Lothar Englisch

Gépi kódú programozás haladóknak

C 64 & PC 128

DATA BECKER – NOVOTRADE

Lothar Englisch

Gépi kódú programozás haladóknak

C 64 & PC 128

DATA BECKER – NOVOTRADE

A mű eredeti címe: Das Machinensprache Buch für Fortgeschrittene zum C 64 & PC 128 (2., javított kiadás), 1985

Fordította: DOBOSNÉ HARTYÁNYI MÁRIA
Lektorálta: DR. OBÁDOVICS J. GYULA és DR. LENGYEL JÓZSEF
A programokat ellenőrizte: DR. LENGYEL JÓZSEF

A kiadásért felel: RÉNYI GÁBOR, a NOVOTRADE RT. igazgatója
Kiadványmenedzser: BÉKÉS TAMÁS
Felelős szerkesztő: TARR KÁLMÁNNÉ
Szerkesztő: ROCHLITZ ANDRÁS
Műszaki szerkesztő: DÉVÉNYI ERIKA

Szedte: a Nyomdaipari Fényszedő Üzem (867316/08)
Készült a Nyírségi Nyomdában, Nyíregyházán (7,6 A/5 ív)
Budapest, 1986
ISBN 963 02 4229 X

© Hungarian translation Dobosné Hartyányi Mária

Copyright © 1985 DATA BECKER GmbH – Merowingerstr. 30. 4000 Düsseldorf

Minden jog fenntartva. A DATA BECKER cég írásbeli hozzájárulása nélkül tilos a könyvet vagy annak részeit bármilyen eljárással (nyomtatás, fotókópia vagy egyéb technika), elektronikus rendszerek felhasználásával másolni, sokszorosítani, terjeszteni.

FONTOS MEGJEGYZÉS

A könyvben szereplő kapcsolások, eljárások és programok közlése a bejelentett szabadalmak figyelembevétele nélkül kerülnek ismertetésre. Kizárólag amatőr és tanulási célokat szolgálnak, egyéb célokra történő felhasználásuk tilos!
A szerzők a közölt műszaki adatokat, kapcsolásokat és programokat a legnagyobb gondossággal állították össze. Ennek ellenére előfordulhatnak hibák, amelyekért a DATA BECKER GmbH sem garanciát, sem jogi felelősséget nem vállal.
A könyvvel kapcsolatos bármilyen észrevételt a szerzők köszönettel vesznek.

TARTALOMJEGYZÉK

BEVEZETÉS 7

1.1 Számábrázolás a COMMODORE 64-es és 128-as gépeken 8
1.2 Átalakítás lebegőpontos formátumra 19
1.3 Átalakítás egész típusú (INTEGER) formátumra 22
1.4 A BASIC értelmező (interpreter) aritmetikai rutinjai 24
1.5 A BASIC értelmező lebegőpontos függvényei 33

2.1 A megszakítások programozása 47
2.2 A CIA 6526-os processzor 50
2.3 A rendszermegszakítások felhasználása 54
2.4 A videovezérlő megszakításai 65
2.5 A CIA 6526-os megszakításai 69
2.6 A számlálók (timerek) felhasználása 72

3.1 Az operációs rendszer és a BASIC bővítései 79
3.2 A BASIC vektorok 81
3.3 A strukturált programozás 88
3.4 Az új utasításszavak beépítése 95
3.5 Az operációs rendszer vektorai 105
3.6 Rendszerfüggetlen nyomtatás (spooling) 115

4.1 Kulcsszavak és tokenjeik táblázata 121
4.2 Nullásiap összehasonlító táblázat 122

BEVEZETÉS

Ez a könyv a „Gépi kódú programozás a Commodore 64-esen" c. könyv folytatásának tekinthető.
Feltételezve, hogy az Olvasó elsajátította az ott közölt alapismereteket, ebben a kötetben a gépi kódú programok magasabb szintű alkalmazási lehetőségeit szeretnénk bemutatni.
Annak ellenére, hogy a programokat a C–64-es gépre irtuk, a C–128-as gép tulajdonosait sem zárjuk ki az Olvasók köréből. A C–128-as gép átkapcsolható C–64-es üzemmódra, a programok tehát változtatás nélkül futtathatóak a C–128-as gépen.
Ha valaki mégis szeretné átírni a programokat C–128-as üzemmódra, hasznos segédeszközökre talál a „128 Intern" c. könyvben, ill. a jelen könyv függelékében található táblázatban. A programok átírása nem jelent különösebben nehéz feladatot, hiszen a 128-as operációs rendszer a 64-es rendszer bővítése. Ahol szükséges, mindig utalunk majd a két gép közötti különbözőségre.

A könyv három fejezetből áll:

Az első fejezetben részletesen bemutatjuk, hogyan tárolja és ábrázolja a Commodore 64-es a számokat, ill. milyen beépített gépi kódú rutinok teremtik meg azt a lehetőséget, hogy az egyik ábrázolási módról áttérjünk a másikra. Ismertetjük a beépített aritmetikai rutinokat, és feltárjuk ezek alkalmazási lehetőségeit. A fejezet egyik legérdekesebb része az, amelyben saját USR függvénnyel hívható aritmetikai rutinokat készítünk.

A második fejezet tartalma biztosan nagyon sok érdekességet nyújt azoknak az Olvasóinknak, akik elsősorban gépi kódban programoznak. A fejezet témája a Commodore 64-es megszakítási technikája. A rendszermegszakítás fogalmának tisztázása után bemutatjuk a megszakítások kiváltásának lehetőségeit.
Az elméleti megfontolásokat mindenütt konkrét feladatok megoldására szolgáló mintaprogramokkal szemléltetjük. A fejezet utolsó példája egy olyan gépi kódú rutin, amely egy BASIC szubrutin megszakításokkal vezérelt hívását valósítja meg.

A harmadik fejezetben a BASIC értelmező (interpreter) és az operációs rendszer vektorait ismertetjük.
Az egyes vektorok működését itt is programok szemléltetik. Példát mutatunk arra is, hogy miként lehet a beépített vektorokat saját utasítások (pl. a REPEAT-UNTIL utasításpár) meghatározására felhasználni.

