18,88
220 DATA 20,4B,88,4C,9A,8A,44,41
230 DATA 54,41,47,45,4E,45,52,41
240 DATA 54,4F,52
```

Az első, kifogástalan indítás után lépjünk be a monitorprogramba, és írjuk be:

```
S"BETOLTO",1,065E,06D1
```

Ez a sor a programot kazettán tárolja. A lemezmeghajtó tulajdonosok a fenti sorban levő 1-est írják át 8-asra. A program visszatöltése BASIC-ben:

```
LOAD"BETOLTO",1,1
```

vagy lemezmeghajtó esetén:

```
LOAD"BETOLTO",8,1
```

Térjünk rá most az assembler listára, amelyen keresztül elmagyarázzuk, hogy mit és miért programoztunk. A magyarázatokkal és a listával rutinról-rutinra haladunk. A magyarázatokban az operációs rendszer néhány rutinjának a használatához is adunk tippeket. Aki most még nem akar ezekkel foglalkozni, az nyugodtan ugorja át ezt a részt, és folytassa a program második részének beírásával.

A legtöbb sor tartalmaz egy pontosvesszővel elválasztott megjegyzést. Ez a későbbiekben is segíti a program gyors megértését. Ha valaki a programot a monitorprogram segítségével is megnézi, az ezeket a megjegyzéseket ott természetesen nem találja meg.

A betöltő program assembler listája

A BASIC tár vége mutató lejjebb állítása $FB00-ra

Amint említettük, a fő programunk a tárban a $FB00 címen fog kezdődni. Ezért a BASIC tár vége mutatót lejjebb kell állítanunk, nehogy a BASIC ezt a területet esetleg felülírja. Ehhez az AKKU-ba közvetlen címzéssel $00-t töltünk, majd a tartalmát beírjuk a $37 (BASIC-tár vége mutató alsó byte-ja) és a $33 (karakter-változó terület kezdete mutató alsó byte-ja) címekre. Most az AKKU-ba a felső byte-ot töltjük, és ezt az értéket beírjuk a $38 (BASIC-tár vége mutató felső byte-ja) és a $34 (karakter-változó terület kezdete mutató felső byte-ja) címekre.

A fő program betöltő rutinja

```
.065E A9 00     LDA #$00    ;AKKU-ba $00-t tolteni
.0660 85 37     STA $37     ;AKKU tartalmat a $37-es
.0662 85 33     STA $33     ;es a $33-as cimre irni
.0664 A9 FB     LDA #$FB    ;AKKU-ba $FB-t tolteni
.0666 85 38     STA $38     ;AKKU tartalmat a $38-as
.0668 85 34     STA $34     ;es a $34-es cimre irni
```

Ennek a rutinnak a feladata a fő program betöltésének az előkészítése. Az előkészítés után meghívja az operációs rendszer betöltő rutinját. Lássuk az ehhez szükséges lépéseket.

```
.066A A9 01     LDA #$01    ;egységszámot betölteni
.066C 85 AE     STA $AE     ;és $AE-n tárolni
.066E A9 01     LDA #$01    ;másodlagos címet betölteni
.0670 85 AD     STA $AD     ;és $AD-re írni
.0672 A0 00     LDY #$00    ;a programnév hosszát
.0674 84 AB     STY $AB     ;$AB-re írni
.0676 88        DEY         ;Y regisztert csökkenteni
.0677 A9 5E     LDA #$5E    ;a programnév címének
.0679 85 AF     STA $AF     ;alsó byte-ját az $AF címre,
.067B A9 02     LDA #$02    ;felső byte-ját a
.067D 85 B0     STA $B0     ;$B0 címre írni
.067F B9 C4 06  LDA $06C4,Y ;a program nevét az $AF
.0682 91 AF     STA ($AF),Y ;és a $B0 által megadott
.0684 88        DEY         ;címen tárolni
.0685 10 F8     BPL $067F   ;ha nem < 0, akkor újra $067F
.0687 A9 00     LDA #$00    ;AKKU-ba $00-t tölteni
.0689 85 9A     STA $9A     ;a betöltő rendszerüzenetek
                            ;elnyomása
.068B 20 D5 FF  JSR $FFD5   ;LOAD rutin meghívása
.068E 90 19     BCC $06A9   ;elágazás az eltoló rutinra
.0690 60        RTS         ;ha töltéskor hiba, akkor vége
```

Ahhoz, hogy a számítógép egy programot betöltsön, ismernie kell a program nevét, annak az egységnek az azonosító számát, amelyről a betöltés történik, valamint a másodlagos címet. Vannak esetek, amikor nem kell ezek mindegyikét megadni. Ilyenkor a számítógép saját maga ad egy megfelelő értéket. Mi a fő programunkat kazettás magnóról töltjük be. Ennek az egységnek az azonosító száma 1, és ezt a számítógép a $AE címen várja. A lemezegységgel dolgozóknak az egységszámra 8-at kell írniuk. Mivel a betöltendő program gépi kódú program, ezt abszolút módon, tehát egy megadott címtől kezdődően kell betölteni a tárba. A másodlagos cím tehát ugyancsak 1. A másodlagos címet a rendszer a $AD címen várja.

A töltő programunknak egy, meghatározott programot kell betöltenie. Ezért szükségünk van a program (file) nevére. Először beírjuk a név hosszát, azaz a nevet alkotó karakterek számát a $AB címre. Az operációs rendszernek azt is tudnia kell, hogy hol találja meg a program nevét. Erre a célra a számítógépnek van egy mutatója a $AF és $B0 címeken. Ezt a mutatót a $025E címre állítjuk.

Miután ezeket a paramétereket átadtuk, elindul egy rövid, eltoló rutin. Ez a kis rutin a betöltő rutinunk $067F–$0685 közötti részén található. Evvel a rutinnal a $06C4–$06D0 közötti címeken levő programnevet átmásoljuk a $025E címen kezdődő területre. Megtehettük volna azt is, hogy a programnév mutatót eltérítjük a $06C4 címre, ezzel azonban sem tárhelyet, sem időt nem nyertünk volna. A $065E-től kezdődő területet egyébként is célszerű saját, rövid rutinok részére szabadon tartani.

Az Y regiszter tartalma a dekrementálás (csökkentés) után $0C. Most a $067F címen levő utasítással egy értéket töltünk az AKKU-ba, abszolút címzésmódban, és az Y regiszterrel indexelve. Érthetőbben fogalmazva: a $06C4 abszolút címhez hozzáadjuk az Y regiszter tartalmát ($0C). Az AKKU-ba tehát a $06C4 + $0C = $06D0 címen levő tartalom kerül. Az AKKU tartalmát indirekt címzésmódban, az Y-nal indexelve helyezzük el a tárba. Esetünkben így kell a tárcímeket kiszámítani: az STA utasítás után ($AF),Y áll. A $AF és a rákövetkező $B0 címek egy mutatót alkotnak. A $AF tartalmazza a mutató alsó, a $B0 pedig a felső byte-ját. Ez a mutató most a mi esetünkben $025E. Ehhez az értékhez kell hozzáadnunk az Y regiszter tartalmát ($0C). Az utasítás tehát az AKKU tartalmát a $025E + $0C = $026A tárcímre írja be.

Miután az eltolás felülről lefelé történik, az Y regisztert dekrementálni, azaz az értékét csökkenteni kell. A regiszter tartalma most $0B lesz. A következő utasítás BPL $067F, ami azt jelenti, hogy a program végrehajtása ágazzon el, ha az utolsó művelet eredménye pozitív volt. A pozitívat itt matematikai értelemben használjuk (+). Amint bizonyára tudjuk, az állapotregiszterben a negatív (N) bit annak a számnak a hetedik bitjét tartalmazza, amellyel a processzor éppen dolgozik, vagy dolgozott. Ha ez a bit magas, akkor ez a processzor számára azt jelenti, hogy az utoljára elvégzett művelet eredménye negatív (−) volt. A programunkban az utolsó művelet az volt, hogy az Y regiszter tartalma $0B-re csökkent. A hetedik bit tehát alacsony, az eredmény pozitív, a feltétel (elágazás, ha pozitív) teljesül, és így a program végrehajtása visszaugrik a $067F címre.

A $067F címen levő utasítás most az AKKU-ba a $06CF ($06C4 + $0B) tartalmát tölti be, a rákövetkező utasítás pedig ezt az értéket a $0269 ($025E + $0B) tárcímre írja be. Az Y regiszter ismét dekrementálódik. A kapott érték ($0A) a processzor számára ismét pozitív, tehát ugrás újra a $067F-re. Kezdődik elölről a játék.

Ez a ciklus mindaddig ismétlődik, míg az Y regiszter értéke $00 nem lesz, és ismét dekrementálódik. Ekkor az értéke $FF-lesz, a hetedik bit, és ezáltal az állapotregiszter negatív bitje is magas lesz, és a BPL... feltétel már nem teljesül. A program végrehajtása a $0687-es címen levő utasítás feldolgozásával folytatódik.

A betöltő rutinunk utolsó részéhez értünk. Az AKKU-ba közvetlen módon $00-t töltünk, és ezt az értéket beírjuk a $9A címre. Ezzel letiltjuk az olyan rendszerüzenetek kiírását, mint pl. SEARCHING FOR... stb. Ha adatmagnóval dolgozunk, akkor a betöltéshez a Commodore billentyűt sem kell lenyomni. Az AKKU tartalmának azonban más ok miatt is $00-nak kell lennie. Ha ugyanis nem 0, akkor a $FFD5-ön kezdődő KERNAL LOAD rutin a betöltés helyett a VERIFY (tártartalmak összehasonlítása) műveletet végzi el. A programunk meghívja ezt a rutint. Ha a DATA-előállító program betöltése utáni visszatérésekor az átvitelbit (ami a jelen esetben a hibajelző bit) nem magas, akkor a programunk a $06A9 címre ágazik. Ha az átvitelbit magas, akkor a betöltés során hiba történt.

Elérkeztünk a memórialapozó rutinjainkhoz, amelyeket szintén a betöltő rutin tartalmaz. A most még a $0691–$06A8 közötti címeken levő mindkét memórialapozó rutint később átmásoljuk a $065E-vel kezdődő területre. Ez a terület ugyanis a fő program betöltése után ismét a rendelkezésünkre áll, hiszen a betöltő programra már nincs szükségünk. Ez azzal az előnnyel is jár, hogy maga a DATA-előállító program is az 1630-as SYS-címmel indítható.

1. számú memórialapozó rutin

```
.0691 78        SEI          ;megszakitas letiltasa
.0692 8D 3F FF  STA $FF3F    ;RAM bekapcsolasa
.0695 20 00 FB  JSR $FB00    ;ugras a DATA-programra
.0698 8D 3E FF  STA $FF3E    ;ROM bekapcsolasa
.069B 58        CLI          ;megszakitas engedelyezese
.069C 60        RTS          ;vissza a BASIC-be
```

Lássuk az első rutint. Miután letiltottuk a megszakításokat, átkapcsolunk a RAM-ba. Így tehát a processzor a mi programunkhoz férhet hozzá. A fő programunk a tárban a $FB00 címen kezdődik, amelyre mint egy alprogramra, ráugrunk. A fő program RTS-sel fejeződik be, tehát a $FB00 utáni tárcímre ugrik vissza. Az itt álló utasítás ismét a ROM-ot kapcsolja vissza. Ezt azért kell megtenni, hogy a processzor ismét

használhassa az interpretert és az operációs rendszert. Ismét engedélyezzük a megszakításokat, és így a billentyűzet lekérdezését is. Az ezután következő RTS-sel visszatérünk a BASIC-hez.

2. számú memórialapozó rutin

```
.069D 8D 3E FF   STA $FF3E   ;ROM bekapcsolasa
.06A0 58         CLI         ;megszakitas engedelyezese
.06A1 20 00 00   JSR $0000   ;lasd a szovegben
.06A4 78         SEI         ;megszakitas letiltasa
.06A5 8D 3F FF   STA $FF3F   ;RAM bekapcsolasa
.06A8 60         RTS         ;vissza a foprogramhoz
```

A SYS utasítással, amellyel a későbbiekben a programot majd indítjuk, a kezdőcím mellett még további paramétereket is átadunk. Mivel ezeket a paramétereket az operációs rendszer meglevő rutinjain keresztül vesszük át, erre a második memórialapozó rutinra is szükség van.

Külön felhívjuk a figyelmet a JSR $0000 utasításra, amely most a $06A1 címen, a későbbiekben pedig a $066E címen fog állni. A JSR utasítást követő címet önkényesen választottuk meg. Ezt a fő program a követelményeknek megfelelően fogja megváltoztatni. A processzorunknak nincs lehetősége arra, hogy közvetett módon hívhasson meg alprogramokat. Mivel azonban az operációs rendszer meghívása előtt át kell kapcsolnunk a ROM-ba, kieszeltünk egy olyan megoldást, amely lehetővé teszi a processzor számára, hogy az átkapcsolás után az operációs rendszer kívánt címére ugorhasson.

E rutin meghívása előtt a fő programunk a JSR után álló két byte-ot úgy változtatja meg, hogy azok a mi ugrási címünket tartalmazzák. Így lehetővé válik, hogy RAM-ból is használhassuk a ROM rutinjait. A többi utasítás bizonyára érthető már, így ezek magyarázatától eltekintünk.

A memórialapozó rutinok eltoló rutinja

```
.06A9 A0 17      LDY #$17    ;a byte-ok szama = $18
.06AB B9 91 06   LDA $0691,Y ;$0691+Y tartalmat betolteni
.06AE 99 5E 06   STA $065E,Y ;es $065E+Y cimre beirni
.06B1 88         DEY         ;Y regisztert dekrementalni
.06B2 10 F7      BPL $06AB   ;ha nincs keszen, akkor $06AB
.06B4 60         RTS         ;a betolto program vege
```

Ezekhez a sorokhoz is fűzünk némi magyarázatot. Ez a rutin eltolja (vagy másként fogalmazva átmásolja) a memórialapozó rutinjainkat a $0691 kezdőcímű területről a $065E kezdőcímű területre. Az átmásolandó terület hossza $18 byte. Az Y regisztert most is indexregiszterként használjuk egyrészt a számláláshoz, másrészt a címzésmódhoz. Mivel a ciklus azt követően, hogy az Y értéke $00 lesz, még egyszer lefut, a byte-ok számlálása $17-tel kezdődik ($17 + $01 = $18). Az AKKU-ba való betöltés és az AKKU tartalmának a tárba írása abszolút címzéssel, utólagos indexeléssel történik. A címzésmód részletes ismertetését a könyv idevonatkozó fejezete tartalmazza.

Az eltolás végeztével a program az RTS-sel visszatér a BASIC-hez.

A töltőprogrammal azonban nem csak a fő programot töltjük be.

OLD program

```
.06B5 A9 01     LDA #$01     ;AKKU-ba $01-et tolteni
.06B7 A0 00     LDY #$00     ;Y regiszterbe $00-t tolteni
.06B9 91 2B     STA ($2B),Y  ;tarolas, indirekt indexelt
                             ;cimzes
.06B8 20 18 88  JSR $8818    ;uj csatolocim kiszamitasa
.06BE 20 4B 88  JSR $884B    ;valtozokezdet ujraszamitasa
.06C1 4C 9A 8A  JMP $8A9A    ;CLR vegehajtasa
```

Az OLD program működését a korábbiakban már ismertettük. Itt csak arra térünk ki, hogy miért van rá szükség most. Mindannyiszor, amikor BASIC-ből abszolút módon, tehát adott kezdőcímtől, 1-es másodlagos címmel töltünk be egy programot, a betöltést követően fontos mutatók elállítódnak. Ezeket a NEW utasítással lehet ismét helyrehozni. Azért, hogy szükség esetén ezt a programot utólag is be lehessen tölteni, a program végéhez hozzáfűztük az OLD rutint. Ha tehát egy program betöltése után kiadjuk a NEW utasítást, akkor a SYS 1717 utasítással meghívhatjuk az OLD rutint, és a BASIC program ismét a tárban van. Ezután dolgozhatunk tovább BASIC-ben, vagy a SYS 1630-cal meghívhatjuk a DATA-előállító programot.

Már csak a fő programunk neve hiányzik.

```
M 06C4 06D0

>06C4 44 41 54 41 47 45 4E 45   :DATAGENE
>06CC 52 41 54 4F 52            :RATOR
```

A név a betöltő program végén áll, és természetesen a programmal együtt ezt is betöltjük.

A betöltő program ismertetésének végére értünk. Reméljük, hogy a gondolatmenetünket nem volt nehéz követni. A biztos tudás azonban csak önálló programozással szerezhető meg. Ez a betöltő program is nyilván még tovább csiszolható. Lehet, hogy valakinek már van is ötlete. Itt is érvényes a mondás, hogy gyakorlat teszi a mestert.

Akkor most térjünk rá a tulajdonképpeni DATA-előállító programra.

21.6.5. A DATA-előállító program

A DATA-előállító, fő programunkhoz is írtunk egy BASIC nyelvű betöltőt. Magukat a DATA-sorokat már ez a program állította elő. Ugye nem rossz? Írjuk be tehát először ezt a betöltőt, tároljuk, majd kapcsoljuk be a monitort. Most ez következik:

```
>07F8 80
```

Ezzel az utasítással a monitort átkapcsoltuk a RAM-hoz való hozzáférésre.

```
S"DATAGENERATOR",1,FB00,FBFC
```

Azért, hogy a program újbóli betöltésekor elkerüljük a hosszabb várakozási időt, a magnóval dolgozók ügyeljenek arra, hogy ez a program közvetlenül a betöltő program utáni kazettarészre kerüljön. A programot a DATAGENERATOR néven kell tárolni, ha ettől eltérő nevet választunk, akkor a megfelelő változtatást a betöltő programban is el kell végezni.

Ennyi bevezető után már csak kitartás kell a program beírásához.