1.1 SZÁMÁBRÁZOLÁS A COMMODORE 64-ES ÉS 128-AS GÉPEKEN

A Commodore 64-es és 128-as kétféle belső számábrázolási móddal rendelkezik:
Az ábrázolási módok egyike, az olvasó által biztosan jól ismert *egész típusú* vagy INTEGER ábrázolás. Az egész típusú változó értéke −32 768-tól +32 767-ig terjedhet és tárolása két byte-ot vesz igénybe. A két byte 16 bitje közül a legfelső bit az előjel meghatározására szolgál.

```
***      T1.      ***

Decimalis        Binaris              Hexadecimalis
-----------------------------------------------------
-32768       1 000 0000 0000 0000        80 00
-32767       1 000 0000 0000 0001        80 01
-32766       1 000 0000 0000 0010        80 02
-32765       1 000 0000 0000 0011        80 03
                    ...
-2           1 111 1111 1111 1110        FF FE
-1           1 111 1111 1111 1111        FF FF
 0           0 000 0000 0000 0000        00 00
 1           0 000 0000 0000 0001        00 01
 2           0 000 0000 0000 0010        00 02
                    ...
 32766       0 111 1111 1111 1110        7F FE
 32767       0 111 1111 1111 1111        7F FF
```

Az előjeles 16 bites számábrázoláshoz hasonló a 8 bites előjeles számábrázolás, ahol a számok értéke −128 és +127 közé eshet és amellyel már találkozhattunk a relatív címzés kapcsán.
Vannak olyan számítógépes feladatok, amelyek megoldásához elegendőek az adott tartományba eső egész számok, ez azonban nem általános. A legtöbb feladatot nem lehet megoldani az egész számok körében és nem sokra mennénk a számítógéppel, ha az nem tudna műveleteket végezni tizedesjegyeket tartalmazó, bővebb értéktartományú számokkal. Az ilyen típusú számokat a gép *lebegőpontos* formában ábrázolja. A lebegőpontos elnevezést a *fixpontos* ellentéteként értelmezhetjük. A fixpontos számábrázolásban a tizedespont helye rögzített, a rögzített számú tárolóhelyek valamelyikén. Az egész típusú számok is fixpontosnak tekinthetőek, hiszen a tizedespont azonos helyen, az utolsó értékes jegy mögött képzelhető el. A lebegőpontos tárolás alapelve egészen más.
Tegyük fel, hogy 10-es számrendszerben dolgozunk. A lebegőpontos ábrázoláshoz a számot először felbontjuk egy 1 és 10 közé eső szám és 10 (a számrendszer alapja) valamely hatványának szorzatára.
Pl.:

$15 = 1.5 * 10^1$
$230 = 2.3 * 10^2$

A felbontásban 10 hatványkitevője természetesen negatív, illetve nulla is lehet:

$5 = 5*10^{0}$
$0.7 = 7*10^{-1}$

Ha a számrendszer alapja rögzített, a fenti felbontás bármely számra egyértelműen elvégezhető, azaz minden számhoz egyértelműen meghatározhatjuk a *mantisszát* (utolsó példánkban 7) és a kitevőt (az utolsó példában -1). A szám így kapott alakját *normál alak*nak nevezzük. A normál alakban szereplő szorzótényező (mantissza) értéke mindig 1 és a számrendszer alapja közé esik. A normál alakban felírt számokkal rögzített matematikai szabályok szerint végezhetünk műveleteket: pl. két ilyen alakú szám szorzatát megkapjuk, ha mantisszáikat összeszorozzuk, kitevőiket pedig összeadjuk. Ha a mantisszák szorzata nagyobb mint 10, akkor ezt 10-zel osztjuk, és a kitevőt módosítjuk. Nézzünk erre példát:

$5*10^{0}$ szorozva $7*10^{-1}$

A mantisszák szorzata 35, a kitevők összege -1, a szorzat tehát $35*10^{-1}$. Az eredményt normalizálni kell. A mantisszában a tizedespontot egy hellyel balra toljuk és a kitevőt ezzel egyidejűleg eggyel megnöveljük. Ha a tizedespontot jobbra kellett volna eltolnunk, a kitevőt természetesen csökkentenünk kellett volna.
Két normál alakban megadott számot a matematika szabályai szerint csak akkor tudunk összeadni, ha a kitevőjük egyenlő. A fenti példában azt a számot módosítjuk, amelyiknek a kitevője kisebb.

$7*10^{-1} = 0.7*10^{0}$

Most már összeadhatjuk a két mantisszát:

$5*10^{0} + 0.7*10^{0} = 5.7*10^{0}$

Az eredmény most olyan, hogy módosítás nélkül megfelel a normál alak követelményeinek.
Hogyan lehet a fenti elgondolásokat a mikroprocesszor belsejében megvalósítani?
Minthogy a processzor csak bináris számokat tud kezelni, először azt gondoljuk végig, hogy a fenti műveleti szabályok érvényben maradnak-e bináris számok esetében is?
A számrendszer alapja tehát 2. Mielőtt a lebegőpontos számábrázolásba belemélyednénk, el kell döntenünk, hogy mekkora számtartományt akarunk átfogni és milyen pontossággal akarjuk az egyes számokat ábrázolni. A fentiek alapján világos, hogy az ábrázolható számtartomány nagyságát a normál alakban szereplő kitevő, a maximálisan tárolható számjegyek számát, azaz a pontosságot pedig a mantissza határozza meg. (A decimális számok lebegőpontos ábrázolásából adódó pontossági kérdésekre még vissza fogunk térni.)
Egy bináris szám normál alakja tehát a következő lehet:

$1.011101*2^{10010}$ vagy $1.011101*2^{18}$

decimálisan:

$$
\begin{aligned}
1*2^{18} &= 262\,144\\
+0*2^{17} &= 0\\
+1*2^{16} &= 65\,536\\
+1*2^{15} &= 32\,768\\
+1*2^{14} &= 16\,384\\
+0*2^{13} &= 0\\
+1*2^{12} &= 4\,096\\
&\overline{\quad 380\,928}
\end{aligned}
$$

Nézzünk egy másik, valódi tizedespontot tartalmazó számot is:

$1.011*2^0$

$$
\begin{aligned}
1*2^{0} &= 1\\
+0*2^{-1} &= 0\\
+1*2^{-2} &= 0.25\\
+1*2^{-3} &= 0.125\\
&\overline{\quad = 1.375}
\end{aligned}
$$

Ha az ábrázolandó szám kisebb mint 1, akkor a normál alakban szereplő kitevő kisebb mint 0. A belső lebegőpontos ábrázolásban tehát a gépnek negatív kitevőt is kell tudnia értelmezni. A negatív számok tárolásának egyik módja a *kettes komplemens*sel történő tárolás. Ha a kitevő tárolására 1 byte-ot szánunk, akkor az ábrázolható kitevők értéke −128 és 127 közé eshet. Mekkora számtartományt jelent ez decimálisan? Az átszámítás eredménye:

$2^{127} = 1.7*10^{38}$
$2^{-128} = 3.9*10^{-39}$

Ebből az következik, hogy 1 byte-os kitevő esetén olyan decimális számok ábrázolhatók, amelyekben a tizedespont előtt 38, a tizedespont után pedig 39 számjegy áll. A kapott számtartományba minden, a gyakorlatban előforduló szám belefér.
A Commodore 64-es a belső lebegőpontos számok kitevőit nem kettes komplemenssel ábrázolja, hanem egy ún. *offset*tel. Ha minden előforduló kitevőhöz hozzáadunk 129-et, vagy hexadecimális $81-et, az eredmény mindig pozitív szám lesz.
A következő táblázat tartalmazza az összefüggést a tárolt és a valódi kitevő között:

*** T2. ***

Hexa ertek	Kitevo	Decimalis ertek
$00	Lasd a szoveget	0
$01	-128	3.9*10E-40
$02	-127	5.9*10E-39
$03	-126	1.2*10E-38
$7F	-2	0.25
$80	-1	0.5
$81	0	1

$82	1	2
$83	2	4
$FE	125	4.3*10E+37
$FF	126	8.5*10E+37

Ha az ábrázolt számban a kitevő értéke nulla, akkor maga a szám is nulla.
A kitevő ábrázolási módjának kiválasztása után nem kevésbé fontos a mantissza ábrázolási módja, hiszen ezen múlik az ábrázolható számok pontossága.
A Commodore 64-es a mantisszát 4 byte-on tárolja, ami azt jelenti, hogy egy szám mantisszája maximálisan 32 bináris jegyet tartalmazhat. Hány decimális jegynek felel ez meg?
Ennek eldöntéséhez hasonlítsuk össze két olyan bináris szám decimális értékét, amelyek binárisan csak az utolsó helyiértékeiken különböznek:

1.111 1111 1111 1111 1111 1111 1111 1111 és
1.111 1111 1111 1111 1111 1111 1111 1110

A fenti két szám tehát 2^{-31} értékkel tér el egymástól. Ez decimálisan kb. a

$4.6566129*10^{-10}$ vagy
$0.46566129*10^{-9}$

értéknek felel meg.
Az eredeti két szám értéke közel 2, és egymástól a tizedik decimális helyiértékben kb. 5-tel különböznek. Ebből arra következtethetünk, hogy a 4 byte-on tárolt mantisszával kb. 9 jegy pontosságot érünk el. A 9 jegy pontosság relatív pontosság, amely független a kitevőtől. Ha a decimális számot normál alakra hozzuk, azaz a tizedespont előtt egy 1 és 9 közé eső szám áll, akkor a lebegőpontos bináris számábrázolással meg tudjuk különböztetni egymástól azokat a számokat is, amelyek decimálisan csak a kilencedik helyen különböznek.
Most már rendelkezünk azzal a lehetőséggel, hogy egy −128 és 126 közé eső kitevő és egy 4 byte-on ábrázolt mantissza segítségével a decimális számokat 9 jegy pontossággal ábrázoljuk. Hiányzik azonban még a számok előjele. Egy kis ötlettel megoldjuk az előjel ábrázolását anélkül, hogy közben a pontosságból vesztenénk.
A számok normál alakjában a mantissza mindig 1 és a számrendszer alapjának értéke közé esik. Kettes számrendszerben 1 és 2 közé, azaz minden szám belső ábrázolásában a mantissza első jegye 1. Minthogy ezt az 1-est teljesen fölösleges tárolni, megállapodás szerint az első bitet az előjel tárolására fogjuk használni.
Ha az első bit értéke 0, akkor a szám pozitívnak, ha 1, akkor a szám negatívnak tekinthető.
Most már minden információ rendelkezésünkre áll ahhoz, hogy egy tetszőleges számot átírjunk lebegőpontos alakra, azaz pontosan úgy, ahogyan azt a gép tárolja. Pl.:

$1 = 1*2^0$
$= 1.000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000*2^0$

Ha figyelembe vesszük az előjel bit jelentését, és a táblázat alapján meghatározzuk a kitevőt, a következő eredményre jutunk:

0000 0000 0000 0000 0000 0000 0000 0000 1000 0001

A gépben a tárolási sorrend fordított: először a kitevő byte-ja, majd a mantissza 4 byte-ja következik:

1000 0001 0000 0000 0000 0000 0000 0000 0000 0000

Ugyanezt a számot hexadecimálisan felírva sokkal áttekinthetőbb képet kapunk:

81 00 00 00 00

Az ábrázolt szám tehát a lebegőpontos 1-es. Nézzük meg, hogyan tárolódik a gépben a 10? Írjuk fel a 10 kettes számrendszerbeli normál alakját:

$$\begin{aligned} 10 &= 8+2 \\ &= 2^1+2^3 \\ &= 1*2^3+0*2^2+1*2^1 \\ &= 1.01*2^3 \end{aligned}$$

A belső ábrázolás:

1000 0100 0010 0000 0000 0000 0000 0000 0000 0000 azaz
84 20 00 00 00

Nézzünk példát egy negatív szám belső ábrázolására:

$$\begin{aligned} -5.5 &= -(4+1+0.5) \\ &= -(2^2+2^0+2^{-1}) \\ &= -(1*2^2+0*2^1+1*2^0+1*2^{-1}) \\ &= -1.011*2^2 \end{aligned}$$

azaz lebegőpontosan:

1000 0011 1011 0000 0000 0000 0000 0000 0000 0000 azaz
83 B0 00 00 00

A negatív számokat arról lehet könnyen felismerni, hogy a mantissza első byte-jának értéke nagyobb vagy egyenlő mint $80.
A fentiek alapján bármely lebegőpontosan tárolt számot átírhatunk decimális alakra. Jelöljük az egymást követő byte-okat az alábbiak szerint (EX jelöli a kitevő byte-ját az „exponens" szó rövidítéseként):

EX M1 M2 M3 M4
83 B0 00 00 00

A tárolt szám decimális értékét az alábbi képlettel határozhatjuk meg:

X = (−SGN(M1 AND 128)*2+1)*2 ↑ (EX−129)*(1+((M1 AND 127)+
+(M2+(M3+M4/256)/256)/256)/128)

A képlet alapján világos, hogy az előjelet a mantissza legfelső byte-jából (M1) az első bit értéke alapján határoztuk meg, a kitevő pontos értékének kiszámításánál pedig figyelembe vettük a 129-es értékkel való eltolást. A mantissza kiszámításánál minden byte értéke súlyozottan szerepel, sorrendjének megfelelően. Az egymást követő byte-ok közül a felső byte tartalma mindig 256-szorosa az őt közvetlenül követő byte tartalmának.
A képlet helyességét érdemes egyszer lépésenként ellenőrizni:

X = (−SGN(176 AND 128)*2+1)*2 ↑ (131−129)*
(1+((176 AND 127)+(0+(0+0/256)/256)/256)/128)

A számítás eredménye: −5.5. A korábbi példa alapján megadott eredeti decimális értéket tehát visszakaptuk.
A számítások, illetve átalakítások közben mind ez ideig semmilyen nehézségbe nem ütköztünk. Próbáljuk azonban a 0.4 decimális számot a megfelelő lebegőpontos alakra átírni:

```
***      T3.      ***

 0.4                          2 kitevoje
 0.25                            -2
=====
 0.15
 0.125                           -3
======
 0.025
 0.015625                        -6
========
 0.009375
 0.0078125                       -7
=========
 0.0015625
 0.0009765625                    -10
==========
 0.0005859375
 0.00048828125                   -11
===========
 0.00009765625
 0.0000610351562S                -14
==============
 0.00003662109275                 stb.
```

A számolást tetszőlegesen sokáig folytathatjuk, az átalakítás sosem fog véget érni. A kapott bináris számban a számjegyek periodikusan ismétlődnek:

1.1001 1001 1001 1001 1001 1001 1001 1001 1001 ...*2^-2

A decimális 0.4 számérték nem fejezhető ki véges bináris tört alakjában.
Az eredmény véges sok (31) bináris jeggyel, korlátozott pontossággal a következőképpen ábrázolható:

1.1001 1001 1001 1001 1001 1001 1001 100*2^-2

A nagyobb pontosság érdekében annyit tehetünk, hogy a bináris számjegyek egyszerű elhagyása helyett felfelé kerekítünk.
Bináris számoknál, ha az elhagyott számjegy 1-es, akkor felfelé, egyébként lefelé kerekítünk. A fenti példában felfelé kell kerekítenünk, a végleges eredmény tehát:

1.001 1001 1001 1001 1001 1001 1001 101*2 ^{2}

A kitevő és az előjel figyelembevételével a 0.4 belső számábrázolás szerinti alakja:

0111 1111 0100 1100 1100 1100 1100 1100 1100 1101

azaz hexadecimális alakban:

7F 4C CC CC CD

Annak, hogy bizonyos véges sok tizedesjegyet tartalmazó számokat nem tudunk átváltani véges sok számjegyet tartalmazó bináris alakra, nem az az oka, hogy az új számrendszer alapja 2. Ez a probléma mindig felmerül, valahányszor az egyik számrendszerről át akarunk térni egy másik számrendszerre. Az 1/3 valódi tört alakja tizes számrendszerben ugyancsak végtelen tizedes tört:

0.33333 33333 33333.....

holott ugyanennek a számnak pl. hármas alapú számrendszerbeli alakja:

0.1

az 1*3 $^{-1}$ értéknek megfelelően.

Miután némileg megismerkedtünk a lebegőpontos számábrázolás elméleti alapjaival, gondoljuk végig, mire lehetne eddigi ismereteinket felhasználni.
A BASIC értelmező (interpreter) jelentős részét foglalják el azok a rutinok, amelyek a számokat az egyik formáról a másikra alakítják át, illetve amelyek a lebegőpontos számokkal aritmetikai műveleteket végeznek. Hogyan tudnánk hasznosítani ezeket a rutinokat saját készítésű programjainkban?
A BASIC értelmező két ún. *lebegőpontos akkumulátor*ban (floating point accu) tárolja azokat a számokat, amelyeket a későbbiek során feldolgoz. Az akkumulátorok megnevezésére az FAC rövidítést használjuk.
Az értelmező az FAC 1-et minden műveletnél, az FAC 2-t pedig csak a két operandussal rendelkező (pl. az összeadás) műveleteknél veszi igénybe. A művelet eredménye mindig az FAC 1-be kerül. Az 1-es akkumulátort gyakran röviden FAC-vel, a 2-es akkumulátort pedig ARG (argumentum)-vel jelöljük. Az akkumulátorokban a számokat az értelmező nem 5 byte-on tárolja, ugyanis az előjel tárolására külön lefoglal egy byte-ot.
Sőt a műveletek elvégzése közben szükséges kerekítésekre még egy további byte is rendelkezésre áll. A lebegőpontos akkumulátorok tárcímei a nullás lapon:

```
***      T4.      ***
```

	FAC	ARG
Exponens	$61	$69
Mantissza 1	$62	$6A
Mantissza 2	$63	$6B
Mantissza 3	$64	$6C
Mantissza 4	$65	$6D
Elojel	$66	$6E
Kerekitesi byte	$70	
Elojel-osszehasonlito byte	$6F	