```
10 FOR I=DEC("FB00") TO DEC("FBFB")
20 READ A$
30 POKE I,DEC(A$)
40 B=B+DEC(A$)
50 NEXT
60 IF B<>31431 THEN PRINT"HIBA A DATA SOROKBAN"
70 :
100 DATA A2,06,8A,48,A9,91,8D,6F,06
110 DATA A9,94,8D,70,06,20,6A,06,A9
120 DATA E1,8D,6F,06,A9,9D,8D,70,06
130 DATA 20,6A,06,68,AA,A5,15,95,D0
140 DATA CA,94,D0,CA,D0,D8,D8,38,A5
150 DATA D6,48,A5,D3,E5,D5,85,D3,60
160 DATA 05,E6,D6,F0,07,38,A5,D4,E5
170 DATA D6,B0,11,A2,0E,A9,83,8D,6F
180 DATA 06,A9,86,8D,70,06,20,6A,06
190 DATA 68,60,85,D4,68,85,D6,A5,2D
200 DATA E9,02,B0,02,C6,2E,85,2D,A0
210 DATA 02,A5,D1,91,2D,C8,A5,D2,91
220 DATA 2D,C8,A9,83,91,2D,C8,A9,20
230 DATA 91,2D,C8,A2,00,A1,D5,48,29
240 DATA F0,4A,4A,4A,4A,AA,BD,EC,FB
250 DATA 91,2D,C8,68,29,0F,AA,BD,EC
260 DATA FB,91,2D,C6,D3,D0,06,A5,D4
270 DATA F0,15,C6,D4,48,E6,D5,D0,02
280 DATA E6,D6,C0,1F,F0,09,C8,A9,2C
290 DATA 91,2D,68,D0,C7,48,C8,A9,00
300 DATA 91,2D,C8,98,48,A0,00,18,65
310 DATA 2D,91,2D,C8,A6,2E,90,01,E8
320 DATA 8A,91,2D,E6,D1,D0,02,E6,D2
330 DATA 68,18,65,2D,85,2D,90,02,E6
340 DATA 2E,68,D0,86,A0,00,A2,02,91
350 DATA 2D,E6,2D,D0,02,E6,2E,CA,D0
360 DATA F5,60,30,31,32,33,34,35,36
370 DATA 37,38,39,41,42,43,44,45,46
```

Ha a beírás után a BASIC betöltő kifogástalanul működik, és az előzőekben leírt lépéseket is elvégeztük, akkor nincs akadálya, hogy ki is próbáljuk. Az itt következő magyarázatokhoz hagyjuk a tárban a BASIC betöltőt.

Töltsük be a betöltő programot a következő utasítássorral:

LOAD"BETOLTO",1,1 ha kazettás magnóval dolgozunk, és
LOAD"BETOLTO",8,1 ha lemezmeghajtóval dolgozunk.

Indítsuk el a betöltő programot:

SYS 1630

A PRESS PLAY ON TAPE üzenet megjelenése, és a felszólítás végrehajtása után a gép elkezdi keresni a kazettán a fő programot, majd betölti. Ha azoknál is megjelenik ez az üzenet, akik lemezmeghajtóval dolgoznak, akkor ők elfelejtették a BASIC betöltőben az egységszámot megváltoztatni. Nézzenek utána a betöltő program leírásánál, és végezzék el a módosítást. A betöltés után ez következik:

NEW

majd SYS 1717. Most a LIST kiadására ismét meg kell jelenni a képernyőn a fő program BASIC betöltőjének. És most értünk ahhoz a ponthoz, amikor kiderül, hogy milyen eredménnyel dolgoztunk:

```
SYS 1630,DEC("FB00") ,DEC("FBFC") ,1000
```

Mivel a sorelőállító programunk néhány mutatót elállított, következzék még:

```
CLR
```

Ha ezután a Plus/4-es a READY üzenettel jelentkezik vissza, az már nem rossz. Ha viszont egy LIST után 1000-es sorszámmal kezdődően megjelennek ugyanazok a DATA sorok, amelyek a 100-as sorszámtól is megvannak, akkor ez egyenesen csodálatos. A DATAGENERATOR működik.

A **SYS** utasításunk **szintaxisa** tehát:

SYS 1630,-tól,-ig +1, első sorszám

Ha valamelyik paramétert hibásan adjuk meg, vagy ha kihagyunk egy vesszőt, akkor a SYNTAX ERROR hibaüzenetet kapjuk.

Következzék ennek a programnak a magyarázata.

(*Megjegyzés:* az egyszerűség kedvéért a továbbiakban az alsó byte helyett LB-t (= LOW BYTE), a felső byte helyett HB-t (= HIGH BYTE) írunk.)

A programunknak szüksége van néhány átmeneti tárhelyre. Ezek címe és tartalma:

cím	*tartalom*
$D1	sorszám LB
$D2	sorszám HB
$D3	VÉGE+1 LB, majd később DBSZÁM LB
$D4	VÉGE+1 HB, majd később DBSZÁM HB
$D5	KEZDET LB
$D6	KEZDET HB

```
.FB00 A2 06     LDX #$06     ;X regiszterbe $06-ot tolteni
.FB02 8A        TXA          ;X-et AKKU-ba vinni
.FB03 48        PHA          ;AKKU-t a verembe tolteni
.FB04 A9 91     LDA #$91     ;ugrasi cim LB-t
.FB06 8D 6F 06  STA $066F    ;$066F-re irni
.FB09 A9 94     LDA #$94     ;ugrasi cim HB-t
.FB0B 8D 70 06  STA $0670    ;$0670-re irni
.FB0E 20 6A 06  JSR $066A    ;2.sz.memorialapozo rutin
                             hivasa
.FB11 A9 E1     LDA #$E1     ;ugrasi cim LB-t
.FB13 8D 6F 06  STA $066F    ;$066F-re irni
.FB16 A9 9D     LDA #$9D     ;ugrasi cim HB-t
.FB18 8D 70 06  STA $0670    ;$0670-re irni
.FB1B 20 6A 06  JSR $066A    ;2.sz.memorialapozo rutin
                             hivasa
.FB1E 68        PLA          ;AKKU-t verembol kivenni
.FB1F AA        TAX          ;AKKU-t X regiszterbe tolteni
.FB20 A5 15     LDA $15      ;AKKU-ba $15 tartalmat tolteni
.FB22 95 D0     STA $D0,X    ;es a $D0+X tarcimre beirni
```

```
.FB24 CA          DEX           ;X reg.-t dekrementalni
.FB25 94 D0       STY $D0,X     ;Y tartalmat $D0+X tarcimre
                                 irni
.FB27 CA          DEX           ;X reg.-t dekrementalni
.FB28 D0 D8       BNE $FB02     ;ha <> 0, akkor elagazas
```

Ebben a programrészben az X regisztert számlálóként is, és címregiszterként is használjuk. A táblázatunkból látható, hogy a $D1–$D6 területre hat értéket kell átadni. Ezért a program elején $06-ot töltünk az X regiszterbe. A $FB04–$FB0B közötti címeken a memórialapozó rutinunk JSR utasítását (JSR $0000) manipuláljuk. A műveletet követően a $066E címen JSR $9491 fog állni. Ezután meghívjuk a 2. számú memórialapozó rutinunkat, amely a ROM-ba való átkapcsolás után a JSR $9491-gyel meghív egy rutint az operációs rendszerből. Ez a rutin a vessző vizsgálatát végzi. A vizsgálat után visszatér a memórialapozó rutinhoz, amely visszakapcsol a RAM-ba, és onnan visszaugrik a programunk $FB11-es címére.

Ismét manipuláljuk a $066F–$0670 közötti címeket, mégpedig ezáltal úgy, hogy az ugró utasítás JSR $9DE1 lesz. Ismét meghívjuk a memórialapozó rutinunkat, ahonnan a program az operációs rendszer $9DE1 címén kezdődő rutinra ugrik. Ez a vessző után álló kifejezést értékeli ki. A rutin a kapott kétbyte-os értéket az Y regiszterbe és a $15 tárcímre írja. Ez a rutin tehát 65535-nél nagyobb számot nem tud feldolgozni. Ha a szám ennél nagyobb, akkor a program hibaüzenettel leáll.

Programunkba való visszatérés után a kapott két byte-ot nulláslapos, abszolút címzésmódban, az X regiszterrel indexelve beírjuk a megfelelő tárcímekre. Ehhez az X regisztert kétszer dekrementáljuk. Ha az X regiszterben levő érték eléri a $00-t, akkor a program az elágazó utasítást (BNE) követő utasítás feldolgozásával folytatódik. Ha viszont az eredmény még nem nulla, akkor visszaugrik a $FB02 címre, és kiértékeli a következő kifejezést.

Darabszám kiszámítása, esetleges hibaüzenet kiírása

```
.FB2A D8          CLD           ;decimalis jelzobitet torolni
.FB2B 38          SEC           ;atvitelbitet magasra allitani
.FB2C A5 D6       LDA $D6       ;KEZDET HB-t AKKU-ba tolteni
.FB2E 48          PHA           ;es AKKUT-t a verembe menteni
.FB2F A5 D3       LDA $D3       ;VEGE+1 LB-t AKKU-ba tolteni
.FB31 E5 D5       SBC $D5       ;AKKU minusz KEZDET LB
.FB33 85 D3       STA $D3       ;DBSZAM LB-t $D3-ra irni
.FB35 B0 05       BCS $FB3C     ;ha nincs kolcsonvetel, akkor
                                 ;$FBC3
.FB37 E6 D6       INC $D6       ;egyebkent KEZDET HB-t
                                 ;inkrementalni
.FB39 F0 07       BEQ $FB42     ;ha tulcsordulas, akkor $FB42
.FB3B 38          SEC           ;atvitelbitet ismet magasra
                                 ;allitani
.FB3C A5 D4       LDA $D4       ;VEGE+1 HB-t AKKU-ba tolteni
.FB3E E5 D6       SBC $D6       ;AKKU minusz KEZDET HB
.FB40 B0 11       BCS $FB53     ;ha nincs kolcsonvetel,  akkor
                                 ;$FB53
.FB42 A2 0E       LDX #$0E      ;X reg.-be hiba kodszamat beirni
.FB44 A9 83       LDA #$83      ;AKKU-ba hibarutin LB-t tolteni
.FB46 8D 6F 06    STA $066F     ;JSR utasitas LB-be beirni
.FB49 A9 86       LDA #$86      ;AKKU-ba hibarutin HB-t tolteni
.FB4B 8D 70 06    STA $0670     ;JSR utasitas HB-be beirni
```

```
.FB4E 20 6A 06    JSR $066A     ;ugras a memorialapozo rutinra
.FB51 68          PLA           ;verem tetejet AKKU-ba  tolteni
.FB52 60          RTS           ;program megszakitasa
.FB53 85 D4       STA $D4       ;DBSZAM HB-t $D4-re irni
.FB55 68          PLA           ;verem tetejet AKKU-ba tolteni
.FB56 85 $D6      STA $D6       ;regi KEZDET HB-t ismet $D6-ra
                                ;irni
```

Ebben a programrészben kiszámítjuk az előállítandó DATA-k darabszámát. Ha a KEZDET-et nagyobbra választjuk mint a VÉGE + 1-et, akkor az X regiszterbe betöltődik a $0E hibaazonosító kódszám. A memórialapozó rutinunk meghívja az operációs rendszer hibaüzenet kiíró rutinját (a címe $8683). A képernyőn megjelenik az ILLEGAL QUANTITY ERROR hibaüzenet, a verem kiürül, és a program futása megszakad.

Ha viszont az eredmény megfelelő, akkor a program a $FB53-as címen folytatódik. Mivel lehetséges, hogy a $D6 tartalmát előzőleg növelni kellett ($FB37), a KEZDET HB régi értékét vissza kell írni a $D6-ra. Az adatok darabszámát most a $D3 és a $D4 tartalmazza. A VÉGE + 1 értékre már nincs szükségünk.

Figyeljünk arra, hogy a hibarutin meghívásakor a hiba kódszámának mindig az X regiszterben kell lennie. A lehetséges hibaüzeneteket és a hozzájuk tartozó kódszámokat a függelék tartalmazza.

Az UTÁN nevű mutató létrehozása

```
.FB58 A5 2D       LDA $2D       ;valtozoterulet kezdete mutato
.FB5A E9 02       SBC #$02      ;LB minusz $02
.FB5C B0 02       BCS $FB60     ;ha nincs kolcsonvetel, akkor
                                ;$FB60
.FB5E C6 2E       DEC $2E       ;egyebkent HB-t dekrementalni
.FB60 85 2D       STA $2D       ;$2D-re az uj LB-t beirni
```

Miután az előkészítő lépéseket megtettük, most rátérhetünk a program lényegi részére. Mivel a programnak tudnia kell, hogy a tár melyik részére helyezze el a sorszámokat, a csatolócímeket és a DATA-kat, létrehozunk egy mutatót, amelyet UTÁN névre keresztelünk el. Ehhez a fenti rutinban eltérítettük a változóterület kezdete mutatót. Ez az UTÁN nevű mutató most a tárban levő utolsó BASIC sor végét jelölő $00 byte utáni első byte-ra mutat. Ez a húzás lehetővé teszi, hogy a DATAGENERATOR-t akkor is elindíthassuk, ha a tárban már van BASIC program.

Sorszám előállítása

```
.FB62 A0 02       LDY #$02      ;Y reg.-be kezdoerteket tolteni
.FB64 A5 D1       LDA $D1       ;AKKU-ba sorszam LB-t tolteni
.FB66 91 2D       STA ($2D),Y   ;es UTAN+Y tarcimre beirni
.FB68 C8          INY           ;Y-t inkrementalni
.FB69 A5 D2       LDA $D2       ;AKKU-ba sorszam HB-t tolteni
.FB6B 91 2D       STA ($2D),Y   ;es UTAN+Y tarcimre beirni
```

Az előállítandó BASIC sor első és második byte-ját egyelőre üresen hagyjuk. A harmadik és a negyedik byte-ba beírjuk a sorszám alsó és felső byte-ját (LB, HB) úgy, hogy az UTÁN mutatóhoz hozzáadjuk az Y regiszter tartalmát (indirekt indexelt címzésmód).

DATA token beírása

```
.FB6D C8          INY           ;Y-t 1-gyel novelni
.FB6E A9 83       LDA #$83      ;AKKU-ba DATA tokenjet beirni
.FB70 91 2D       STA ($2D),Y ;es UTAN+Y tarcimre beirni
```

A DATA tokenjét beírjuk a sor ötödik byte-jába.

Szóköz beírása

```
.FB72 C8          INY           ;Y-t 1-gyel novelni
.FB73 A9 20       LDA #$20      ;szokoz CHR$ kodja
.FB75 91 2D       STA ($2D),Y ;UTAN+Y tarcimre beirni
```

Azért, hogy az előállítandó sor áttekinthetőbb legyen, a DATA és az első DATA-elem közé beírunk egy szóközt.

DATA-elemek előállítása és tárolása

```
.FB77 C8          INY           ;Y reg.-t  inkrementalni
.FB78 A2 00       LDX #$00      ;X reg.-be $00-t tolteni
.FB7A A1 05       LDA ($05,X)   ;KEZDET+X cim tartalmat
                                 betolteni
.FB7C 48          PHA           ;AKKU-t a verembe menteni
.FB7D 29 F0       AND #$F0      ;0-3.biteket torolni
.FB7F 4A          LSR           ;4-7.biteket a
.FB80 4A          LSR           ;0-3.bitek
.FB81 4A          LSR           ;helyere
.FB82 4A          LSR           ;eltolni
.FB83 AA          TAX           ;eredmenyt X regiszterbe vinni
.FB84 BD EC FB    LDA $FBEC,X ;es a megfelelo erteket
                                 betolteni
.FB87 91 2D       STA ($2D),Y ;es a programsorba beirni
.FB89 C8          INY           ;Y-t inkrementalni
.FB8A 68          PLA           ;verembe mentett erteket
                                 visszahozni
.FB8B 29 0F       AND #$0F      ;4-7.biteket torolni
.FB8D AA          TAX           ;eredmenyt X regiszterbe vinni
.FB8E BD EC FB    LDA $FBEC,X ;DATA masodik karakteret
                                 beolvasni
.FB91 91 2D       STA ($2D),Y ;es a programsorba beirni
```

Mivel a processzor indirekt betöltő utasítást nem tud végrehajtani, ezért az X regisztert nullára kell állítani, és ezután lehet az AKKU-ba indexelt, indirekt címzéssel értéket tölteni. Az AKKU-ba tehát azon a tárcímen levő érték kerül, amely címre a KEZDET mutató mutat.

Az egyes DATA-elemek a későbbiekben két karakterből fognak állni (hexadecimális számábrázolás). Az AKKU tartalmát az első karakter előállításához át kell alakítani, de később, a második karakter előállításához még szükségünk lesz rá, ezért beírjuk a verembe. Most az AKKU és a $F0 között elvégezzük az AND logikai műveletet, azaz a 0–3. biteket töröljük. Az így kapott eredmény négyszeri jobbra tolásával a számot 16-tal elosztjuk. A kapott hányados egy mutatóul szolgál, amely

a $FBEC címen kezdődő ASCII-táblázatunknak arra az elemére mutat, amely a DATA-sorunk első elemének első karaktere lesz. Ezt a mutatót most átvisszük az X regiszterbe, a táblázatból az X-ik elemet ($FBEC+X) betöltjük az AKKU-ba, majd az AKKU tartalmát beírjuk a programsor megfelelő helyére.

Ahhoz, hogy megkapjuk azt a mutatót, amely a DATA-elem második karakterére mutat, a verem tetején levő byte-ot visszahozzuk az AKKU-ba, az AND $0F művelettel töröljük a 4–7. biteket, és az eredményt átvisszük az X regiszterbe. Az ezután következő műveletek pontosan megegyeznek az előbb leírtakkal.

Az ASCII-táblázatunk:

```
M FBEC FBFB

>FBEC 30 31 32 33 34 35 36 37   :01234567
>FBF4 38 39 41 42 43 44 45 46   :89ABCDEF
```

Folytassuk a munkát:

DBSZÁM dekrementálása

```
.FB93 C6 D3       DEC $D3       ;DBSZAM LB-t dekrementalni
.FB95 D0 06       BNE $FB9D     ;ha nincs tulcsordulas, akkor
                                 $FB9D
.FB97 A5 D4       LDA $D4       ;DBSZAM HB-t betolteni
.FB99 F0 15       BEQ $FBB0     ;ha ez is nulla, akkor vege
.FB9B C6 D4       DEC $D4       ;egyebkent DBSZAM HB-t
                                 dekrementalni
```

Ebben a programrészben a még előállítandó DATA-k darabszámát 1-gyel csökkentjük. A rutinból az is látható, hogy a VÉGE helyett miért a VÉGE+1 értéket kellett megadni.

A KEZDET mutató inkrementálása

```
.FB9D 48          PHA           ;AKKU-t a verembe menteni
.FB9E E6 D5       INC $D5       ;KEZDET LB-t inkrementalni
.FBA0 D0 02       BNE $FBA4     ;ha nincs tulcsordulas, akkor
                                 $FBA4
.FBA2 E6 D6       INC $D6       ;egyebkent KEZDET HB-t
                                 inkrementalni
```

Először az AKKU-t a verembe mentjük. Ennek a tartalma a későbbiekben egy feltétel vizsgálatára fog szolgálni. Ezután megnöveljük a KEZDET-et.

Az Y regiszter maximális értékének vizsgálata

```
.FBA4 C0 1F       CPY #$1F      ;Y tartalma = $1F ?
.FBA6 F0 09       BEQ $FBB1     ;ha igen, akkor $FBB1
.FBA8 C8          INY           ;Y-t inkrementalni
.FBA9 A9 2C       LDA #$2C      ;a vesszo CHR$ kodja
.FBAB 91 2D       STA ($2D),Y   ;a sorba beirni
.FBAD 68          PLA           ;verembol AKKU-t visszahozni
.FBAE D0 C7       BNE $FB77     ;mindig, mert sohasem 0
```

Miért pont $1F? Számoljunk csak utána, mégpedig – mert ezt azért jobban megszoktuk – ezúttal decimális számrendszerben.

Mindegyik DATA sor 9 elemet fog tartalmazni. Mindegyik elem két értékből áll. Ez eddig összesen 18 byte. Az egyes elemeket egymástól vessző választja el, 9 elem esetén ez 8 vesszőt jelent, ez már összesen 26 byte. Ehhez jön még egy szóköz, a DATA egybyte-os tokenje, továbbá a csatolócím és a sorszám két-két byte-ja. Egy sorhoz tehát 32 byte-ra van szükség (a sorvég byte-ot most nem számítva). Mivel az Y regiszter értékét $00-ról indítjuk, a maximális értékre 31, azaz $1F adódik.

Ha egy sorba több vagy kevesebb DATA-elemet akarunk írni, akkor ezt az értéket kell – hármas lépésekben – növelni vagy csökkenteni. Mivel a „beírás"-kor az interpreter ki van kapcsolva, 88 karakternél hosszabb sorok is előállíthatók.

Menjünk tovább:

Ha az Y regiszter tartalma kisebb, akkor egy vesszőt írunk a BASIC sorba. Most a verem tetején levő byte-ot betöltjük az AKKU-ba. Ennek a byte-nak az értéke – amikor a program erre a helyre ér – sohasem nulla. Ha megnézzük az előző, a DATA-elemeket előállító rutinunk utolsó két utasítását, akkor ez rögtön világossá válik. A program tehát visszatér erre a rutinra.

A programsort lezáró nulla beírása

```
.FBB0 48        PHA          ;AKKU-t verembe tolteni
.FBB1 C8        INY          ;Y-t inkrementalni
.FBB2 A9 00     LDA #$00     ;sorveg $00 byte-ot
.FBB4 91 2D     STA ($2D),Y  ;UTÁN+Y tarcimre beirni
```

Ha a program végére érünk, akkor még a vizsgálóbyte-ként szolgáló AKKU-t a verembe kell írni. Ha csak egy programsor végét értük el, akkor a rutinba a $FBB1 címen ugrunk be. Mindkét esetben beírjuk a tárba a sorvéget jelölő $00-t.

Csatolócím kiszámítása és beírása a sor elejére

```
.FBB6 C8        INY          ;Y-t inkrementalni
.FBB7 98        TYA          ;Y tartalmat AKKU-n keresztul
.FBB8 48        PHA          ;a verembe menteni
.FBB9 A0 00     LDY #$00     ;Y-ba $00-t irni
.FBBB 18        CLC          ;atvitelbitet torolni
.FBBC 65 2D     ADC $2D      ;AKKU+$2D tartalma
.FBBE 91 2D     STA ($2D),Y  ;eredmeny = csatolocim LB
.FBC0 C8        INY          ;Y-t inkrementalni
.FBC1 A6 2E     LDX $2E      ;X-be $2E tartalmat tolteni
.FBC3 90 01     BCC $FBC6    ;ha nincs atvitel,akkor $FBC6
.FBC5 E8        INX          ;egyebkent X-t inkrementalni
.FBC6 8A        TXA          ;X-t AKKU-ba vinni
.FBC7 91 2D     STA ($2D),Y  ;csatolocim HB
```

Akár a program, akár egy sor végére érünk, a csatolócímet mindegyik esetben ki kell számítani. A számítás eredményét a sor első két byte-jába kell beírni.

Azért, hogy ezt mindkét esetben megtehessük, a rutinunk egy kicsit hosszabbra sikeredett. Lehet, hogy van valakinek ennél jobb megoldása is? Nézzük azonban tovább, hogy lássuk, miért kellett az X regisztert is használnunk.

A sorszám és az UTÁN mutató növelése

```
.FBC9 E6 D1       INC $D1      ;sorszam LB-t megnovelni
.FBCB D0 02       BNE $FBCF    ;ha nincs tulcsordulas,.akkor
                                $FBCF
.FBCD E6 D2       INC $D2      ;egyebkent sorszam HB-t
                                megnovelni
.FBCF 68          PLA          ;Y elozo tartalmat verembol
                                elovenni
.FBD0 18          CLC          ;atvitelbitet torolni
.FBD1 65 2D       ADC $2D      ;AKKU+$2D tartalma
.FBD3 85 2D       STA $2D      ;eredmenyt UTÁN LB-be irni
.FBD5 90 02       BCC $FBD9    ;ha nincs atvitel, akkor $FBD9
.FBD7 E6 2E       INC $2E      ;egyebkent $2E-t megnovelni
.FBD9 68          PLA          ;vizsgalobyte-ot verembol
                                elovenni
.FBDA D0 86       BNE $FB62    ;nincs vege, uj sor
```

Ebben a programrészben először megnöveljük a sorszámot. Erre akkor is sor kerül, ha a program már befejezte a DATA-elemek előállítását. Most az UTÁN mutatót még az aktuális helyzetnek megfelelően be kell állítani, mivel egy új sor előállításához az Y regiszterbe ismét a $02 kezdőértéket kell beírni. A PLA utasítással kivesszük a veremből a vizsgálóbyte-unkat. Ha ez nulla, akkor a program a következő rutin feldolgozása után befejeződik. Ellenkező esetben visszatér a sorelőállító rutin elejére.

A program vége

```
.FBDC A0 00       LDY #$00     ;Y-ba $00-t tolteni
.FBDE A2 02       LDX #$02     ;X-be $00-t tolteni
.FBE0 91 2D       STA ($2D),Y  ;2.nullat beirni
.FBE2 E6 2D       INC $2D      ;UTÁN mutatot megnovelni
.FBE4 D0 02       BNE $FBE8    ;ha nincs tulcsordulas, akkor
                                $FBE8
.FBE6 E6 2E       INC $2E      ;egyebkent UTÁN HB-t novelni
.FBE8 CA          DEX          ;X-et dekrementalni
.FBE9 B0 F5       BNE $FBE0    ;ha nem zerus, 3.nullat
                                eloallitani
.FBEB 60          RTS          ;DATAGENERATOR program vege
```

Erre a részre a program csak a befejeződéskor tér rá. Mivel az utolsó programsort (beleértve a sorvég $00-t is) mindig két $00 byte-nak kell követnie, ezeket a nullákat itt állítjuk elő. Az X regiszter itt a ciklusszámláló szerepét tölti be. A ciklusban az UTÁN mutatót is visszaállítjuk a változóterület kezdetére. Ez a mutató most ismét az utolsó $00 byte utáni első byte-ra mutat.

Ezzel a programunk ismertetésének végére értünk. Reméljük, sikerült tovább bővítenünk a gépi kódú programozásban eddig szerzett ismereteket. A bemutatott program számos lehetőséget kínál a javításra, ill. továbbfejlesztésre. Nem vizsgáltuk pl. azt, hogy a program nem lépi-e túl a tár végét, így az is előfordulhat, hogy önmagát felülírja.

A célunk nem valamilyen díj elnyerése volt, hanem az, hogy egy, a gyakorlatban is használható program példáján keresztül elmagyarázzuk a gépi kódú utasításokat, és bemutassunk néhány programozási fogást.

22. A felhasználói csatlakozó (User-Port)

Valamennyi Commodore típusú számítógépnek van egy különleges csatlakozója. Ez az úgynevezett felhasználói csatlakozó, vagy elterjedt, eredeti nevén User Port. Ennek a csatlakozónak az a sajátossága, hogy univerzálisan programozható. Így pl. ezen keresztül igen egyszerűen megoldható külső készülékek ki- vagy bekapcsolása. A csatlakozó soros interface-ként is használható. A lehetőségek tehát igen sokfélék, és szinte korlátlanok.

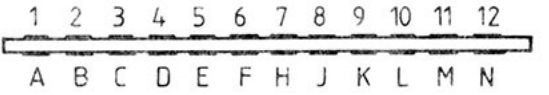
Sajnos itt is igaz azonban az a mondás, miszerint nincsen öröm üröm nélkül. A Commodore-gépek építői mintha szándékosan törekedtek volna arra, hogy mindegyik gépnek más és más legyen ez a csatlakozója. Kompatibilitásról tehát szó sem lehet. Ez azt jelenti, hogy azok a készülékek, amelyek a C 64-es csatlakozójába illesztve kifogástalanul működnek, a Plus/4-eshez nem csatlakoztathatók. Úgy látszik, hogy a kompatibilitás már túlzott követelmény lenne.

Ezen a tényen sajnos nem is változtathatunk. Bemutatjuk viszont azt, hogy mire használható ez a csatlakozó, és hogy miként programozható. A fejezetben arról is szó lesz, hogy hogyan csatlakoztatható egy Centronics-interface-szel nyomtató a Plus/4-eshez. Ez azt jelenti, hogy nem kényszerülünk arra, hogy csak Commodore nyomtatót használjunk, hanem a kereskedelemben kapható más, jóval nagyobb teljesítményűekkel is dolgozhatunk.

Lássuk először a felhasználói csatlakozó (User Port) érintkezőit.

22.1. táblázat:
A felhasználói csatlakozó érintkezői

Érintkező	Jelentés	Érintkező	Jelentés
1	Föld	A	Föld
2	+5 V	B	P0
3	−BRESET	C	RxD
4	P2	D	RTS
5	P3	E	DTR
6	P4	F	P7
7	P5	H	DCD
8	RxC	J	P6
9	ATN	K	P1
10	9 V AC	L	DSR
11	9 V AC	M	TxD
12	Föld	N	Föld



22.1. ábra: Felhasználói csatlakozó (User Port)

Nézzük először az ATN jelölésű érintkezőt. Erre a mikroprocesszor P2 jelű csatlakozója van kivezetve, ami a 7501-es processzor hét, programozható kimenetének egyike. Ez a kimenet egy inverteren keresztül csatlakozik az ATN-re. Az ATN ugyanakkor a soros buszra is rá van kapcsolva. Ha most a soros buszhoz egy lemezmeghajtó csatlakozik, és eközben az ATN-t átprogramozzuk, akkor ezzel megzavarhatjuk a meghajtó működését. Az ATN programozásánál ezért megfelelő óvatossággal kell eljárni.

A 10-es és a 11-es érintkezőkön 9 V váltakozó feszültség van. Ezeket a feszültségeket külső készülékek csatlakoztatásánál felhasználhatjuk. Azt azonban nem szabad figyelmen kívül hagyni, hogy a Plus/4-es hálózati egysége maximum 1 A-rel terhelhető, de ekkor már nagyon melegszik.

A BRESET a külsőleg csatlakoztatott készülékek RESET vezetéke. Amikor a Plus/4-es végrehajt egy RESET-et, akkor az ezen a vezetéken levő jelszint nulla lesz. Ez tehát egy ún. low-aktív jel (= akkor vált ki valamilyen hatást, ha alacsony szintű lesz). Ha viszont egy külső készülékről érkezik egy RESET, akkor ez a Plus/4-est nem befolyásolja. A felhasználói csatlakozó többi kivezetését 2 integrált áramkör foglalja el. Ezek típusjele 6551 és 6529.

A Plus/4-es gép egyébként már tartalmazza az RS-232 interface-hez alkalmas szoftvert. Ennek kezelését a géphez szállított kézikönyv részletesen ismerteti. Szó van ott a 6551-es IC (integrált áramkör) két regiszteréről is, viszont kimaradtak a regiszterek címei. A 6551-es IC neve ACIA (Asynchronous Communications Interface Adapter). Ez az IC alkotja a számítógép és a soros adatátvivő készülék vagy modem közötti interface-t.

A 6551-es IC a felhasználói csatlakozó következő érintkezőire van kivezetve:

érintkező	**jelölés**
M	TxD
E	DTR
D	RTS
8	RxC
H	DCD
L	DSR
C	RxD

A 6551-gyel 50–19200 baud (bit/s) közötti sebességgel valósítható meg a soros adatátvitel. Az IC-re egy 1,8432 MHz-es kvarc csatlakozik. A különböző adatátviteli sebességekhez tartozó frekvenciák ennek a frekvenciának a leosztásából jönnek létre.

Az IC-nek négy regisztere van, amelyek címei és nevei:

SFD00	0. regiszter	adatregiszter
SFD01	1. regiszter	állapotregiszer
SFD02	2. regiszter	parancsregiszter
SFD03	3. regiszter	vezérlőregiszter

A 6551-essel teljes és félduplex, 5 és 9 bit közötti jelátvitelek, és soros, echo üzemmód is megvalósítható. Arra nincs módunk, hogy a 6551-es teljes adatlapját itt közreadjuk. Úgy gondoljuk, ha valaki ez iránt érdeklődik, akkor az más forrásból megszerezheti. Azok számára, akik ezt a dokumentációt beszerezték, és az IC-t programozni akarják, de a Plus/4-esről nincs kapcsolási rajzuk, megjegyezzük, hogy a 6551-es néhány csatlakozója inverteren keresztül van kivezetve a felhasználói csatlakozóra. Az IC 11-es lába (−DTR) és a 8-as lába (−RTS) inverteren keresztül van a felhasználói csatlakozó E és D érintkezőjére kötve. Az 5-ös láb (RxC) és a 12-es láb (RxD) közvetlenül csatlakozik. A felhasználói csatlakozó H és L érintkezői ismét csak inverteren keresztül kapcsolódnak a 16-os (−DCD) és a 17-es (−DSR) lábakhoz, és 10 kOhmos ellenállásokon keresztül +5 V feszültségre. Az IC 9-es lába (−CTS) földpotenciálon van.

Térjünk rá most a 6529-esre. Ez az IC a 6551-eshez képest igen szerény felépítésű, mindössze egy 8 bites, párhuzamos interface. Az IC nyolc kimenete a felhasználói csatlakozó P0...P7 érintkezőire van kivezetve. A 6529-esnek csak egy regisztere van, amely a $FD10-es címen helyezkedik el.

22.1. A PROGRAMOZÁS

A 6529-es programozása meglehetősen egyszerű, és ehhez adatlapra sincs szükség. Ha a regiszterbe beírunk egy értéket, akkor a nyolc kimenő vezeték azonnal felveszi az ennek megfelelő állapotot. Írjuk be pl. a következő utasítást:

```
POKE 64784,255
```

vagy gépi kódban:

```
LDA #$FF; a bitminta %11111111
STA $FD10
```

Az utasítás hatására az IC valamennyi kimenete 1-es – magas – állapotú lesz. Ha viszont csak a P4-P7 kimeneteknek kell magasnak lenniük, akkor az utasítás:

```
POKE 64784,240
```

vagy gépi kódban:

```
LDA #$F0; a bitminta %11110000
STA $FD10
```

Ily módon bármelyik kimenő vezetéket be- vagy kikapcsolhatjuk attól függően, hogy milyen bitmintát írunk a 6529-es regiszterébe.

Az IC kimeneteit bemenetekként is lehet használni. Erre két lehetőség van: az egyik az lenne, hogy a regiszterben az illető kimenő vezetéket 0-ra állítjuk. Ha most erre a vezetékre kívülről egy 1-es, azaz +5 V érkezik, akkor a regiszterben a megfelelő bit értéke 1 lesz. A regiszter értékének olvasásával ezt a változást igen egyszerűen észlelhetjük. Ahhoz viszont, hogy ez működjön, rendkívül nagy áramnak kell folynia, ami hosszabb távon a 6529-est és esetleg magát a külső készüléket is tönkreteheti. Ezt a megoldást tehát semmi esetre sem ajánljuk.

Ezzel szemben nagyon jól működik a másik megoldás: a megfelelő kimenő vezetékeket magasra állítjuk. Ha most egy külső vezeték a földpotenciálra kerül, azaz

alacsony lesz, akkor a 6529-es regiszterének olvasásakor az illető bit alacsony lesz. Erre egy példa:

A felhasználói csatlakozó P3-as érintkezőjéhez egy kapcsolót csatlakoztatunk, pl. legyen ez egy riasztóberendezés ablakra szerelhető kapcsolója. Ha valaki ezt a kapcsolót működteti, akkor a P3-as vezeték földpotenciálra kerül. Ezt a következő programmal tudjuk megállapítani:

```
10 PRINT CHR$ (147)
20 POKE 64784,8:REM %00001000 BITMINTA
30 IF PEEK(64784)>0THEN30
40 PRINT "A P3 KAPCSOLOT MEGNYOMTAK"
50 VOL 8
60 SOUND1,950,10
70 SOUND1,980,10
80 GOTO 60
```

Mihelyt lenyomja valaki a kapcsolót, ezt a tényt a számítógép kiírja a képernyőre, és megszólaltat egy akusztikus jelet. Tehát a Plus/4-es mint riasztóberendezés? Ugyan miért ne! Éjszakára erre a célra is lehet használni. Ha viszont ehhez a tévé-készüléknek állandóan bekapcsolva kellene lennie, akkor ez az egész értelmetlen lenne, hiszen a hosszú üzemelés nyilván nem használ a készüléknek. Azonban ezen is lehet segíteni, méghozzá úgy, hogy akkor, amikor valaki működteti a kapcsolót, akkor egy másik kimenő vezetéket használunk vészjelző vezetékként. Ezt a vezetéket rákötjük egy kis tranzisztoros áramkörre, amely mondjuk egy jelfogón keresztül bekapcsol egy riasztólámpát és egy hangjelet. Ilyen módon a 6529-essel tehát maximum hét bemenő vezeték, és egy kimenő riasztó vezeték valósítható meg. Sőt, még további, két bemenő vezetéket is tudunk kapcsolni, hiszen a 6551-es állapotregiszterében még a DCD és a DSR csatlakozók állapotai is lekérdezhetők.

A teljesség kedvéért még megjegyezzük, hogy a felhasználói csatlakozón akár tíz bemenő vezeték és egy kimenő vezeték is megvalósítható: ezek a 6529-es nyolc bemenete, ehhez jön még a 6551-es két bemenete (DCD és DSR), riasztó vezetékként pedig a 6551-es RTS csatlakozója. Sőt, ha még az ATN csatlakozót is használnánk...! Amint látjuk, a felhasználói csatlakozó számos célra használható. Most bemutatjuk egy 10 bemenetű riasztóberendezés programját.

22.2. A PLUS/4-ES MINT RIASZTÓBERENDEZÉS

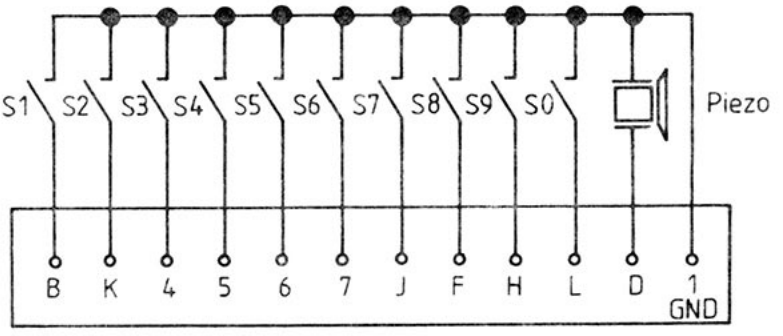
Lássuk először a programot. A $4000–$4009 címeken adatbyte-ok vannak.