Az előjel összehasonlító byte-ot a két operandusú műveletek használják. Értéke egyező előjeleknél $00, ellentétes előjeleknél $FF.
A BASIC értelmező egy sor rutinja szolgál a lebegőpontos számok kezelésére. Kezdjük az elemzést azzal a rutinnal, amely beolvas egy decimális számot és átalakítja lebegőpontossá. Ezt a rutint az értelmező minden numerikus adatbevitelnél meghívja.
Mielőtt belemélyednénk a rutin működésének elemzésébe, tegyünk egy kis kitérőt és vizsgáljuk meg az ún. CHRGET rutint, amely beolvas egy karaktert a billentyűzetről, vagy a BASIC szövegből. Ez a rutin is a nulláslapon található és feladata a karakter beolvasásán túl annak ellenőrzése, vizsgálata.
A CHRGOT rutin az előző egy beugrási pontja, és arra szolgál, hogy az utoljára olvasott karaktert még egyszer betöltsük.

```
***      P1.      ***
CHRGET  INC TXTPTR
        BNE CHRGOT
        INC TXTPTR+1
CHRGOT  LDA TEXT
        CMP #":"
        BCS EXIT
        CMP #" "
        BEQ CHRGET
        SEC
        SBC ##30
        SEC
        SBC ##D0
EXIT    RTS
```

```
***      P1.      ***
CHRGET  INC TXTPTR
        BNE CHRGOT
        INC TXTPTR+1
CHRGOT  LDA TEXT
        CMP #":"
        BCS EXIT
        CMP #" "
        BEQ CHRGET
        SEC
        SBC ##30
        SEC
        SBC ##D0
EXIT    RTS
```

A rutin leglényegesebb tulajdonsága, hogy a RAM területén lévén, képes önmagát módosítani.
A TXTPTR mutató, amely a betöltendő karakter helyét adja meg, a programban van elhelyezve. Ez azonnal világossá válik, ha az alábbi assembler listát szemügyre vesszük:

```
***      P2.      ***
0073 E6 7A      INC $7A
0075 D0 02      BNE $0079
0077 E6 7B      INC $7B
0079 AD 02 02   LDA $0202
007C C9 3A      CMP ##3A
007E B0 10      BCS $008A
0080 C9 20      CMP ##20
0082 F0 EF      BEQ $0073
0084 38         SEC
0085 E9 30      SBC ##30
```

```
0087 38        SEC
0088 E9 D0     SBC #$D0
008A 60        RTS
```

Amikor meghívjuk a CHRGET rutint, először a betöltő utasítás címmezője (a CHRGOT címkével jelzett utasítás) megnő eggyel, majd a megnövelt cím tartalmát beolvassuk. A beolvasást különböző vizsgálatok követik. A beolvasott karaktert először összehasonlítjuk a kettősponttal. Ha ASCII kódja nagyobb vagy egyenlő, mint a kettőspont ASCII kódja, akkor közvetlenül az RTS utasításra ugrunk. Ekkor az átviteli kapcsoló (carry flag) értéke 1. Ha a beolvasott karakter kettőspont, akkor a zérókapcsoló értéke is 1 lesz.
Az utasítás végét a fentiek szerint a zéró kapcsoló értéke is mutatja. Ha a beolvasott karakter ASCII kódja kisebb, mint a kettőspont kódja, összehasonlítjuk a szóköz karakterrel (kódja a decimális 32, illetve hexadecimális 20).
Ha összehasonlítás után a Z kapcsoló értéke 0, azaz a két kód egyenlő, akkor a rutin elejére ugrunk, azaz olvassuk a következő karaktert. Ez azt jelenti, hogy az értelmező olvasás közben a szóköz karaktereket átlépi. A következő két kivonás a beolvasott karakter kódját nem változtatja meg, feladatuk mindössze annyi, hogy a C kapcsoló értékét módosítsák. A C kapcsoló értéke törlődik (0 lesz), valahányszor az olvasott karakter ASCII számjegy, azaz hexadecimális kódja $30 és $39 közé esik.

Foglaljuk össze röviden még egyszer az elmondottakat: a CHRGET rutin megnöveli a szövegmutatót (TXTPTR-t) és a kapott címen talált értéket betölti az akkumulátorba.
Ha az olvasott érték megfelel a kettőspontnak vagy nullabyte, akkor vége a sornak vagy az utasításnak. Ezt a rutin azzal jelzi, hogy a Z kapcsoló értékét 1-re állítja. Ha a beolvasott karakter számjegy, akkor a C kapcsoló értékét törli.
Térjünk most vissza arra a rutinra, amelyet eredetileg elemezni készültünk. Ez a rutin a decimális számot lebegőpontos alakra hozza. Hívásakor a decimális szám első karakterének az akkumulátorban kell lennie és a kapcsolók értékét a CHRGET rutinnak megfelelően be kell állítani. A szövegmutató (TXTPTR) természetesen most az átalakítandó számra mutat. A következő kis program beolvas egy számot és átalakítja lebegőpontossá:

```
***      P3.      ***

100: 033C                        .OPT P,00
105: 033C                        **   828
110: 007A            TXTPTR      =    $7A
120: 0079            CHRGOT      =    $79
130: BCF3            ASCFLOAT    =    $BCF3
140: 033C A9 4B                  LDA  #<ZAHL
150: 033E A0 03                  LDY  #>ZAHL
160: 0340 85 7A                  STA  TXTPTR
170: 0342 84 7B                  STY  TXTPTR+1
180: 0344 20 79 00               JSR  CHRGOT
190: 0347 20 F3 BC               JSR  ASCFLOAT
200: 034A 00                     BRK
210: 034B 31 2E 32 ZAHL          .ASC "1.2345"
220: 0351 00                     .BYT 0
```

Ha a programot assembláljuk és a Monitor programból

G 033C

utasítással elindítjuk és lefuttatjuk, az 1.2345 szám lebegőpontos alakban a rendelkezésünkre áll. Ezt az

```
M 0061 0066
```

utasítással ellenőrizhetjük is, melynek hatására a következő sor jelenik meg a képernyőn:

```
***       T5.       ***
>: 0061 81 9E 04 18 93 00
```

Próbáljuk meg a programmal a 0.4 decimális számot konvertálni. Ehhez a $034B címtől kezdve el kell helyeznünk a számjegyeket és egy nullabyte-tal lezárni:

```
***       T6.       ***
>M 034B 034B
>: 034B 30 2E 34 00
```