```
.4000 31 32 33 34 35 36 37 38
.4008 39 30

.400A A9 0E     LDA #$0E
.400C A2 CE     LDX #$CE
.400E 78        SEI
.400F 8D 14 03  STA $0314   ;megszakitasvektort
.4012 8E 15 03  STX $0315   ;visszaallitani
.4015 58        CLI
.4016 A9 93     LDA #$93    ;kepernyot torolni
```

```
.4018 20 4C FF    JSR $FF4C     ;PRINT rutin
.401B A2 00       LDX #$00
.401D BD 00 40    LDA $4000,X ;az 1 es 9 kozotti szamokat
.4020 9D 00 0C    STA $0C00,X ;valamint a 0-t a
.4023 E8          INX           ;kepernyore irni
.4024 E0 0A       CPX $0A
.4026 D0 F5       BNE $401D
.4028 A9 FF       LDA #$FF      ;%11111111 bitmintat
.402A 8D 10 FD    STA $FD10     ;a 6529-esbe irni
.402D A9 00       LDA #$00
.402F 85 D0       STA $D0
.4031 85 D1       STA $D1
.4033 85 D2       STA $D2
.4035 A9 00       LDA #$00
.4037 8D 11 FF    STA $FF11     ;hangregiszter a TED-ben
.403A A9 FF       LDA #$FF      ;a 6529-es regiszterevel
.403C 4D 10 FD    EOR $FD10     ;KIZARO-VAGY muvelet
.403F 85 D0       STA $D0       ;eredmeny $D0-ba
.4041 A9 60       LDA #$60      ;a 6551-es allapotregiszterebol
.4043 2D 01 FD    AND $FD01     ;az 5-os es a 6-os biteket
                                 lekerdezni
.4046 4A          LSR           ;otszor jobbra tolni
.4047 4A          LSR
.4048 4A          LSR
.4049 4A          LSR
.404A 4A          LSR
.404B 85 D1       STA $D1       ;eredmeny $D1-be
.404D 20 11 DB    JSR $DB11     ;billentyuzetet lekerdezni
.4050 D0 B8       BNE $400A     ;billentyu lenyomva
.4052 A5 D0       LDA $D0       ;voltak lenyomott kapcsolok?
.4054 D0 04       BNE $405A     ;igen
.4056 A5 D1       LDA $D1
.4058 F0 E0       BEQ $403A
.405A A5 D2       LDA $D2       ;veszjelzo magas?
.405C D0 1A       BNE $4078     ;igen
.405E 78          SEI           ;nem. Riasztohangot bekapcsolni.
.405F A9 A0       LDA #$A0      ;a riasztohang megszakito
.4061 A2 40       LDX #$40      ;rutinjanak kezdocimet
.4063 8D 14 03    STA $0314     ;a megszakito rutin
.4066 8E 15 03    STX $0315     ;vektoraba beirni
.4069 58          CLI
.406A A9 C7       LDA #$C7
.406C 8D 12 FF    STA $FF12     ;a TED hangregiszteret
.406F A9 1F       LDA #$1F      ;bekapcsolni
.4071 8D 11 FF    STA $FF11
.4074 A9 01       LDA #$01
.4076 85 D2       STA $D2
.4078 A2 FF       LDX #$FF
.407A A9 80       LDA #$80
.407C E8          INX
.407D E0 08       CPX #$08
.407F F0 0D       BEQ $408E
.4081 46 D0       LSR $D0
.4083 90 F5       BCC $407A
.4085 1D 00 08    ORA $0800,X ;mindegyik zart kapcsolohoz
.4088 9D 00 08    STA $0800,X ;tartozo szamot a
.408B 4C 7A 40    JMP $407A     ;kepernyon villogtatni
.408E 46 D1       LSR $D1
.4090 90 06       BCC $4098
.4092 1D 00 08    ORA $0800,X
.4095 9D 00 08    STA $0800,X
.4098 E8          INX
```

```
.4099 E0 0A       CPX #$0A
.409B D0 F1       BNE $408E
.409D 4C 3A 40    JMP $403A    ;ugras az ujabb lekerdezeshez
.40A0 A2 1F       LDX #$1F     ;uj megszakitorutinnal
.40A2 A9 08       LDA #$08     ;riasztohangot eloallitani
.40A4 4D 02 FD    EOR $FD02
.40A7 8D 02 FD    STA $FD02
.40AA A0 1F       LDY #$1F
.40AC 88          DEY
.40AD D0 FD       BNE $40AC
.40AF CA          DEX
.40B0 D0 F0       BNE $40A2
.40B2 A9 FF       LDA #$FF
.40B4 4D 0E FF    EOR $FF0E
.40B7 8D 0E FF    STA $FF0E
.40BA 4C 0E CE    JMP $CE0E    ;vissza a regi megszakito
                               ;rutinhoz
```



22.2. ábra A felhasználói csatlakozó kapcsolása a riasztóberendezéshez

Ezzel a riasztóberendezéssel tehát tíz kapcsoló állapotát kérdezhetjük le. Ha valaki a kapcsolók egyikét működteti, akkor megszólal egy hang, és a képernyőn a kapcsolóhoz tartozó szám villog. A szám még akkor is villog, amikor a kapcsolót elengedjük. A riasztóhang is szól tovább. Így tehát bármikor ellenőrizhető, hogy valaki működtetett-e egy kapcsolót. A riasztást a billentyűzet bármely billentyűjének lenyomásával lehet megszüntetni.

A programnak az a különlegessége, hogy a berendezés tévékészülék nélkül is működtethető, mert a riasztó hangjelzést a felhasználói csatlakozóról is le lehet ágaztatni. Ehhez egy piezokristályos hangszórót (és csak ilyet!) kell a felhasználói csatlakozó D (RTS) érintkezője és a földpont (GND) közé csatlakoztatni. Ez a hangszóró vészjelzéskor mindenen áthatoló, még a halottakat is felébresztő hangot ad. Ha valakinek még ez a hangerő sem lenne elegendő, akkor a hangszórót bekötheti a magnócsatlakozó hüvely 1-es és 3-as érintkezői közé. Ezen a két érintkezőn a kazettásmagnó motorját meghajtó, közel 9V feszültség van. Ezt a feszültséget a 7501-es mikroprocesszor P3-as kimenetével lehet be-, ill. kikapcsolni. Ehhez az adat-

irányregiszter ($0000 címen) 3-as bitjét magasra kell állítani. Ezután az adatregiszter ($0001 címen) 3-as bitjének magasra állításával, ill. törlésével lehet a feszültséget be-, ill. kikapcsolni. Az említett hangszóró szaküzletekben kapható.

A program indítása:

SYS 16394

vagy gépi kódban:

G 400A

Indítás előtt a programot tároljuk a TEDMON segítségével. Ne felejtsük el, hogy a program a $4000 címen kezdődik!

Ha valakinek a riasztóberendezés tíz bemenete nem lenne elegendő, akkor ezen könnyen lehet segíteni. Mi ehhez egy bemeneti/kimeneti chip-et csatlakoztattunk a Plus/4-es bővítőjére. A kísérleti berendezésünkhöz egy 6522-es típusú chip-et használtunk, amelynek 16 bemenő és kimenő vezetéke van. Ez a chip a Plus/4-eshez pl. modulként is csatlakoztatható (lásd a 21. fejezetet). Ez a chip ekkor a $8000–$BFFF közötti teljes címtartományt elfoglalja, de végül is ez senkit sem zavar.

A riasztóberendezésünk programjához további hozzáfűznivalónk nincs. Úgy gondoljuk, hogy elegendők a programban adott magyarázatok. A riasztóhanggal kapcsolatban még megjegyezzük, hogy ezt egy rövid rutin állítja elő, amelyet az operációs rendszer minden egyes megszakításkor, a tulajdonképpeni megszakító rutin előtt dolgoz fel.

22.3. A CENTRONICS-NYOMTATÓK ILLESZTÉSE A FELHASZNÁLÓI CSATLAKOZÓN KERESZTÜL

A piacon ma már szinte áttekinthetetlen a nyomtatók választéka. Aki a számítógépén intenzíven és célirányosan akar dolgozni, annak előbb vagy utóbb nyomtatót is be kell szereznie, mert ugyan mi haszna egy szövegfeldolgozó programnak, ha a megszerkesztett szövegeket nem tudjuk papírra írni?

A vásárláskor körültekintően mérlegelnünk kell, hogy melyek a nyomtatóval kapcsolatos jelenlegi és jövőbeli elképzeléseink. Beszerezhetünk pl. egy soros interface-ű, speciális Commodore nyomtatót, vagy pedig egy Centronics, azaz párhuzamos interface-ű nyomtatót.

A Plus/4-esnek van egy soros interface-e, amelyhez pl. az 1541-es lemezmeghajtó is csatlakoztatható. A gépbe párhuzamos interface viszont nincs beépítve, de ez a felhasználói csatlakozón leképezhető. Ehhez természetesen minden alkalommal be kell tölteni a megfelelő programot, ami persze nehezíti a dolgunkat, de arra is gondoljunk, hogy a nyomtatóra még akkor is szükségünk lehet, amikor esetleg egy másik számítógépre térünk át. A Commodore AMIGA gépe pl. már sorozatban Centronics-interface-szel készül.

A következőkben bemutatjuk, hogy miként lehet a felhasználói csatlakozót Centronics-interface-szé átalakítani. (Sajnos a példaprogramunk a Plus/4-esbe épített szoftverekkel nem tud együttműködni.)

22.4. A CENTRONICS-RÓL

A Centronics-interface-t a Centronics cég fejlesztette ki a nyomtatói számára. Ez az interface rövid idő alatt egyfajta szabvánnyá vált, és időközben szinte valamennyi nyomtatóhoz kapható, ill. a nyomtatókba már be van építve. A Centronics-interface esetén a számítógép az adatokat párhuzamosan küldi a nyomtatóhoz. A nyolc adatvezeték mellett van még egy sereg más, a vezérlő funkciókat ellátó vezeték is, amelyek egy 36 pólusú csatlakozóban vannak összefogva. A Centronics-interface működtetéséhez nekünk viszont csak a 8 adatvezetékre, a BUSY, a -STROBE és a földvezetékre van szükségünk.

Eszerint tehát kell egy minimun 11 erű kábel, amelynek az egyik végén a Plus/4-es felhasználói csatlakozójába illő, a másik végén pedig a Centronics-csatlakozóba illő dugasz van. Az alkatrészeket megvehetjük szaküzletben, a forrasztásokat saját magunk végezhetjük. A kábel bekötése a következő:

Felhasználói csatlakozó érintkezői	Jelölés	Centronics-csatlakozó érintkezői
A	föld	16
B	P0	2
K	P1	3
4	P2	4
5	P3	5
6	P4	6
7	P5	7
J	P6	8
F	P7	9
H	(DCD) BUSY	11
D	(RTS) -STROBE	1

Ezzel a kábelünk már meg is volna. Lássuk most a hozzá való programot:

```
.065E A9 00     LDA #$00     ;a 6529-es kimeno vezetekeit
.0660 8D 10 FD  STA $FD10    ;nullara allitani
.0663 A9 08     LDA #$08     ;a -STROBE-ot '1'-re allitani
.0665 8D 02 FD  STA $FD02    ;-STROBE = 6551-es RTS-e
.0668 A9 75     LDA #$75     ;a program kezdocimenek also
                              byte-ja
.066A A2 06     LDX #$06     ;a program kezdocimenek felso
                              byte-ja
.066C 78        SEI
.066D 8D 24 03  STA $0324    ;kezdocimet a karakter-
.0670 8E 25 03  STX $0325    ;kiiro vektorba irni
.0673 58        CLI
.0674 60        RTS
.0675 48        PHA          ;AKKU-ban levo karaktert a
                              verembe
```

```
.0676 A5 99     LDA $99     ;az aktualis kiviteli (output)
                             egyseg
.0678 C9 04     CMP #$04    ;a #04 azonositoju nyomtato?
.067A F0 03     BEQ $067F   ;igen
.067C 4C 4E EC  JMP $EC4E   ;ha nincs nyomtato, akkor
                             vissza
.067F 68        PLA         ;karaktert a verembol elovenni
.0680 8D 10 FD  STA $FD10   ;karaktert a 6529-esbe irni
.0683 A9 00     LDA #$00    ;
.0685 8D 02 FD  STA $FD02   ;-STROBE-ot '0'-ra
.0688 A9 08     LDA #$08    ;
.068A 8D 02 FD  STA $FD02   ;-STROBE-ot ismet '1'-re
.068D A9 20     LDA #$20    ;
.068F 2C 01 FD  BIT $FD01   ;a BUSY meg '1' ?
.0692 F0 FB     BEQ $068F   ;igen
.0694 18        CLC         ;a nyomtato kesz,
.0695 60        RTS         ;tehat vissza
```

A program indítása: SYS 1630

Ezután az OPEN 1,4

utasítással megnyithatjuk a nyomtató csatornát. A kiírás a Centronics-interface-en keresztül történik. Ha mindent jól csináltunk, akkor ennek működnie kell. Most pl. papírra nyomtathatjuk az előbbi Centronics-programunkat:

```
CMD 1
MONITOR
D 065E 0695
```

Az itt kinyomtatott programmal (és egy 1541-es meghajtót csatlakoztatva) a Plus/4-es kifogástalanul működtetett egy EPSON-LX80 és egy Speedy 100-80 típusú nyomtatót. Más készülékeket sajnos nem tudtunk kipróbálni.

Még néhány szó a Centronics-interface-hez. A programot elláttuk ugyan néhány megjegyzéssel, de azért szeretnénk elmagyarázni a működését.

Először az adatvezetékeket kell bekapcsolni, ami a programban a $0680-as címen történik. Ezután közölni kell a nyomtatóval, hogy az adatvezetékeken érvényes adatok vannak. Ez a -STROBE vezetéken keresztül történik. Mivel a -STROBE low-aktív jel, ezt nullára kell állítani. Ennek a jelnek minimum 0,5 mikroszekundum hosszan aktívnak kell lennie, ezután ismét magasra (1) állítható. Mivel a Plus/4-esünk azért ilyen gyorsan nem dolgozik, megtehetjük, hogy a -STROBE-ot közvetlenül egymás után 0-ra majd ismét 1-re állítjuk.

Közvetlenül aztán, hogy a -STROBE aktív lett, a nyomtató a BUSY (magyarul = foglalt) vezetéket magasra állítja. A BUSY mindaddig magas, míg a karakter kinyomtatását be nem fejezte. Amikor a nyomtatással végzett, akkor az alacsonyra állításával jelzi, hogy kész az újabb karakter fogadására. A BUSY vezeték állapotát a programban a $068F címen kérdezzük le.

Még megjegyezzük, hogy a Plus/4-es LF-jelet (LINE FEED, soremelés) nem küld a nyomtatóhoz. A nyomtatón levő mikrokapcsoló beállításával azonban bekapcsolhatjuk az úgynevezett AUTO-FEED üzemmódot. Ilyenkor a nyomtató minden esetben, amikor egy kocsi vissza (CR) jelet kap, automatikusan egy soremelést is végrehajt. Ha a nyomtatónkat mégsem lehetne így beállítani, akkor a Centronics-programunkat kell módosítanunk. Ekkor minden jelnél meg kell vizsgálni, hogy nem CR ($CD) jel-e. Amennyiben igen, akkor még egy LF ($0A) jelet is ki kell küldeni.

A FÜGGELÉK

A BASIC hibaüzenetei

A Plus/4-esbe számos hibaüzenet van beépítve, amelyek megkönnyítik a hiba felismerését.

Az ER változó a hiba számát, az ERR$ karakterváltozó a hibaüzenet szövegét tartalmazza. Ha a hiba a közvetlen (parancs) üzemmódban következik be, akkor a számítógép az ERR$ elé kérdőjelet tesz. Ha a hiba egy programon belül lép fel, akkor – amennyiben a program futásának megszakadását egy TRAP-pal nem kerüljük meg – a hibaüzenet után kiírja a sorszámot is. Azt a sorszámot, amelyben a hiba fellépett, az EL változó tartalmazza. Itt most csak az ER és az ERR$ változók tartalmával foglalkozunk.

ER	Az ERR$ tartalma	A hiba leírása
1	TOO MANY FILES	A Plus/4-es géppel egyidejűleg maximum 10 állomány (file) nyitható meg. Ha ez a hiba lép fel, akkor a CLOSE állomány száma utasítással egy állományt le kell zárni.
2	FILE OPEN	Egy már megnyitott állományt még egyszer nem lehet megnyitni. Ha újabb állományt kell megnyitni, akkor ennek más számot kell adni vagy le kell zárni a régit.
3	FILE NOT OPEN	Ezt a hibaüzenetet akkor kapjuk, ha olyan állományba akarunk írni, vagy onnan olvasni, amelyet előzőleg egy OPEN utasítással nem nyitottunk meg.
4	FILE NOT FOUND	A rendszer megkísérelte egy program vagy egy adatállomány beolvasását, de nem találta a lemezen. Adatmagnó esetén azt jelenti, hogy a gép elolvasta már a szalag vége jelzést, de a megadott nevű programot vagy állományt nem találta meg.
5	DEVICE NOT PRESENT	Olyan külső egységgel próbáltuk meg az adatátvitelt, amely nincs a géphez csatlakoztatva.

6	NOT INPUT FILE	Olyan állományból akarunk olvasni, amely 1 vagy 2 másodlagos címmel lett megnyitva. Az olvasás csak 0 másodlagos címmel való megnyitás esetén lehetséges. Lásd még OPEN.
7	NOT OUTPUT FILE	Olyan állományba akarunk írni, amelyet 0 másodlagos címmel nyitottunk meg. Ez csak 1 vagy 2 másodlagos címmel való megnyitás esetén lehetséges. Lásd még OPEN.
8	MISSING FILE NAME	Lemezmeghajtó esetén a programnak vagy az adatállománynak mindig nevet kell adni.
9	ILLEGAL DEVICE NUMBER	A készülékszám és az utasítás nem illik egymáshoz. Például: SAVE"PROBA",4. A négyes szám a nyomtató azonosítója.
10	NEXT WITHOUT FOR	A program egy NEXT utasításhoz érkezett úgy, hogy előzőleg nem találkozott a hozzá illő FOR...TO utasítással, vagy a NEXT után hibás a változónév.
11	SYNTAX	Hibásan írtunk egy BASIC utasítást, nem megengedett karaktert használtunk, vagy elfelejtettünk egy kötelező írásjelet kitenni.
12	RETURN WITHOUT GOSUB	A program úgy érkezett egy RETURN utasításhoz, hogy előzőleg egy GOSUB nem ágaztatta el erre az alprogramra.
13	OUT OF DATA	A READ utasítással a meglevő adatoknál többet akartunk beolvasni. Lehet, hogy egy RESTORE hiányzik a programból.
14	ILLEGAL QUANTITY	Az argumentumban levő szám vagy túl nagy vagy túl kicsi. Például: POKE 3400,300. Az argumentum értéke maximum 255 lehet. Vagy POKE 3400,−1. Az argumentum minimális értéke 0 lehet.
15	OVERFLOW	Olyan számokkal dolgozunk, vagy olyan eredményt kaptunk, ami nagyobb mint 1.701411833E+38.
16	OUT OF MEMORY	A hibának sok oka lehet: túl hosszú a program, túl sok változót használtunk vagy túl nagy tömböket definiáltunk. Akkor is ezt a hibaüzenetet kapjuk, ha túlságosan sok ciklust vagy alprogramot ágyaztunk egymásba, vagy ha túl sok saját függvényt használunk.

17	UNDEF'D STATEMENT	Olyan sorszámra adtuk ki a RUN, GOTO, GOSUB vagy a RESUME utasítást, amely a programban nem létezik.
18	BAD SUBSCRIPT	Olyan indexes változóra hivatkozunk, amelyet a tömbváltozókban nem definiáltunk.
19	REDIM'D ARRAY	Egy tömböt csak egyszer szabad dimenzionálni. Lásd DIM is.
20	DIVISION BY ZERO	A 0-val való osztás nem megengedett.
21	ILLEGAL DIRECT	Ezt a hibaüzenetet akkor kapjuk, ha csak programban használható utasításokat – pl. INPUT, GET – közvetlen üzemmódban adunk ki.
22	TYPE MISMATCH	Szám helyett betűt, vagy betű helyett számot írtunk.
23	STRING TOO LONG	Karakterláncok vagy programsorok beírásánál a 88 karaktert, vagy karakterlánc műveletek esetén a 255 karaktert túlléptük.
24	FILE DATA	A lemezről vagy kazettáról betű helyett számot, vagy fordítva akartunk beolvasni. Lásd még TYPE MISMATCH-nél is.
25	FORMULA TOO COMPLEX	A megadott számítási művelet túlságosan összetett. Először részeredményeket kell számítani, majd ezekből a végeredményt.
26	CAN'T CONTINUE	Ha egy programot a BREAK után megváltoztattunk vagy hibát csináltunk, akkor a CONT utasítással már nem folytatható.
27	UNDEF'D FUNCTION	A meghívott függvényt DEF FN utasítással nem definiáltuk.
28	VERIFY	A lemezen vagy kazettán levő adatok és a tárban levő adatok nem egyeznek meg.
29	LOAD	A program vagy az állomány betöltése nem volt hibátlan.
30	BREAK	A programot egy STOP utasítás vagy a STOP billentyű lenyomása megszakította.

31	CAN'T RESUME	A program egy RESUME utasítással találkozott úgy, hogy előzőleg a hibaüzenetet nem hívtuk meg.
32	LOOP NOT FOUND	A programban előfordult DO utasításhoz nem tartozik LOOP.
33	LOOP WITHOUT DO	A program LOOP vagy EXIT utasítással találkozott anélkül, hogy a ciklust egy DO-val előzőleg megnyitottuk volna.
34	DIRECT MODE ONLY	A programnak olyan utasítást kellene feldolgoznia, amely csak közvetlen üzemmódban használható, pl. DELETE.
35	NO GRAPHICS AREA	A program grafikus utasítással találkozott anélkül, hogy előzőleg bekapcsoltuk volna a grafikus üzemmódot. (nincs grafikus terület lefoglalva)
36	BAD DISK	A lemez formázása közben hiba lépett fel.

B FÜGGELÉK

A lemezmeghajtó hibaüzenetei

Ha lemezmeghajtóval dolgozunk és a munka során a meghajtóval kapcsolatos hiba lép fel, akkor a meghajtón levő piros LED lámpa villog. A DS és a DS$ foglalt változók tartalmát a meghajtó 15-ös, parancs- vagy hibacsatornáján keresztül lehet elolvasni. A DS a hiba számát tartalmazza, a DS$ pedig a hiba szöveges megnevezését, valamint a lemez azonos sáv- és szektorszámát, ahol a hiba fellépett, ill. ahol azt a lemezoperációs rendszer észlelte.

A hibaüzenetek száma és jelentése

A hiba		
Száma	**Üzenete**	**Leírása**
00	OK	Nincs hiba
01	FILES SCRATCHED	Egy SCRATCH utasítás után megadja a törölt állományok számát.
20	READ ERROR (block header not found)	A lemezmeghajtó nem találta a blokk fejlécét. Ok: nem megengedett szektorszám, elrontott fejléc, hibás lemez.
21	READ ERROR (no sync character)	A lemezmeghajtó nem találja a szinkronjeleket. Oka lehet, hogy nincs lemez a meghajtóban vagy a lemez nincs formálva, vagy az író/olvasófej elállítódott.
22	READ ERROR (data block no present)	A meghajtó olyan blokkot olvasott, amelynek a fejlécében hibás a vizsgálóösszeg.
23	READ ERROR (checksum error in data block)	Mint a 22-es, de a gép a blokkot a DOS tárjából olvasta.
24	READ ERROR (byte decoding error)	Mint a 23.
25	WRITE ERROR (write-verify error)	A meghajtó egy blokkot hibásan tárolt. Hibás lemez is okozhatja.

A hiba		
Száma	Üzenete	Leírása
26	WRITE PROTECTION	A lemezen írásvédő van. Ha ilyen lemezre írni akarunk, akkor ezt a hibaüzenetet kapjuk.
27	READ ERROR (checksum error in header)	Mint a 22.
28	WRITE ERROR (long data block)	A lemez formázása nem volt hibátlan. Ezt a hibaüzenetet csak a lemezre való íráskor kapjuk.
29	DISK ID MISMATCH	A lemez ID-je nem egyezik meg a DOS-ban tárolt ID-vel.
30	SYNTAX ERROR (general syntax)	Hibás utasítást adtunk ki.
31	SYNTAX ERROR (invalid command)	Mint a 30.
32	SYNTAX ERROR (invalid command)	58 karakternél hosszabb utasítássort írtunk be.
33	SYNTAX ERROR (invalid file name)	Nem megengedett állománynév. A jokerjel hibás alkalmazása okozhatja.
34	SYNTAX ERROR (no file given)	A DOS nem ismerte fel az állomány nevét. Lehet, hogy az utasításszó és az állomány neve közötti kettőspont kimaradt.
39	SYNTAX ERROR (invalid command)	Parancscsatorna küldi
50	RECORD NOT PRESENT	Egy relatív állomány olyan rekordjához próbáltunk meg hozzáférni, amelyet még nem hoztunk létre. Ezt a hibaüzenetet kapjuk akkor is, ha az állományt bővíteni akarjuk. Ebben az esetben a hibajelzést nem kell figyelembe venni.
51	OVERFLOW IN RECORD	Túlléptünk a maximális rekordhosszon. (A rekordhoz tartozik pl. a kocsi vissza – CHR$(13) – karakter is!)

A hiba		
Száma	Üzenete	Leírása
52	FILE TOO LARGE	A megadott rekordszám túl nagy (meghaladja a lemez kapacitását).
60	WRITE FILE OPEN	Lezáratlan állományt akartunk megnyitni.
61	FILE NOT OPEN	Olyan állományhoz akartunk hozzáférni, amelyet előzőleg nem nyitottunk meg.
62	FILE NOT FOUND	A keresett állomány nincs a lemezen.
63	FILE EXITS	Olyan néven akartunk a lemezre írni egy állományt, amilyen nevű már van rajta.
64	FILE TYPE MISMATCH	Hibás állománytípust adtunk meg.
65	NO BLOCK	A B-A (Block Allocate) utasítással le akartunk foglalni egy blokkot, de a blokk már foglalt volt. A DS$ tartalmazza a legközelebbi szabad blokk sáv- és szektorszámát (0, ha az összes, magasabb számú blokk foglalt).
66	ILLEGAL TRACK AND SECTOR	Egy blokkutasítással egy nem létező blokkhoz fordultunk.
67	ILLEGAL SYSTEM TORS	Egy blokkban hibás csatolócím van.
70	NO CHANNEL (available)	A lemezmeghajtó valamennyi csatornája foglalt.
71	DIRECTORY ERROR	A BAM és a tartalomjegyzék nem egyezik meg. A meghajtót inicializálni kell.
72	DISK FULL	Már 662 blokk betelt (vagy a tartalomjegyzék már 144 bejegyzést tartalmaz.)
73	DOS MISMATCH	A lemezre az adatokat, ill. állományokat más lemezmeghajtó egység rendszere írta fel, ezért ezek az 1541-es vagy az 1551-es meghajtóval nem olvashatók.
74	DRIVE NOT READY	Nincs lemez a meghajtóban.

C FÜGGELÉK

A BASIC-utasítások rövidítései

A legtöbb BASIC-kulcsszó rövidítve is írható. Ezáltal a programozásnál idő takarítható meg. A rövidítések használatával egy programsorba 88-nál több karakter is írható. Az erre vonatkozó tudnivalókat a programozásról szóló fejezet tartalmazza. A programok ilyen írásmódjával tárhely is megtakarítható, mivel kevesebb sorszámra van szükség. Egy programsorba azonban 88-nál több karaktert nem célszerű írni, mert ezek a sorok nehezen áttekinthetők és szerkeszthetők. A LIST utasítást követően a rövidítve beírt utasításszavak is a teljes hosszukban jelennek meg a képernyőn. Vegyük figyelembe, hogy a TAB és az SPC rövidítései már az első zárójelet is tartalmazzák.

Az összes utasítást nagybetű/kisbetű üzemmódban nyomtattuk a könyvbe. A nagybetűket tehát úgy kapjuk, hogy az illető billentyűvel együtt a SHIFT billentyűt is lenyomjuk. Ha nagybetű/grafikus üzemmódban írjuk a programot, akkor az illető billentyűhöz szintén használni kell a SHIFT billentyűt. Ekkor a képernyőn az illető betű helyett a grafikus karakter jelenik meg.

Név	Rövidítés	Vonatkozás
abs	aB	Számítás
and	aN	Logika
asc	aS	Karakterlánc
atn	aT	Számítás
auto	aU	Programozási segítség
backup	bA	Mágneslemez
box	bO	Grafika
char	chA	Grafika/szöveg
chr$	cH	Karakterlánc
circle	cL	Grafika
close	clO	Lemez/kazetta
clr	cL	Változók
cmd	cM	Kiviteli (output) készülék
collect	colL	Mágneslemez
color	coL	Szín
cont	cO	Programfutás
copy	coP	Mágneslemez
data	dA	Programadatok
def	dE	Számítás
delete	deL	Törlés
dim	dI	Tömbök (indexes változók)
directory	diR	Mágneslemez
dload	dL	Mágneslemez
draw	dR	Grafika
dsave	dS	Mágneslemez
end	eN	Programfutás
err$	eR	Változó
exp	eX	Számítás
for	fO	Ciklus
fre	fR	Tár
get	gE	Bevitel
getkey	getkE	Bevitel
get#	gE#	Lemez/kazetta
gosub	goS	Programugrás
goto	gO	Programugrás
graphic	gR	Grafika
gshape	gS	Grafika
header	heA	Mágneslemez
hex$	hE	Számok
input#	iN	Lemez/kazetta
instr	inS	Karakterlánc

Név	Rövidítés	Vonatkozás
joy	jO	Bevitel
key	kE	Billentyűzet
left$	leF	Karakterlánc
let	lE	Változók
list	lI	Programozási segítség
load	lO	Lemez/kazetta
locate	loC	Grafika
loop	loO	Ciklus
mid$	mI	Karakterlánc
monitor	mO	Gépi kód
next	nE	Ciklus
not	nO	Logika
open	oP	Lemez/kazetta
paint	pA	Grafika
peek	pE	Tár
poke	pO	Tár
print	?	Kiírás
print#	pR	Lemez/kazetta
printusing	?usI	Kiírás
pudef	pU	Print Using
rclr	rC	Grafika
rdot	rD	Grafika
read	rE	Data
rename	reN	Mágneslemez
renumber	renU	Programozási segítség
restore	reS	Data
resume	resU	Hibakezelés
return	reT	Programugrás

Név	Rövidítés	Vonatkozás
rgr	rG	Grafika
right$	rI	Karakterlánc
rlum	rL	Grafika
rnd	rN	Számok
run	rU	Programfutás
save	sA	Lemez/kazetta
scale	scA	Grafika
scnclr	sC	Grafika
scratch	scR	Mágneslemez
sgn	sG	Számok
sin	sI	Számítás
sound	sO	Zene
spc(	sP	Kiírás
sqr	sQ	Számítás
sshape	sS	Grafika
stop	sT	Programfutás
str$	stR	Karakterlánc
sys	sY	Gépi kód
tab(	tA	Kiírás
trap	tR	Hibakezelés
troff	troF	Hibakezelés
tron	trO	Hibakezelés
until	uN	Ciklus
usr	uS	Gépi kód
val	vA	Karakterlánc
verify	vE	Lemez/kazetta
vol	vO	Zene
wait	wA	Tár
while	wH	Ciklus

D FÜGGELÉK

A BASIC tokenek

Mielőtt a BASIC interpreter a programsorokat beírja a tárba, megkeresi a BASIC kulcsszavakat, és tárhely megtakarítás céljából ezeket 128 és 253 közötti számokkal helyettesíti. Ezeket a számokat nevezzük BASIC tokeneknek.

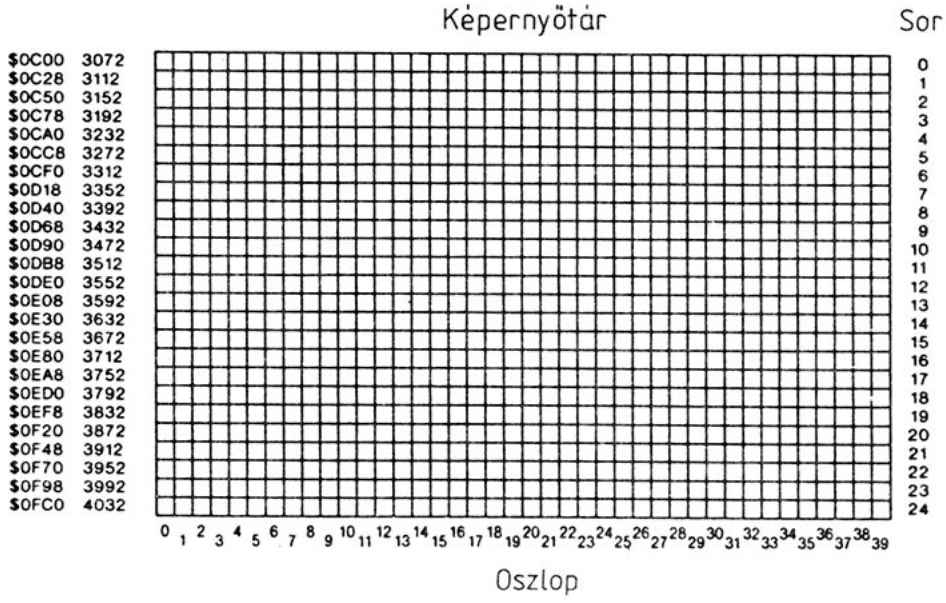
Szám	Utasítás	Szám	Utasítás	Szám	Utasítás
128	END	161	GET	194	PEEK
129	FOR	162	NEW	195	LEN
130	NEXT	163	TAB(	196	STR$
131	DATA	164	TO	197	VAL
132	INPUT#	165	FN	198	ASC
133	INPUT	166	SPC(	199	CHR$
134	DIM	167	THEN	200	LEFT$
135	READ	168	NOT	201	RIGHT$
136	LET	169	STEP	202	MID$
137	GOTO	170	+	203	GO (a GOTO
138	RUN	171	−		része)
139	IF	172	*	204	RGR
140	RESTORE	173	/	205	RCLR
141	GOSUB	174	^	206	RLUM
142	RETURN	175	AND	207	JOY
143	REM	176	OR	208	RDOT
144	STOP	177	>	209	ON
145	ON	178	=	210	HEX$
146	WAIT	179	<	211	ERR$
147	LOAD	180.	SGN	212	INSTR
148	SAVE	181	INT	213	ELSE
149	VERIFY	182	ABS	214	RESUME
150	DEF	183	USR	215	TRAP
151	POKE	184	FRE	216	TRON
152	PRINT#	185	POS	217	TROFF
153	PRINT	186	SQR	218	SOUND
154	CONT	187	RND	219	VOL
155	LIST	188	LOG	220	AUTO
156	CLR	189	EXP	221	PUDEF
157	CMD	190	COS	222	GRAPHIC
158	SYS	191	SIN	223	PAINT
159	OPEN	192	TAN	224	CHAR
160	CLOSE	193	ATN	225	BOX

Szám	Utasítás
226	CIRCLE
227	GSHAPE
228	SSHAPE
229	DRAW
230	LOCATE
231	COLOR
232	SCNCLR
233	SCALE
234	HELP
235	DO
236	LOOP
237	EXIT
238	DIRECTORY
239	DSAVE
240	DLOAD
241	HEADER
242	SCRATCH
243	COLLECT
244	COPY
245	RENAME
246	BACKUP
247	DELETE
248	RENUMBER
249	KEY
250	MONITOR
251	USING
252	UNTIL
253	WHILE

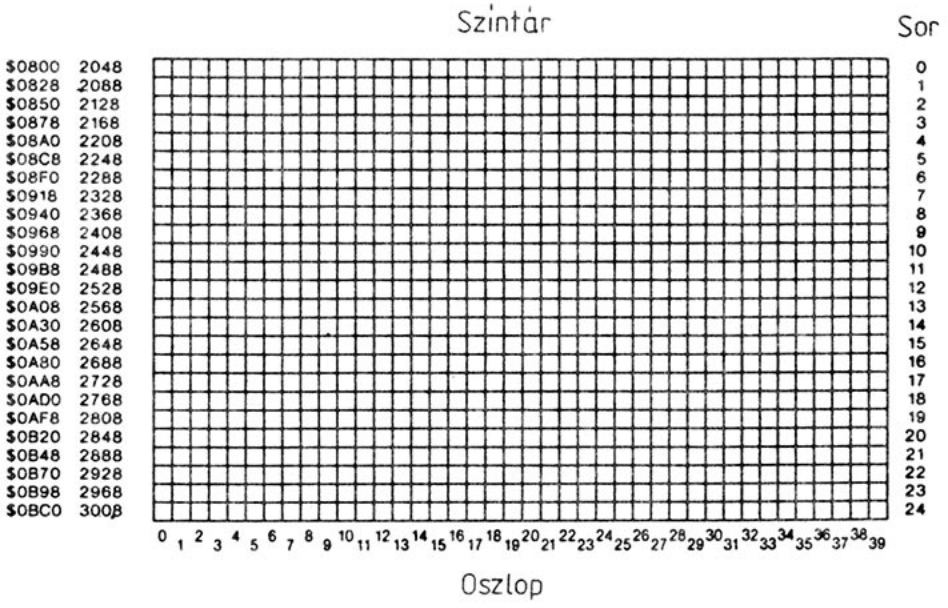
E FÜGGELÉK

A képernyő felépítése

A következő két ábrán a képernyőtárat és a színtárat mutatjuk be. Az első ábrán a képernyőtár látható. A bal oldalon levő számok az illető sorok kezdőcímét jelentik. A második ábrán a színtár látható.



E.1. ábra A képernyőtár címterülete



E.2. ábra A színtár címterülete

F FÜGGELÉK

A színek kialakítása a COLOR és a POKE utasításokkal

A COLOR utasításnál először a színmezőt kell meghatározni. Ehhez a 0...4 közötti számok használhatók. Az egyes számok jelentése:

Színmező	Jelentése
0	Háttér színe
1	Karakter színe
2	Többszínű üzemmód, 1. szín
3	Többszínű üzemmód, 2. szín
4	Képernyőkeret színe

A színeket az 1...16 közötti számokkal adjuk meg. Az egyes számokhoz tartozó színek:

Szám	Szín	Szám	Szín
1	Fekete	9	Narancs
2	Fehér	10	Barna
3	Piros	11	Sárgászöld
4	Türkiz	12	Rózsaszín
5	Bíbor	13	Kékeszöld
6	Zöld	14	Világoskék
7	Kék	15	Sötétkék
8	Sárga	16	Világoszöld

A COLOR utasítás harmadik paraméterével a színek fényessége állítható be. Ez a paraméter a 0...7 közötti értékeket veheti fel. A 0 a legsötétebb, a 7 a legvilágosabb tónust jelenti, a többi érték pedig a kettő közötti árnyalatokat.

Ha a színtárba közvetlenül, POKE utasítással írunk, akkor a COLOR-nál megadott értékek nem érvényesek. A színre vonatkozó értékek most 0–255 között lehetnek. A fényesség értékét külön nem kell megadni, mert ezt már maga a „színérték" tartalmazza.

A színek és a hozzájuk tartozó értékek:

A szín száma	Szín	Fényességi fokozat							
		0	1	2	3	4	5	6	7
1	Fekete	0	16	32	48	64	80	96	112
2	Fehér	1	17	33	49	65	81	97	113
3	Piros	2	18	34	50	6	82	98	114
4	Türkiz	3	19	35	51	67	83	99	115
5	Bíbor	4	20	36	52	68	84	100	116
6	Zöld	5	21	37	53	69	85	101	117
7	Kék	6	22	38	54	70	86	102	118
8	Sárga	7	23	39	55	71	87	103	119
9	Narancs	8	24	40	56	72	88	104	120
10	Barna	9	25	41	57	73	89	105	121
11	Sárgászöld	10	26	42	58	74	90	106	122
12	Rózsaszín	11	27	43	59	75	91	107	123
13	Kékeszöld	12	28	44	60	76	92	108	124
14	Világoskék	13	29	45	61	77	93	109	125
15	Sötétkék	14	30	46	62	78	94	110	126
16	Világoszöld	15	31	47	63	79	95	111	127

A feketének csak egy fényességi fokozata létezik. A többi, számított értéket csak a teljesség kedvéért adtuk meg.

Ha a kiválasztott értékhez hozzáadunk 128-at, akkor a képernyő megfelelő helyén levő karakter a kiválasztott színben és fényességi fokozatban villog.

G FÜGGELÉK

A CHR$ kódok

A következő táblázatokban bemutatjuk a Plus/4-es gép valamennyi CHR$ kódját. Az egyes kódokhoz tartozó jeleket úgy kapjuk meg, ha kiadjuk a CHR$ (kódszám) utasítást. Ezeknek a jeleknek nem mindegyike ábrázolható karakter, hanem vannak közöttük úgynevezett vezérlő karakterek is, mint pl. a RETURN. Az ábrázolható karakterek úgy jelennek meg a képernyőn, amint a táblázatban is láthatók, tehát a számítógépnek a bekapcsoláskor érvényes, nagybetű/grafika üzemmódjában. Ha átkapcsolunk a kisbetű/nagybetű üzemmódra (pl. a PRINT CHR$(14) utasítás kiadásával, vagy a SHIFT/COMMODORE-billentyűk lenyomásával), akkor a 65–90 közötti CHR$ kódokra a megfelelő kisbetűket kapjuk. A nagybetűknek ekkor a 97–122 közötti kódok felelnek meg, továbbá más grafikus karaktereket kapunk a 126, 127, 169, 186, 233, 250 és a 255 kódokra.

A szóköz (space) karakterhez két különböző kód tartozik: az egyik, a CHR$(32) a „normál" szóközhöz, míg a másik, a CHR$(160) az a SHIFT és a szóköz billentyű egyidejű lenyomásához.

A művelet fordítottja az ASC(„X"), ahol az X egy tetszőleges karakter, a művelet pedig az illető karakter CHR$ kódját adja. Ez a művelet programokban is igen jól használható, pl. ha valamely billentyűt akarunk vizsgálni.

A billentyűzet figyelésére van még egy érdekes lehetőség: a CTRL-billentyű és egy karakterbillentyű kombinációjával megkaphatjuk az 1...29 közötti CHR$ kódokat. Ez is jól használható a programokban. Ily módon egy sor újabb „funkcióbillentyűt" alakíthatunk ki. Pl. a CTRL/K billentyűkombináció a 11-es CHR$ kódot adja. A 6.1. táblázatban összefoglaltuk a lehetséges összes billentyűkombinációt, és a hozzájuk tartozó kódokat.

Még egy megjegyzés: Az úgynevezett idézőjel üzemmódban lehetőség van arra, hogy az 1...29 közötti CHR$ kódokat inverz betűkkel ábrázoljuk. Ehhez csak egy idézőjelet kell beírni, és ezután a kiválasztott CTRL/betű billentyűkombinációt. A 0, 10, 13, 19, 20 és a 27-es kódoknál ez egy kicsit körülményesebb: az illető inverz karaktert inverz üzemmódban kell a programba beírni.

CHR$	KARAKTER	CHR$	KARAKTER	CHR$	KARAKTER	CHR$	KARAKTER
0		32		64	@	96	
1		33	!	65	A $41	97	
2		34	"	66	B	98	
3		35	#	67	C	99	
4		36	$	68	D	100	
5	Fehér	37	%	69	E	101	
6		38	&	70	F	102	
7		39	'	71	G	103	
8	SHIFT/COMM.- átkapcs. letiltja	40	(	72	H	104	
9	SHIFT/COMM.- átkapcs. felold.	41	)	73	I	105	
10		42	*	74	J	106	
11		43	+	75	K	107	
12		44	,	76	L	108	
13	RETURN	45	-	77	M	109	
14	Kisbetük	46	.	78	N	110	
15		47	/	79	O	111	
16		48	0	80	P	112	
17	Kurzor le	49	1	81	Q	113	
18	Inverz mód be	50	2	82	R	114	
19	Home	51	3	83	S	115	
20	Delete	52	4	84	T	116	
21		53	5	85	U	117	
22		54	6	86	V	118	
23		55	7	87	W	119	
24		56	8	88	X	120	
25		57	9	89	Y	121	
26		58	:	90	Z	122	
27	Escape mód	59	;	91	[	123	
28	Piros	60	<	92	£	124	
29	Kurzor jobbra	61	=	93	]	125	
30	Zöld	62	>	94	↑	126	
31	Kék	63	?	95	←	127	

CHR$	KARAKTER	CHR$	KARAKTER	CHR$	KARAKTER	CHR$	KARAKTER
128		160		192		224	
129	Narancs	161		193		225	
130	Villogás be	162		194		226	
131		163		195		227	
132	Villogás ki	164		196		228	
133	F1	165		197		229	
134	F3	166		198		230	
135	F5	167		199		231	
136	F7	168		200		232	
137	F2	169		201		233	
138	F4	170		202		234	
139	F6	171		203		235	
140	F8	172		204		236	
141	SHIFT/RETURN	173		205		237	
142	Nagyb./kisbetű üzemmód	174		206		238	
143		175		207		239	
144	Fekete	176		208		240	
145	Kurzor fel	177		209		241	
146	Inverz ki	178		210		242	
147	CLR-kép.törl.	179		211		243	
148	INST-beszúrás	180		212		244	
149	Barna	181		213		245	
150	Sárgászöld	182		214		246	
151	Rózsaszín	183		215		247	
152	Kékeszöld	184		216		248	
153	Világoskék	185		217		249	
154	Sötétkék	186		218		250	
155	Világoszöld	187		219		251	
156	Bíbor	188		220		252	
157	Kurzor balra	189		221		253	
158	Sárga	190		222		254	
159	Türkiz	191		223		255	

CHR$	BILLENTYŰK	KARAKTER	FUNKCIÓ
0		@	
1	CTRL/A	A	
2	CTRL/B	B	
3	CTRL/C	C	
4	CTRL/D	D	
5	CTRL/E	E	Fehér
6	CTRL/F	F	
7	CTRL/G	G	
8	CTRL/H	H	SHIFT/COMMODORE-átkapcs.letiltja
9	CTRL/I	I	
10	CTRL/J		
11	CTRL/K	K	
12	CTRL/L	L	
13	CTRL/M		RETURN
14	CTRL/N	N	Átkapcs. kisbetű/nagyb. üzemmódra
15	CTRL/O	O	
16	CTRL/P	P	
17	CTRL/Q	Q	Kurzor le
18	CTRL/R	R	Inverz mod be
19	CTRL/S	S	Home
20	CTRL/T	T	Delete (törlés balra)
21	CTRL/U	U	
22	CTRL/V	V	
23	CTRL/W	W	
24	CTRL/X	X	
25	CTRL/Y	Y	
26	CTRL/Z	Z	
27	CTRL/:	[	Escape mód
28	CTRL/£	£	Piros
29	CTRL/;	]	Kurzor jobbra

H FÜGGELÉK

A Plus/4-es fontosabb tárcímei

Az első 4072 byte-ot BASIC nyelvű programok elhelyezésére nem használhatjuk. Ezeken a tárcímeken maga a gép operációs rendszere tárol értékeket, ezeket olvassa ill. változtatja. BASIC nyelvből a POKE utasítással lehet ezeket az értékeket megváltoztatni. Azt azonban figyelembe kell venni, hogy egy esetleges hibás POKE utasítással a számítógépet teljes mértékben működésképtelenné tehetjük. Ekkor gyakran már csak a RESET segíthet.

Hexadecimális	Decimális	Leírás
$0000	0	A processzor adatirány-regisztere
$0001	1	A processzor I/O kapuja
$0002	2	Token kódja kereséshez
$0003-0004	3-4	RENUMBER új kezdőcíme
$0005-0006	5-6	RENUMBER lépésköze
$0007	7	Keresett karakter/részeredmény
$0008	8	Idézőjel üzemmód kapcsolója/keresett karakterszámjegyek száma
$0009	9	TAB oszlopszámlálója
$000A	10	jelző: 0 = LOAD, 1 = VERIFY
$000B	11	Input-puffer mutatója, az elemek száma tokenizált hossz/dimenziók száma
$000C	12	DIM állapotjelzője (0 nem dimenzionált)
$000D	13	Adattípus: $FF = karakterlánc $00 = numerikus
$000E	14	Adattípus: $80 = egész, $00 = valós
$000F	15	DATA, LIST jelzője (idézőjelkapcsoló, szemétgyűjtéskapcsoló)
$0010	16	Elem jelzője, FNx jelzője
$0011	17	Jelző: $00 = INPUT, $40 = GET, $98 = READ
$0012	18	ATN előjel jelzője (ARG előjel)
$0013	19	INPUT prompt jelzője (?) (programstop)
$0014-0015	20-21	egész értékű cím, pl. sorszám
$0016	22	Átmeneti karakterlánc-verem mutatója $19 verem üres, $22 verem tele)
$0017-0018	23-24	Utolsó átmeneti karakterlánc-vektor
$0019-0021	25-33	Átmeneti karakterlánc-verem
$0022-0023	34-35	1. segédmutató területe
$0024-0025	36-37	2. segédmutató területe
$0026-002A	38-42	Szorzásnál szorzat munkaterülete
$002B-002C	43-44	BASIC programkezdet mutató

Hexadecimális	Decimális	Leírás
$002D-002E	45-46	Változóterület kezdete mutató/BASIC-program vége + 1
$002F-0030	47-48	Indexes változóterület (tömb) kezdet mutató
$0031-0032	49-50	Indexes változóterület vége + 1 mutató
$0033-0034	51-52	Karakterlánc változóterület vége mutató
$0035-0036	53-54	Karakterlánc változók segédmutatója
$0037-0038	55-56	BASIC programtár vége mutató
$0039-003A	57-58	Aktuális BASIC sor száma
$003B-003C	59-60	Aktuális BASIC byte, TXTPTR, CHRGET mutató
$003D-003E	61-62	BASIC CONT mutatója
$003F-0040	63-64	Aktuális DATA sor száma
$0041-0042	65-66	Aktuális DATA elem címe
$0043-0044	67-68	INPUT ugrásvektora
$0045-0046	69-70	Aktuális változó neve
$0047-0048	71-72	Aktuális változó címe
$0049-004A	73-74	FOR...NEXT ciklusváltozó mutatója
$004B-004C	75-76	BASIC mutatók ideiglenes tára
$004D	77	Összehasonlító jelek akkumulátora
$004E-0052	78-82	Munkaterület (mutatók stb.)
$0053	83	Grafika üzemmód
$0054-0056	84-86	Műveletek ugrásvektora
$0057-0060	87-96	Numerikus műveletek területe
$0061	97	1. FAC (lebegőpontos akkumulátor) kitevő
$0062-0065	98-101	1. FAC mantissza
$0066	102	1. FAC előjel
$0067	103	Polinomértékelés mutatója
$0068	104	1. FAC túlcsordulás
$0069-006E	105-110	2. FAC kitevő mantissza
$006F	111	1. FAC és 2. FAC előjelösszehasonlítás
$0070	112	1. FAC kerekítés
$0071-0072	113-114	Kazettapuffer hossza mutató
$0073-0074	115-116	AUTO utasítás sorszám növekménye 0 = KI
$0075	117	Grafikus terület foglaltság jelzője
$0076-0078	118-120	Munkaterület
$0079-007B	121-123	DS$ jellemzői
$007C-007D	124-125	BASIC pszeudo-veremmutató
$007E-008F	126-143	Munkaterület (hang)
$0090	144	ST állapotváltozó
$0091	145	STOP billentyű = 127/RVS billentyű
$0094	148	Soros busz kimeneti puffermutatója
$0095	149	Soros busz puffere
$0096	150	Átmeneti tár
$0097	151	Nyitott állományok (file-ok) száma
$0098	152	Beviteli (input) készülék (alapértelmezés 0)
$0099	153	Kiviteli (output) készülék (alapértelemezés 3)
$009A	154	Üzemmódjelző: $80 = közvetlen, $00 = programmód

Hexadecimális	Decimális	Leírás
$009B-009C	155-156	Kazettapuffer/scrolling mutató
$009D-009E	157-158	Program vége mutató
$009F-00A0	159-160	Átmeneti adattár
$00A1-00A2	161-162	Átmeneti adattár
$00A3-00A5	163-165	Valósidejű óra
$00A6	166	Soros busz regisztere
$00A7	167	Kazettarutin regisztere
$00A8	168	Soros busz regisztere
$00A9	169	Ideiglenes színvektor
$00AA	170	Kazettarutin regisztere
$00AB	171	Állománynév (file-név) karaktereinek száma
$00AC	172	Aktuális logikai állományszám
$00AD	173	Aktuális másodlagos cím
$00AE	174	Aktuális készülékszám
$00AF-00B0	175-176	Állománynév mutatója
$00B1	177	Hibaüzenet rutin területe
$00B2-00B3	178-179	I/O kezdőcím
$00B4-00B5	180-181	LOAD kezdőcím
$00B6-00B7	182-183	LOAD vége cím mutatója
$00BA-00B	186-187	Kazettapufferben levő karakter mutatója
00BE-00BFB	190-191	LONG FETCH rutin regisztere
$00C0-00C1	192-193	Képernyő görgetés (scrolling) regisztere
$00C2	194	Inverz (RVS) üzemmód jelzője
$00C3	195	Sor vége mutató INPUT-nál
$00C4-00C5	196-197	Kurzor pozíciója INPUT-nál sor/oszlop
$00C6	198	Lenyomott billentyű kódja ($40 = nincs lenyomva)
$00C7	199	Bevitel típusának jelzője (GET,INPUT)
$00C8-00C9	200-201	Kurzor képernyősorának mutatója
$00CA	202	Kurzor pozíciója az aktuális soron belül
$00CB	203	Idézőjel mód jelzője ($00 = KI, $0F = BE)
$00CC	204	Aktuális képernyősor hossza
$00CD	205	Kurzor sorának száma
$00CE	206	Utolsó karakter (I/O)
$00CF	207	Karakterek száma, beszúrás (INS) üzemmódban