A futtatás eredménye:

```
***       T7.       ***
>: 0061 7F CC CC CC CC 00
```

Az előjelet a hatodik byte tartalma határozza meg, értéke pozitív szám esetén nulla.
A fenti programmal a féllogaritmikus alakban megadott decimális számokat is fel tudjuk dolgozni, mint pl. −1.4E−7 vagy 1E12. Vegyünk példaként most egy negatív, féllogaritmikus alakban megadott számot, pl. a −1E8-at. A program futtatásának eredménye:

```
***       T8.       ***
>: 0061 9B BE BC 20 00 FF
```

Az utolsó byte tartalma ($FF) a negatív előjelre utal.
Térjünk vissza egy pillanatra a 0.4 átalakításának eredményéhez. Ha az eredményt alaposan megnézzük, azt tapasztaljuk, hogy az nem egyezik meg a kézi számítás eredményével, annál az utolsó helyiértéken eggyel kisebb. Az eltérés oka, hogy az átalakító rutin nem veszi figyelembe a kerekítésből adódó jegyet. Futtassuk le még egyszer a programot a 0.4 értékkel és ellenőrizzük a $70-es címen a kerekítő byte tartalmát. Az ott található $80 érték arra utal, hogy a kerekítésnél átvitel keletkezett, tehát a lebegőpontos számot az utolsó helyiértéken meg kell növelni eggyel. Egészítsük ki az előző programot az alábbiak szerint, így az a továbbiakban automatikusan elvégzi a kerekítést.

```
***       P4.       ***
100:  033C                        .OPT P,00
105: *033C                        *=   828
110:  007A             TXTPTR     =    $7A
120:  0079             CHRGOT     =    $79
130:  BCF3             ABCFLOAT   =    $BCF3
140:  BC1B             ROUND      =    $BC1B
150:  033C A9 4E                  LDA  #<ZAHL
160:  033E A0 03                  LDY  #>ZAHL
```

```
170:  0340 85 7A          STA  TXTPTR
180:  0342 84 7B          STY  TXTPTR+1
190:  0344 20 79 00       JSR  CHRGOT
200:  0347 20 F3 BC       JSR  ASCFLOAT
210:  034A 20 1B BC       JSR  ROUND
220:  034D 00             BRK
230:  034E 30 2E 34 ZAHL  .ASC "0.4"
240:  0351 00             .BYT 0
```

Az FAC tartalma az új program futtatása után már megfelel a várakozásnak:

```
***       T9.       ***

>: 0061 7F CC CC CC CD 00
```

A kerekítés után a kerekítő byte tartalma törlődik, erről az Olvasó könnyen meggyőződhet.
Miután megértettük a számjegyfüzérek lebegőpontos számmá alakításának módját, nézzük meg, hogyan lehet az eljárást megfordítani, azaz lebegőpontos számot decimális számjegyekké alakítani. A megoldás nagyon egyszerű, minthogy a feladatot elvégző rutin készen van. A rutin neve FLOATASC és kezdőcíme $BDDD. A rutin az FAC-ben található lebegőpontos számnak megfelelő számjegyfüzért a $0100-as címtől adja vissza. Írjuk be az FAC-be a következőt:

```
***       T10.      ***

>: 0061 90 8F 00 00 00 00
```

A rutin hívása után írassuk ki a képernyőre a $0100-as cím tartalmát:

```
***       T11.      ***

>M 0100 0107
>: 0100 2D 33 36 36 30 38 00     -36608
```

Az FAC tartalma tehát a decimális −36608.
A rutin hívása után az akkumulátor és az Y regiszter mindig a füzér címét tartalmazza. Példánkban az A = 0 és az Y = 1 (alsó/felső byte). A kapott füzért kiírathatjuk a cím alapján a képernyőre. A $AB1E címen kezdődő STROUT rutin pontosan ezt a célt szolgálja.
Mielőtt rátérnénk a lebegőpontos műveleteket végző rutinok ismertetésére, ismerkedjünk meg azokkal a rutinokkal, amelyek az egész típusú számok lebegőpontossá alakítását végzik el. Ezekre a rutinokra igen nagy szükség van, hiszen nagyon gyakran előfordul, hogy a lebegőpontos művelet bemenő operandusa egész típusú, illetve az, hogy a művelet eredményét ismét egész típusú számként kell visszaadni.

1.2 Átalakítás lebegőpontos formátumra

Egy byte-os előjeles egész szám átalakítása
Az alábbi rutin egy egybyte-os, előjeles, −128 és 127 közé eső egész számot alakít lebegőpontos formátumra.
Az egész számot az akkumulátorban kell a rutinnak átadni:

```
***       P5.       ***

LDA #BYTE
JSR #BC3C
```

A $80, a $FF, illetve a $7F hexadecimális értékeknek a rutin rendre a −128, −1, ill. 127 értékeket feleleti meg.

Egybyte-os előjel nélküli egész szám átalakítása

Az alábbi rutin figyelmen kívül hagyja az előjelet:

```
***       P6.       ***
LDA #BYTE
JSR #B3A2
```

Ebben az esetben a $00 értéknek a nulla, a $80-nak a 128, a $FF-nek a 255 felel meg.

Kétbyte-os előjeles egész szám átalakítása

```
***       P7.       ***

LDY #LOW
LDA #HIGH
JSR #B395
```

Amint látjuk, a rutin hívása előtt a szám alsó byte-ját az Y regiszterbe, felső byte-ját pedig az akkumulátorba kell tölteni.
Az alábbi táblázat szemlélteti a megfeleltetést:

```
***         T12.        ***

 A     Y     Lebegopontos ertek

00    00     0
00    01     1
00    FF     255
01    00     256
7F    FF     32767
80    00     -32768
FF    FF     -1
```

Kétbyte-os előjel nélküli egész szám átalakítása
A következő rutin figyelmen kívül hagyja az előjelet:

```
***       P8.       ***

LDY #LOW
LDA #HIGH
STY $63
STA $62
```

```
LDX #$90
SEC
JSR $BC49
```

Mivel az előjel nem foglal el helyet, az ábrázolható számtartomány ebben az esetben 0-tól 65535-ig terjed:

```
***      T13.      ***

 A     Y     Lebegopontos ertek

00    00    0
00    01    1
00    FF    255
01    00    256
7F    FF    32767
80    00    32768
FF    FF    65535
```

Hárombyte-os előjeles egész szám átalakítása

Annak ellenére, hogy a gyakorlatban ritkán dolgozunk hárombyte-os egész számokkal, az értelmező tartalmazza a konvertálásukhoz szükséges rutint:

```
***       P9.       ***

LDA #LOW
LDX #MID
LDY #HIGH
STY $62
STX $63
STA $64
LDA $62
EOR #$FF
ASL  A
LDA #0
STA $65
LDX #$98
JSR $BC4F
```

A megfeleltetés táblázata:

```
***         T14.         ***

Y     X     A     Lebegopontos ertek

00    00    00    0
00    00    FF    255
00    FF    FF    65535
7F    FF    FF    8388607
80    00    00    -8388608
FF    FF    FF    -1
```

A hárombyte-os egész számok értéke a −8388608-tól 8388607-ig terjedő tartományba eshet.