$00D0-00E8	208-232	Programozók számára fenntartott szabad terület
$00E9	233	Munkaterület
$00EA-00EB	234-235	Képernyőszerkesztő (szín)
$00EC-00EE	236-238	Munkaterület (képernyő)
$00EF	239	Billentyűzetpufferben levő karakterek száma
$00F0	240	CTRL-S jelzője (különböző, ha 0 = prg. állj.)
$00F1-00F2	241-242	Monitor regisztere
$00F5	245	Vizsgálóösszeg regisztere
$00FA	250	X regisztere a STOP vizsgálatnál
$00FB	251	Aktuális BANK konfiguráció (memórialapozáshoz)
$00FE	254	Munkaregiszter (képernyőszerkesztő)
$0100-01FF	256-511	Processzor veremtár

Hexadecimális	Decimális	Leírás
$0200-0258	512-600	Input puffer
$0259-025C	601-604	BASIC puffer
$025D-02AC	605-684	BASIC DOS munkaterület
$025D	605	DOS ciklusszámláló
$025E-026D	606-621	Állománynév területe
$026E	622	1. állománynév (hossz)
$026F	623	DOS, 1. lemezegység száma
$0270-0271	624-625	1. állománynév (cím)
$0272	626	2. állománynév (hossz)
$0273	627	DOS, 2. lemezegység száma
$0274-0275	628-629	2. állománynév (cím)
$0276	630	DOS logikai állományszám
$0277	631	DOS eszközszám
$0278	632	DOS másodlagos cím
$0279-027A	633-634	DOS lemez ID
$027B	635	ID jelző
$027C	636	DOS Parancspuffer
$027D-02AC	637-684	DOS munkatár
$02AD-02B0	685-688	Grafikus kurzor
$02B1-02B4	689-692	Grafikus kurzor (regiszter)
$02B5-$02CB	693-715	Grafika által használt regiszterek
$02CC-02E3	716-739	PRINT USING, grafika munkatár
$02E4	740	Karakter ROM címének felső byte-ja
$02EB	747	TRACE jelző ($00 = KI, nagyobb, mint $80 = BE)
$02F0	752	Grafikus paraméterek száma
$02F1	753	Paraméter, relatív = 1, abszolút = 0
$02F2-02F3	754-755	Valós-egész konverzió mutató
$02F4-02F5	756-757	Egész-valós konverzió mutató
$0300-0301	768-769	Ugrásvektor: hibaüzenet ($8686)
$0302-0303	770-771	Ugrásvektor: BASIC melegindítás ($8712)
$0304-0305	772-773	Ugrásvektor: tokenizálás ($8956)
$0306-0307	774-775	Ugrásvektor: kulcsszóelőállítás ($8B6E)
$0308-0309	776-777	Ugrásvektor: főciklus ($8BD6)
$030A-030B	778-779	Ugrásvektor: kiértékelés ($9417)
$030C-030D	780-781	Ugrásvektor: parancsfeldolgozás ($896A)
$030E-030F	782-783	Ugrásvektor: kulcsszóelőállítás ($8B88)
$0310-$0311	784-785	Ugrásvektor: User token végrehajtása ($8C8B)
$0312-0313	786-787	Ugrásvektor: megszakítás (raszter) ($CE42)
$0314-0315	788-789	Ugrásvektor: IRQ megszakítás ($CE0E)
$0316-0317	790-791	Ugrásvektor: BRK megszakítás ($F44C)
$0318-0319	792-793	Ugrásvektor: OPEN ($FF5B)
$031A-031B	794-795	Ugrásvektor: CLOSE ($EE5D)
$031C-031D	796-797	Ugrásvektor: input csatorna definiálása ($ED18)
$031E-$031F	798-799	Ugrásvektor: output csatorna definiálása ($ED60)
$0320-0321	800-801	Ugrásvektor: I/O csat. visszaállítása ($EF0C)
$0322-0323	802-803	Ugrásvektor: INPUT rutin ($EBE8)

Hexadecimális	Decimális	Leírás
$0324-0325	804-805	Ugrásvektor: kiíró rutin ($EC4B)
$0326-0327	806-807	Ugrásvektor: STOP lekérdező rutin ($F265)
$0328-0329	808-809	Ugrásvektor: GETIN rutin ($EBD9)
$032A-032B	810-811	Ugrásvektor: összes állomány lezárása ($EF0B)
$032C-032D)	812-813	Ugrásvektor: Monitor BREAK ($F44C)
$032E-032F	814-815	Ugrásvektor: LOAD rutin ($F04A)
$0330-0331	816-817	Ugrásvektor: SAVE rutin ($F1A4)
$0332-03F2	818-1010	Kazettapuffer
$03F3-03F4	1011-1012	Adatszámláló szalagra íráskor
$03F5-03F6	1013-1014	Adatszámláló szalagról olvasáskor
$03F7-0436	1015-1078	RS-232 input puffer (64 byte)
$0437-0454	1079-1108	Rendszer munkaterület
$0455-0472	1109-1138	Rendszer munkaterület
$0473-0478	1139-1144	CHRGET rutin
$0479-0484	1145-1156	CHRGOT rutin
$0494-04E6	1172-1254	Memórialapozó rutinok (Bankswitching)
$04E7-04EA	1255-1258	PRINT USING paraméterek
$04EF-04F6	1263-1270	TRAP és Error (hiba) jelzők
$0500-0502	1280-1282	USER ugró utasítás
$0503-0507	1283-1287	RND induló értéke
$0508	1288	Hideg/melegindítás jelző
$0509-0512	1289-1298	Logikai állományszámok táblázata
$0513-051C	1299-1308	Készülékszámok táblázata
$051D-0526	1309-1318	Másodlagos címek táblázata
$0527-0530	1319-1328	Billentyűzet puffer
$0531-0532	1329-1330	Szabad RAM kezdőcíme (MEMBOT)
$0533-0534	1331-1332	Szabad RAM végcíme (MEMTOP)
$0539	1337	Kazettapuffer mutató
$053A	1338	Kazettaállomány típusa
$0540	1344	Billentyűismétlés jelzője: $80 = mindegyik, $40 = egyik sem, $00 = SPC, DEL, CURSOR
$0541-542	1345-1346	Billentyűismétlés számlálója
$0543	1347	SHIFT jelző
$0544	1348	utolsó SHIFT karakter
$0545-0546	1349-1350	Billentyűzet dekóder ugrásvektor
$0547	1351	Szöveg/grafika üzemmód jelzője
$0548	1352	Scroll (görgetés) jelző
$054B-055C	1355-1372	CPU munkaterület
$0552-0557	1362-1367	CPU regiszterek
$0558-055C	1368-1372	CPU munkaterület
$055D	1373	Funkcióbillentyűk számlálója
$055E	1374	Funkcióbillentyűk szövegmutatója
$055F-05E6	1375-1510	Funkcióbillentyűk tárterülete
$05EC-$06EB	1516-1771	ROM BANK rutin
$05EC-$05EF	1516-1519	ROM BANK jelzőkódok és rutinszámok táblázata
$05F0-05F1	1520-1521	LONG JUMP (cím)
$05F2	1522	LONG JUMP (akkumulátor)

Hexadecimális	Decimális	Leírás
$05F3	1523	LONG JUMP (X regiszter)
$05F4	1524	LONG JUMP (állapotregiszter)
$05F5-065D	1525-1629	Memórialapozás RAM területe
$065E-06EB	1630-1771	Felhasználói, szabad RAM terület
$06EC-07AF	1772-1967	BASIC pszeudo-verem
$07B0-07CC	1968-1996	Kazetta munkaterület
$07CD-07D0	1997-2000	RS-232 munkaterület
$07D1	2001	RS-232 mutató az input puffer kezdetére
$07D2	2002	RS-232 mutató az input puffer végére
$07D3	2003	Input pufferben levő karakterek száma
$07E5	2021	Képernyő alsó széle
$07E6	2022	Képernyő felső széle
$07E7	2023	Képernyő bal oldali széle
$07E8	2024	Képernyő jobb oldali széle
$07E9	2025	Scroll $00 = BE $80 = KI
$07EA	2026	Automatikus beszúrás $00 = KI, $80-tól = BE
$07EB	2027	ESC le lett nyomva? $27 = IGEN
$07EE-07F1	2030-2033	Folytató sorok bittérképe
$07F2	2034	A regiszter mentése SYS-nél
$07F3	2035	Az X regiszter mentése SYS-nél
$07F4	2036	Az Y regiszter mentése SYS-nél
$07F5	2037	Állapotregiszter mentése SYS-nél
$07F6	2038	Billentyűzetlekérdezés regisztere
$07F7	2039	CTRL-S jelző: $00 = nyitva $06 = zárva
$07F8	2040	Monitor RAM/ROM átkapcsolás: $00 = ROM, $80 = RAM
$07FC	2044	Kazettamotor kapcsoló
$0800-0BFF	2048-3071	Színtár (szöveg)
$0C00-0FE7	3072-4071	Képernyőtár (szöveg)

A BASIC RAM elhelyezkedése a grafika meghívása előtt:

$1000-FCFF	4096-64767	A Plus/4-es BASIC RAM területe

A RAM felosztása a grafika meghívása után:

$4000-FCFF	16384-64767	A Plus/4-es BASIC RAM-ja
$1800-1BFF	6144-7167	Fényerőtáblázat (grafika)
$1C00-1FFF	7168-8191	Színtáblázat (grafika)
$2000-3FFF	8192-16383	Grafikus képernyő

I FÜGGELÉK

A 7360-as chip (TED) fontosabb regiszterei

A Plus/4-esbe a gyártók egy új chipet építettek be, amelynek a típusjele 7360, neve TED (TExt Display). Ez a chip igen nagy teljesítményű: feladatai közé tartozik a tárkezelés (max. 64 kbyte), a kép és a hang előállítása, valamint a billentyűzet és a botkormányok figyelése.

A TED báziscíme (kezdőcíme) $FF00 (65280). Ezután egy sor regiszter következik, amelyek közül a legfontosabbakat ismertetjük.

$FF00 65280	1. időzítő, alsó byte	
$FF01 65281	1. időzítő, felső byte	
$FF02 65282	2. időzítő, alsó byte	
$FF03 65283	2. időzítő, felső byte	
$FF04 65284	3. időzítő, alsó byte	
$FF05 65285	3. időzítő, felső byte	
$FF06 65286	0–2. bit:	képernyő tartalom függőleges pozíciója (lehetővé teszi a képernyő eltolását 8 képponttal)
	3. bit:	0 = 24 képernyősor, 1 = 25 képernyősor
	4. bit:	0 = képernyő kikapcsolva 1 = képernyő bekapcsolva
	5. bit:	0 = finomfelbontás bekapcsolva 1 = finomfelbontás kikapcsolva
	6. bit:	0 = bővített háttérszín mód kikapcsolva 1 = bővített háttérszín mód bekapcsolva
	7. bit:	mindig 0
$FF07 65287	0–2. bit:	a képernyő vízszintes pozíciója
	3. bit:	0 = 38 oszlop, 1 = 40 oszlop
	4. bit:	0 = többszínű üzemmód kikapcsolva 1 = többszínű üzemmód bekapcsolva
	5. bit:	1 = képernyő vízszintes pozíciójának befagyasztása
	6. bit:	0 = PAL-rendszer (tv) 1 = NTSC-rendszer (tv)
	7. bit:	0 = inverz ábrázolást engedélyezi 1 = inverz ábrázolást letiltja
$FF08 65288	Billentyűzetállapot (latch)	
$FF09 65289	Megszakításérzékelő regiszter	
	0. bit:	nem használt
	1. bit:	1 = rasztermegszakítás
	2. bit:	nem használt (fényceruza megszakítás helye)
	3. bit:	1 = 1. időzítő alulcsordulás megszakítás
	4. bit:	1 = 2. időzítő alulcsordulás megszakítás

	5. bit:	nem használt
	6. bit:	1 = 3. időzítő alulcsordulás megszakítás
	7. bit:	megszakítás megtörtént jelzőbit
$FF0A 65290	0. bit:	a $FF0B (65291) regiszter 8. bitje, a többi bit szerepe ugyanaz, mint a 65289-es regiszternél. Minimum 1 bit megegyezésekor létrejön egy megszakítás (IRQ).
$FF0B 65291	0–7. bit:	rasztersor (8. bit) a 65290-es regiszterben
$FF0C 65292	0. bit:	a hardver-kurzor 8. bitje
	1. bit:	a hardver-kurzor 9. bitje
	2–7. bit:	nem használt
$FF0D 65293	0–7. bit:	a hardver-kurzor bitjei
$FF0E 65294	0–7. bit:	1. hanggenerátor frekvencia
$FF0F 65295	0–7. bit:	2. hanggenerátor frekvencia
$FF10 65296	0. bit:	2. hanggenerátor frekvencia 8. bitje
	1. bit:	2. hanggenerátor frekvencia 9. bitje
	2–7. bit:	nem használt
$FF11 65297	0–3. bit:	hangerő
	4. bit:	1. hanggenerátor hang be/kikapcsolása
	5. bit:	2. hanggenerátor hang be/kikapcsolása
	6. bit:	2. hanggenerátor (zaj) be/kikapcsolása
	7. bit:	1 = hanggenerátorok bekapcsolási állapotának visszaállítása
$FF12 65298	0. bit:	1. hanggenerátor frekvencia 8. bitje
	1. bit:	1. hanggenerátor frekvencia 9. bitje
	2. bit:	1 = karakterkészlet a ROM-ban van 0 = karakterkészlet a RAM-ban van
	3–5. bit:	bittérkép báziscíme
	6–7. bit:	nem használt
$FF13 65299	0. bit:	állapot, 1 = ROM
	1. bit:	single clock
	2–7. bit:	karaktergenerátor kezdőcíme
$FF14 65300	0–2. bit:	nem használt
	3–7. bit:	képernyő és színtár kezdőcíme
$FF15 65301	0. háttérszínregiszter	
	0–3. bit:	szín
	4–6. bit:	fényesség
	7. bit:	nem használt
$FF16 65302	1. háttérszínregiszter bitkiosztás, mint a $FF15 regiszternél	
$FF17 65303	2. háttérszínregiszter bitkiosztás, mint a $FF15 regiszternél	
$FF18 65304	3. háttérszínregiszter bitkiosztás, mint a $FF15 regiszternél (keretszín)	
$FF19 65305	4. háttérszínregiszter bitkiosztás, mint a $FF15 regiszternél	

K FÜGGELÉK

A csatlakozók lábkiosztása

A Plus/4-esen egy sor csatlakozó van, amelyekhez különböző külső készülékek csatlakoztathatók. A következőkben bemutatjuk az egyes csatlakozók lábkiosztását:

- Adatmagnó (Datasette)
- Audio/video csatlakozó
- Botkormány
- Soros interface
- Bővítő csatlakozó
- Felhasználói csatlakozó

Adatmagnó (Datasette)

Érintkező	Foglaltság
1	Föld
2	+5 V
3	Motor
4	Olvasás
5	Írás
6	Kapcsoló
7	Föld

Datasette

Audio/video csatlakozó

Érintkező	Foglaltság
1	Fényesség
2	Föld
3	Audio kimenet
4	Video kimenet
5	Audio bemenet
6	Szín kimenet
7	+5 V
8	Üres

Audio/Video

1. botkormány

Érintkező	Foglaltság
1	Key 0
2	Key 1
3	Key 2
4	Key 3
5	+5 V
6	Tűzgomb, Key 6
7	Föld
8	D2

2. botkormány

Érintkező	Foglaltság
1	Key 0
2	Key 1
3	Key 2
4	Key 3
5	+5 V
6	Tűzgomb, Key 7
7	Föld
8	D1

Joystick

Soros interface

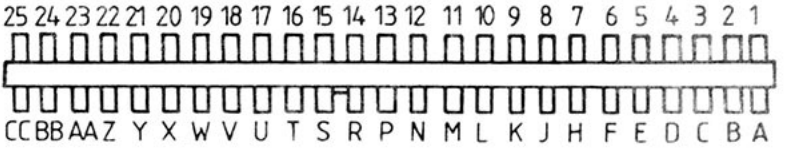
Érintkező	Foglaltság
1	Service Request
2	Föld
3	ATN
4	ÓRA
5	Adat
6	RESET

Bővítő csatlakozó (Expansion Port)

Érintkező	Foglaltság	Érintkező	Foglaltság
1	Föld	A	Föld
2	+5 V	B	C1 alsó
3	+5 V	C	BRESET
4	IRQ	D	RAS
5	Olvasás/írás	E	I0
6	C1 felső	F	A15
7	C2 alsó	H	A14
8	C2 felső	J	A13
9	CS1	K	A12
10	CS0	L	A11
11	CAS	M	A10
12	MUX	N	A9
13	BA	P	A8
14	D7	R	A7
15	D6	S	A6
16	D5	T	A5
17	D4	U	A4
18	D3	V	A3
19	D2	W	A2
20	D1	X	A1

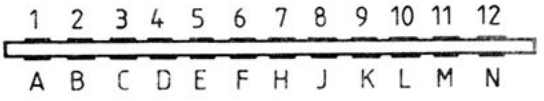
Seriell

Érintkező	Foglaltság	Érintkező	Foglaltság
21	D0	Y	A0
22	AEC	Z	Üres
23	Külső audio	AA	Üres
24	02	BB	Üres
25	Föld	CC	Föld



Felhasználói csatlakozó (User Port)

Érintkező	Foglaltság	Érintkező	Foglaltság
1	Föld	A	Föld
2	+5 V	B	P0
3	BRESET	C	RxD
4	P2	D	RTS
5	P3	E	DTR
6	P4	F	P7
7	P5	H	DCD
8	RxC	J	P6
9	ATN	K	P1
10	9 V AC	L	DSR
11	9 V AC	M	TxD
12	Föld	N	Föld



L FÜGGELÉK

Átváltási táblázat a hexadecimális, decimális és bináris számok között

Az alábbi táblázat segítségével a számokat könnyen átválthatjuk a leggyakrabban használatos számrendszerek egyikéből a másikába. A bináris számoknál a jobb szélső helyen a 0. bit áll. Ettől balra következik rendre az 1. bit, a 2. bit..., 7. bit.

Dec.	$Hex.	%Bináris	Dec.	$Hex.	%Bináris	Dec.	$Hex.	%Bináris
0	$00	%00000000	33	$21	%00100001	66	$42	%01000010
1	$01	%00000001	34	$22	%00100010	67	$43	%01000011
2	$02	%00000010	35	$23	%00100011	68	$44	%01000100
3	$03	%00000011	36	$24	%00100100	69	$45	%01000101
4	$04	%00000100	37	$25	%00100101	70	$46	%01000110
5	$05	%00000101	38	$26	%00100110	71	$47	%01000111
6	$06	%00000110	39	$27	%00100111	72	$48	%01001000
7	$07	%00000111	40	$28	%00101000	73	$49	%01001001
8	$08	%00001001	41	$29	%00101001	74	$4A	%01001010
9	$09	%00001001	42	$2A	%00101010	75	$4B	%01001011
10	$0A	%00001010	43	$2B	%00101011	76	$4C	%01001100
11	$0B	%00001011	44	$2C	%00101100	77	$4D	%01001101
12	$0C	%00001100	45	$2D	%00101101	78	$4E	%01001110
13	$0D	%00001101	46	$2E	%00101110	79	$4F	%01001111
14	$0E	%00001110	47	$2F	%00101111	80	$50	%01010000
15	$0F	%00001111	48	$30	%00110000	81	$51	%01010001
16	$10	%00010000	49	$31	%00110001	82	$52	%01010010
17	$11	%00010001	50	$32	%00110010	83	$53	%01010011
18	$12	%00010010	51	$33	%00110011	84	$54	%01010100
19	$13	%00010100	52	$34	%00110100	85	$55	%01010101
20	$14	%00010100	53	$35	%00110101	86	$56	%01010110
21	$15	%00010101	54	$36	%00111011	87	$57	%01010111
22	$16	%00010110	55	$37	%00110111	88	$58	%01011000
23	$17	%00010111	56	$38	%00111000	89	$59	%01011001
24	$18	%00011000	57	$39	%00111001	90	$5A	%01011010
25	$19	%00011001	58	$3A	%00111010	91	$5B	%01011011
26	$1A	%00011010	59	$3B	%00111011	92	$5C	%01011100
27	$1B	%00011011	60	$3C	%00111100	93	$5D	%01011101
28	$1C	%00011100	61	$3D	%00111101	94	$5E	%01011110
29	$1D	%00011101	62	$3E	%00111110	95	$5F	%01011111
30	$1E	%00011110	63	$3F	%00111111	96	$60	%01100000
31	$1F	%00011111	64	$40	%01000000	97	$61	%01100001
32	$20	%00100000	65	$41	%01000001	98	$62	%01100010

Dec.	$Hex.	%Binários
99	$63	%01100011
100	$64	%01100100
101	$65	%01100101
102	$66	%01100110
103	$67	%01100111