A hárombyte-os előjel nélküli egész szám átalakítása

A 24 bites egészek konvertálására szolgáló rutin:

```
***       P10.       ***
LDA #LOW
LDX #MID
LDY #HIGH
JSR $AF87
JSR $AF7E
```

A lefedett számtartomány 0-tól $2^{24}-1$ = 16777215-ig

```
***          T15.          ***
Y     X     A     Lebegopontos ertek

00    00    00    0
00    00    FF    255
00    FF    FF    65535
7F    FF    FF    8388607
80    00    00    8388608
FF    FF    FF    16277215
```

Négybyte-os előjeles egész számok átalakítása

A következő rutin 32 bites egész számokat alakít lebegőpontossá. A négy byte-ot a FAC-ben a $62 címtől kezdve keresi, nevezetesen a legfelső byte-ot a $62-es, a legalsó byte-ot pedig a $65-ös címen.

```
***       P11.       ***
LDA $62
EOR #$FF
ASL  A
LDA #0
LDX #$A0
JSR $BC4F
```

A megfeleltetés táblázata

```
***          T16.          ***
$62   63   64   65      Lebegopontos ertek

 00   00   00   00      0
 00   00   00   FF      255
 00   00   FF   FF      65535
 00   FF   FF   FF      16777215
 7F   FF   FF   FF      2147483647     (2.14748365E+09)
 80   00   00   00      -2147483648    (-2.14748365E+09)
 FF   FF   FF   FF      -1
```

Négybyte-os előjel nélküli egész szám átalakítása

Az előzőhöz hasonlóan ez a rutin is az FAC-ben keresi az átalakítandó számot.
A lefedett számtartomány 0-tól $2^{32}-1$ = 4294967295-ig terjed

```
***       P12.       ***
SEC
LDA #0
LDX #$A0
JSR $BC4F
```

A megfeleltetés táblázata

```
***          T17.          ***
$62   63   64   65      Lebegopontos ertek

00    00   00   00      0
00    00   00   FF      255
00    00   FF   FF      65535
00    FF   FF   FF      16777215
7F    FF   FF   FF      2147483647     (2.14748365E+09)
80    00   00   00      2147483648     (2.14748365E+09)
FF    FF   FF   FF      4294967295     (4.29496729E+09)
```

A bemutatott rutinokra minden olyan gépi kódú programban szükség van, amelyben egész számokkal lebegőpontos aritmetikai műveleteket kell elvégezni.

1.3 ÁTALAKÍTÁS EGÉSZ TÍPUSÚ (INTEGER) FORMÁTUMRA

A lebegőpontos számok egész típusúvá alakításához egyetlen rutin áll rendelkezésünkre. Az átalakítás eredménye mindig egy 4 byte-os előjeles egész szám. A rutin hívása előtt a lebegőpontos számot az FAC-be kell tölteni.
A rutin kezdőcíme $BC9B, hívása:

```
JSR $BC9B
```

Mivel az átalakítás eredménye csak 2^{31}-nél kisebb szám lehet, a rutin megvizsgálja, hogy a lebegőpontos szám kitevőrésze kisebb-e, mint $A0.
Átalakítás után az egész szám a $62-es címtől (legfelső byte, előjellel) a $65-ös byte-ig (legalsó byte) található.
Nézzünk erre egy példát:
Legyen az FAC tartalma 10.

```
***       T18.      ***

         EX M1 M2 M3 M4 SGN
>: 0061 84 A0 00 00 00 00
```

A JSR $BC9B utasítás végrehajtása után az eredmény:

```
***       T19.      ***
>: 0061 84 00 00 00 0A 00
           -----------
```

Ha az FAC tartalma nem egész szám, az átalakítás után a szám tizedesvessző utáni részét az értelmező levágja, az INT függvényhez hasonlóan.

Ha pl. az FAC tartalma 321.25

```
***       T20.      ***
         EX M1 M2 M3 M4 SGN
>: 0061 89 A0 A0 00 00 00
```

az eredmény

```
***       T21.      ***
>: 0061 89 00 00 01 41 00
           -----------
```

azaz $41+$100 = 65+256 = 321. Negatív, tizedesrészt tartalmazó egész szám esetében a számhoz legközelebb eső, nála nem nagyobb negatív egész számot kapjuk, pl. −0.5-ből −1 lesz.

```
***        T22.        ***
          EX  M1  M2  M3  M4  SGN
>1  0061  80  80  00  00  00  FF
```

Az átalakítás eredménye:

```
***        T23.        ***
>1  0061  80  FF  FF  FF  FF  FF
          ----------
```

azaz −1.