104	$68	%01101000
105	$69	%01101001
106	$6A	%01101010
107	$6B	%01101011
108	$6C	%01101100
109	$6D	%01101101
110	$6E	%01101110
111	$6F	%01101111
112	$70	%01110000
113	$71	%01110001
114	$72	%01110010
115	$73	%01110011
116	$74	%01110100
117	$75	%01110101
118	$76	%01110110
119	$77	%01110111
120	$78	%01111000
121	$79	%01111001
122	$7A	%01111010
123	$7B	%01111011
124	$7C	%01111100
125	$7D	%01111101
126	$7E	%01111110
127	$7F	%01111111
128	$80	%10000000
129	$81	%10000001
130	$82	%10000010
131	$83	%10000011
132	$84	%10000100
133	$85	%10000101
134	$86	%10000110
135	$87	%10000111
136	$88	%10001000
137	$89	%10001001
138	$8A	%10001010
139	$8B	%10001011
140	$8C	%10001100
141	$8D	%10001101
142	$8E	%10001110
143	$8F	%10001111
144	$90	%10010000
145	$91	%10010001
146	$92	%10010010
147	$93	%10010011
148	$94	%10010100
149	$95	%10010101
150	$96	%10010110
151	$97	%10010111
152	$98	%10011000
153	$99	%10011001
154	$9A	%10011010
155	$9B	%10011011
156	$9C	%10011100
157	$9D	%10011101
158	$9E	%10011110
159	$9F	%10011111
160	$A0	%10100000
161	$A1	%10100001
162	$A2	%10100010
163	$A3	%10100011
164	$A4	%10100100
165	$A5	%10100101
166	$A6	%10100110
167	$A7	%10100111
168	$A8	%10101000
169	$A9	%10101001
170	$AA	%10101010
171	$AB	%10101011
172	$AC	%10101100
173	$AD	%10101101
174	$AE	%10101110
175	$AF	%10101111
176	$B0	%10110000
177	$B1	%10110001
178	$B2	%10110010
179	$B3	%10110011
180	$B4	%10110100
181	$B5	%10110101
182	$B6	%10110110
183	$B7	%10110111
184	$B8	%10111000
185	$B9	%10111001
186	$BA	%10111010
187	$BB	%10111011
188	$BC	%10111100
189	$BD	%10111101
190	$BE	%10111110
191	$BF	%10111111
192	$C0	%11000000
193	$C1	%11000001
194	$C2	%11000010
195	$C3	%11000011
196	$C4	%11000100
197	$C5	%11000101
198	$C6	%11000110
199	$C7	%11000111
200	$C8	%11001000
201	$C9	%11001001
202	$CA	%11001010
203	$CB	%11001011
204	$CC	%11001100
205	$CD	%11001101
206	$CE	%11001110
207	$CF	%11001111
208	$D0	%11010000
209	$D1	%11010001
210	$D2	%11010010
211	$D3	%11010011
212	$D4	%11010100
213	$D5	%11010101
214	$D6	%11010110
215	$D7	%11010111
216	$D8	%11011000
217	$D9	%11011001
218	$DA	%11011010
219	$DB	%11011011
220	$DC	%11011100
221	$DD	%11011101
222	$DE	%11011110
223	$DF	%11011111
224	$E0	%11100000
225	$E1	%11100001
226	$E2	%11100010
227	$E3	%11100011
228	$E4	%11100100
229	$E5	%11100101
230	$E6	%11100110
231	$E7	%11100111
232	$E8	%11101000
233	$E9	%11101001
234	$EA	%11101010
235	$EB	%11101011
236	$EC	%11101100

Dec.	$Hex.	%Bináris
237	$ED	%11101101
238	$EE	%11101110
239	$EF	%11101111
240	$F0	%11110000
241	$F1	%11110001
242	$F2	%11110010
243	$F3	%11110011

Dec.	$Hex.	%Bináris
244	$F4	%11110100
245	$F5	%11110101
246	$F6	%11110110
247	$F7	%11110111
248	$F8	%11111000
249	$F9	%11111001
250	$FA	%11111010

Dec.	$Hex.	%Bináris
251	$FB	%11111011
252	$FC	%11111100
253	$FD	%11111101
254	$FE	%11111110
255	$FF	%11111111

M FÜGGELÉK

A 7501-es mikroprocesszor gépi kódú utasításai

A táblázatban használt rövidítések:

közv	közvetlen címzés
absz	abszolút címzés
zp	nullás lapos (zero page) címzés
akku	akkumulátoros címzés
impl	beleértett (implied) címzés
indX	indexelt, indirekt címzés
indY	indirekt, indexelt címzés
z,X	nullás lapos, X regiszterrel indexelt címzés
ab,Y	abszolút, X regiszterrel indexelt címzés
rel	relatív címzés
ind	indirekt címzés
z,Y	nullás lapos, Y regiszterrel indexelt címzés
A	akkumulátor
M	tár (Memory)
SP	veremmutató (Stack-Pointer)
S	veremtár (Stack)
SR	állapotregiszter (Status-Register)

Utasítás/hatás	közv	bsz	zp	akku	impl	indX	indY	z,X	ab,X	ab,Y	rel	ind	z,Y	NZCIDV
Adatátviteli utasítások														
LDA M→akku	A9	AD	A5			A1	B1	B5	BD	B9				xx
STA akku→M		8D	85			81	91	95	9D	99				
LDX M→X	A2	AE	A6							BE			B6	xx
STX X→M		8E	86										96	
LDY M→Y	A0	AC	A4					B4	BC					xx
STY Y→M		8C	84					94						
TAX akku→X					AA									xx
TAY akku→Y					A8									xx
TXA X→akku					8A									xx
TYA Y→akku					98									xx
TXS X→SP					9A									
TSX SP→X					BA									xx
PLA S→akku					68									xx
PHA akku→S					48									
PLP S→SR					28									S-től függ
PHP SP→S					08									
Állapotregiszterre vonatkozó utasítások														
CLC C−bit = 0					18									0
SEC C−bit = 1					38									1
CLD D−bit = 0					D8									0
SED D−bit = 1					F8									1
CLI I−bit = 0					58									0
SEI I−bit = 1					78									1
CLV V−bit = 0					B8									0

Utasítás/hatás		közv	bsz	zp	akku	impl	indX	indY	z,X	ab,X	ab,Y	rel	ind	z,Y	NZCIDV
Aritmetikai utasítások															
ADC	A = A + M + C	69	6D	65			61	71	75	7D	79				xxx x
SBC	A = A − M − C	E9	ED	E5			E1	F1	F5	FD	F9				xxx x
INC	M = M + 1		EE	E6					F6	FE					xx
DEC	M = M − 1		CE	C6					D6	DE					xx
INX	X = X + 1					E8									xx
DEX	X = X − 1					CA									xx
INY	Y = Y + 1					C8									xx
DEY	Y = Y − 1					88									xx
Logikai utasítások															
AND	A = A AND M	29	2D	25			21	31	35	3D	39				xx
ORA	A = A OR M	09	0D	0D			01	11	15	1D	19				xx
EOR	A = A EOR M	49	4D	45			41	51	55	5D	59				xx
Elágazó utasítások															
BCC	elágazás, ha C = 0											90			
BCS	elágazás, ha C = 1											B0			
BNE	elágazás, ha Z = 0											D0			
BEQ	elágazás, ha Z = 1											F0			
BPL	elágazás, ha N = 0											10			
BMI	elágazás, ha N = 1											30			
BVC	elágazás, ha V = 0											50			
BVS	elágazás, ha V = 1											70			
Ugróutasítások															
JMP	ugrás	4C											6C		
JSR	ugrás szubr.-ra	20													
RTS	vissza szubr.-ból					60									
RTI	vissza megsz.-ból					40									

Utasítás/hatás	közv	bsz	zp	akku	impl	indX	indY	z,X	ab,X	ab,Y	rel	ind	z,Y	NZCIDV	
Összehasonlító utasítások															
CMP A és M	C9	CD	C5			C1	D1	D5	DD	D9				xxx	
CPX X és M	E0	EC	E4											xxx	
CPY Y és M	C0	CC	C4											xxx	
BIT A AND M		2C	24											7x	6
Eltoló/elforgató utasítások															
ASL		0E	06	0A				16	1E					xxx	
LSR		4E	46	4A				56	5E					xxx	
ROL		2E	26	2A				36	3E					xxx	
ROR		6E	66	6A				76	7E					xxx	
Egyéb utasítások															
NOP No Operation (üres utasítás)					EA										1
BRK szoftveroldali megszakítás					00										

N FÜGGELÉK

Számítástechnikai kislexikon

A számítástechnikában – és így a mi könyvünkben is – számos olyan, speciális kifejezés fordul elő, amelyek a kezdők számára eleinte nehézséget okozhatnak. Ebben a függelékben arra törekszünk, hogy a könyvben előforduló fogalmakat és jelöléseket érthetővé tegyük.

Reméljük, hogy ez az összeállításunk a későbbi munka során is mindenki hasznára lesz. A számítástechnika legfontosabb fogalmait ezért ábécé sorrendbe szedtük.

Adatállomány

Az egymással összefüggő adatokat adatállománynak (eredeti néve file-nak, ejtsd: fájl) nevezzük.

Alfanumerikus

Betűk és számok, azaz a különleges jeleket kivéve valamennyi karakter.

Állandók

A számítógépbe megváltoztathatatlanul beírt érték (pl. a pi = 3,14159265).

ASCII

Az American Standard Code for Information Interchange kifejezés rövidítése. A számítógépeknél használatos karakterek szabványosított kódja (CHR$). A Plus/4-esnél ezek a kódok részben eltérnek, amit figyelembe kell venni, ha a géphez nem Commodore-típusú nyomtatót csatlakoztatunk.

Assembler

Fordítóprogram, amely az assembly nyelvű programokat gépi kódra fordítja le. Lásd TEDMON is.

Az @ karakter (at)

A Plus/4-es gépnél az @ karakter teszi lehetővé, hogy a kazettán vagy lemezen meglévő programjainkat a már meglévő nevükön felülírjuk. Így egy korábbi programunk javított változatát az előzővel azonos néven tárolhatjuk.

BASIC

A Beginner1s All-Purpose Symbolic Instruction Code rövidítése. Magas szintű, kezdők által is könnyen megtanulható programozási nyelv. Fejlődése során számos változata alakult ki.

Baud

A soros adatátvitel sebességének mértékegysége. A Baud a másodpercenként átvitt bitek számát adja meg.

Bit

A binary digit (2-es számjegy) rövidítése. Ez a számítógép által feldolgozható információ legkisebb egysége. Értéke 1 vagy 1 lehet.

Busz

Vezetékrendszer, amelyen keresztül a számítógépen belüli egységek között az adatátvitel bonyolódik. Vannak cím-, adat- és rendszerbuszok.

Byte

Egy byte 8 bitből áll. A byte-okat általában hexadecimális számrendszerben ábrázolják, amelyben a „számjegyek" 0-tól F-ig terjednek. Egy kbyte = 1024 byte, azaz 210 byte. 1 Mbyte (Megabyte) = 1024 kbyte, azaz 1048576 byte.

Centronics-interface

Nyomtatók párhuzamos adatbevitelre alkalmas csatlakozója.

Compiler

Fordítóprogram, amely egy magasszintű programozási nyelven megírt programot gépi kódú nyelvre fordít le. Az így lefordított programok futtatásához – az interpretertől eltérően – magára a fordítóprogramra nincs szükség.

CPU

A Central Processing Unit rövidítése. Jelentése magyarul központi végrehajtó egység. A házi és a személyi számítógépeknél a mikroprocesszort értjük alatta.

Directory

Egy mágneslemez tartalomjegyzéke.

Disassembler

Fordítóprogram, amely a gépi kódú programot assembly nyelvű utasításokra (mnemonikokra) és a hozzá tartozó adatokra fordítja vissza.

DOS

A Disc Operating System (magyarul: lemezműveleteket vezérlő rendszer) rövidítése.

Editor

Szerkesztőprogram. A Plus/4-es számítógépnek ún. képernyőorientált szerkesztőprogramja van, ami azt jelenti, hogy a kurzorvezérlő billentyűkkel a képernyő bármely részére állhatunk, ott adatokat vihetünk be, és azokat javíthatjuk.

EPROM

Az Erasable Programable Read Only Memory (magyarul = törölhető, programozható, csak olvasható memória) rövidítése. A memória tartalma ibolyántúli sugárzással törölhető.

File

Lásd „adatállomány" címszót.

Gépi kódú program

Az a programnyelv, amelyet a számítógép mikroprocesszora közvetlenül megért, és végrehajt.

Hardcopy

Program, ill. művelet, amely a (grafikus vagy karakteres) képernyő tartalmát a nyomtatókészüléken kinyomtatja.

Hardver

A számítógéprendszert alkotó alkatrészek összessége.

Hexadecimális

A 16-os alapú számrendszer neve, a hexadecimális számrendszert a számítástechnikában általában gépi kódú utasítások ábrázolására használjuk. Egy hexadecimális szám 4 bitet jelöl.

Integer változó

Egész számú változó, amelyet a más típusú változóktól a változó neve után írt % (százalékjel) különböztet meg, pl. A%. Ezek a változók a −32768 és a +32767 közötti értékeket vehetik fel.

Interpreter

Olyan program, amely egy magas szintű programozási nyelven megírt programot értelmez úgy, hogy az gépi kódú utasítások sorozatára alakíthassa át. A magas szintű nyelven írt programot utasításról utasításra értelmezi, lefordítja, és azonnal végrehajtja.

Interrupt

Megszakítás. A számítógép operációs rendszere a gépen futó normál (például az általunk írt) programot megszakítja azzal a céllal, hogy a saját, úgynevezett megszakító programját végrehajtsa. Ezt követően folytatódik a megszakított program végrehajtása. A Plus/4-es gépnél erre a megszakításra másodpercenként 60-100-szor kerül sor.

Karakter generátor

A számítógépben levő karaktergenerátor. Azt határozza meg, hogy milyen karakterek jelennek meg a képernyőn. A Plus/4-es gépnél a karaktergenerátor a $D000–$d7FF közötti tárcímeken helyezkedik el.

Karakterlánc

Karakterláncnak (vagy az elterjedt, eredeti néven stringnek) az alfanumerikus jelek sorozatát nevezik. A karakterlánc típusú változók azonosítására a karakterlánc neve után álló $-jel (dollárjel) szolgál.

Modem

Távadatátvitelre szolgáló készülék, amely az adatok átvitelét a telefonhálózaton keresztül, egy akusztikus csatlakozó segítségével teszi lehetővé.

Munkatár

A számítógép tárának az a része, amelybe írhatunk, és amelyből olvashatunk is. A Plus/4-es munkatára 64 kbyte. Lásd RAM is.

Műveleti kód

Műveleti kód (ill. eredeti néven operációs kód) alatt a régi kódú utasításokat értjük.

Operációs rendszer

Azoknak a rendszerprogramoknak a gyűjteménye, amelyek lehetővé teszik, hogy a számítógépen dolgozhassunk. Például: karakterek beírása, megjelenítése, külső egységek működtetése stb.

Paraméter

Egy utasításhoz tartozó kiegészítő információk, amelyek az utasítást részletesen meghatározzák.

Párhuzamos interface

A rendszer a karakterek valamennyi bitjét (a mikroszámítógépeknél általában 8-at) egyidejűleg továbbítja. Ennek előfeltétele, hogy mindegyik bit részére külön adatvezeték álljon rendelkezésre.

Periféria

Azoknak a készülékeknek a gyűjtőneve, amelyek a számítógéphez csatlakoztathatók, de nem a számítógép közvetlen alkatrészei. (Ebbe a gyűjtőnévbe tartozik pl. a képernyő, a nyomtató, lemezmeghajtó, MODEM stb.)

Puffertár

Adatok ideiglenes tárolására alkalmas tárterület. A Plus/4-esben van például egy billentyűzetpuffer és egy kazettapuffer.

RAM

Random Access Memory rövidítése. Magyar jelentése: írható és olvasható tár. Ez lényegében a számítógép munkatára, amelybe programot és adatokat írhatunk, és azokat onnan elolvashatjuk. A számítógép kikapcsolásakor ennek a tárnak a tartalma törlődik.

RESET gomb

Ennek a gombnak a megnyomására a számítógép ismét a bekapcsolási állapotba kerül.

ROM

A Read Only Memory kifejezés rövidítése. Magyar jelentése: csak olvasható tár. Ilyen tár tartalmazza a számítógép operációs rendszerét, a BASIC interpretert, a karaktergenerátort, valamint a Plus/4-esnél a beépített szoftvereket.

RS-232

Az RS-232 szabvány szerinti, soros interface megnevezése. A Plus/4-esnél ez az interface áll a felhasználói csatlakozón (a User Port-on) rendelkezésre. Azt figyelembe kell venni, hogy ennél a gépnél a csatlakozón TTL-szintű, azaz 5 V és 0 V feszültség van.

Soros adatátvitel

Az adatfeldolgozásnak, ill. az adatátvitelnek az a módja, amelynél az adatok időben egymás után, tehát egyik bit a másik után kerül átvitelre.

Szoftver

A szoftver a számítógépre írt valamennyi program gyűjtőneve. A szoftver és a hardver együttesen alkotnak egy működőképes számítógépes rendszert.

Tárcím

A számítógépben a tárcímek (rekeszek) folyamatosan sorszámozva vannak. Mindegyik tárcímnek van tehát egy száma. A tárcímek legnagyobb számát a címbuszban levő vezetékek száma határozza meg.

249,—