1.4 A BASIC ÉRTELMEZŐ (INTERPRETER) ARITMETIKAI RUTINJAI

A számok beolvasásához, kiírásához és átalakításához szükséges rutinok tárgyalása után rátérünk a műveletvégző rutinok ismertetésére.
Az értelmező tartalmazza az öt alapművelet – összeadás, kivonás, szorzás, osztás és hatványozás – elvégzéséhez szükséges rutinokat.
A műveletek elvégzése előtt az egyik operandust az FAC-be, a másikat az ARG-ba kell betölteni. A rutinhívás után az eredmény mindig az FAC-ben található. A rutinok címei:

```
***        T24.        ***

OSSZEADAS          FAC := ARG + FAC      $B86A
KIVONAS            FAC := ARG - FAC      $B853
SZORZAS            FAC := ARG * FAC      $BA2B
OSZTAS             FAC := ARG / FAC      $BB12
HATVANYOZAS        FAC := ARG ^ FAC      $BF7B
```

Hívás előtt az akkumulátor tartalma az FAC ($61) kitevője. Ha a kitevő nulla, megegyezés szerint az FAC értéke is nulla. A különleges esetek kezelése az alábbiak szerint történik: ARG + 0 = ARG; ARG*0 = 0; ARG/0: a DIVISION BY ZERO hibaüzenetet eredményezi; végül ARG↑0 = 1. Ha olyan rutint hívunk, amely az FAC és az ARG tartalmát meghatározza, a kitevők értékét automatikusan megkapjuk az akkumulátorban.
Vizsgáljuk meg a 7*13 = 91 művelet elvégzését:

```
***        T25.        ***

 7 = 83 E0 00 00 00 00
13 = 84 D0 00 00 00 00
```

A fenti értékeket betöltjük a lebegőpontos akkumulátorokba, az FAC kitevőjét az akkumulátorba, majd meghívjuk a rutint:

```
***        T26.        ***

>: 0061 83 E0 00 00 00 00
>: 0069 84 D0 00 00 00 00

>, 1000    A5 61       LDA $61
>, 1002    20 2B BA    JSR $BA2B
>, 1005    00          BRK

>G 1000

B*
    PC   IRQ   SR AC XR YR SP   NV-BDIZC
>; 1006 EA31 A0 87 B6 00 F8   10100000

>: 0061 87 B6 00 00 00 00
```

Váltsuk át az eredményt decimális számmá:

$1.0110110*2^6 = 1011011$
$= 64+16+8+2+1 = 91$

Most végezzük el a $3^7 = 2187$ hatványozást:

```
***       T27.       ***
3 = 82 C0 00 00 00 00
7 = 8E E0 00 00 00 00
```

Az értékek átadása után meghívjuk a hatványozó rutint:

```
***       T28.       ***
>: 0061 83 E0 00 00 00 00
>: 0069 82 C0 00 00 00 00

>, 1000    A5 61        LDA $61
>, 1002    20 7B BF     JSR $BF7B
>, 1005    00           BRK

>G 1000

B*
    PC  IRQ  SR AC XR YR SP  NV-BDIZC
>; 1006 EA31 22 00 61 00 F8  10100000

>: 0061 8C 88 B0 00 02 00 00
```

A kapott eredményt alakitsuk ismét decimális számmá:

$1.000\ 1000\ 1011\ 0000\ 0000\ 0000\ 0000\ 0010 * 2^{11} =$
$1000\ 1000\ 1011.0000\ 0000\ 0000\ 0000\ 0010$

$= 2^{11} + 2^7 + 2^3 + 2^1 + 2^0 + 2^{-19}$
$= 2048 + 128 + 8 + 2 + 1 + 1.9 * 10^{-6}$
$= 2187.0000019$

Az utolsó helyiértéken eltérés mutatkozik a valódi eredménytől. Azt, hogy a bináris szám decimálissá alakitása csak 9 jegyre pontos, a következőképpen mutathatjuk meg: a

```
PRINT 3↑7
```

utasítás eredménye 2187. Mégis a

```
PRINT 3↑7 - 2187
```

utasítás végrehajtásakor az

```
1.90734863E-06
```

eredményt kapjuk. A hatványozást az értelmező a következők szerint végzi el:

$A \uparrow B \Rightarrow EXP\ (B * LOG\ (A))$

A fenti eltérés érthetővé válik, ha figyelembe vesszük, hogy az EXP és a LOG függvények közelítő értékekkel dolgoznak. A hatványozás az egyik leglassúbb

aritmetikai művelet – kb. 50 ms – ami szintén érthető, hiszen végrehajtása közben az értelmező két függvényt is meghív. Célszerű tehát a csak egész kitevőt tartalmazó hatványműveleteket a szorzásra visszavezetni egyrészt a pontosság, másrészt a végrehajtási sebesség érdekében.
Érdemes a programba 3 ↑ 2 helyett 3*3-at írni.
A szorzás ez esetben 20-szor gyorsabb a hatványozásnál. A műveletek végrehajtási idejéről a későbbiekben közlünk egy táblázatot.
Eddigi ismereteinket csak akkor tudjuk valóban hasznosítani, ha tisztában vagyunk azzal, hogy a BASIC értelmező hogyan kezeli a változókat. A nullásiapon találhatóak azok a mutatók, amelyek a BASIC program, az egyszerű- és tömbváltozók és a füzérek tárbeli helyét meghatározzák. A 128-as üzemmód megfelelő címeit a függelékbeli táblázat tartalmazza.
A gép bekapcsolásakor a BASIC kezdete a $801 = 2049-es, vége pedig a $A000 = 40960-as cím. Ha leírunk egy programsort, pl. a következőt:

```
10 A = 1
```

a tár tartalma a $0801-es címtől:–

```
A következő sor címe
Sorszám
A programsor
0
```

A tárba monitorprogrammal közvetlenül bepillanthatunk:

```
***        T29.        ***

>M 0800 080F
>: 0800 00 09 0B 0A 00 41 B2 31
>: 0808 00 00 00
```

A programmutatók értéke:

```
***        T30.        ***

>M 002B 0037
>: 002B 01 0B 0B 0B 0B 0B 0B 0B
>: 0808 00 A0 00 00 00 A0
```

Értelmezzük a fentieket! A ($2B/$2C) címek tartalma a $801-es cím, ahol a következő programsor címe található alsó, ill. felső byte-ra bontva, azaz $0809. Ezt követi a sorszám alsó és felső byte-ja: $000A = 10. A sorszám után a program szövege következik: $41 = "A", a $B2 az "=" jel értelmező kódja, és az "1" ASCII kódja azaz $31. A sor végét a nulla byte jelzi.
A program a tárban hasonló szerkezetben folytatódna. Mivel a fenti program egyetlen sorból áll, a következő programsor címe $0000, ami megegyezés szerint arra utal, hogy a programnak vége.
A következő cím a $080B a ($2D/$2E) címeken található, és a program végére, ill. egyben az egyszerű változók kezdetére mutat. Mivel eddig egyetlen változó sem kapott még értéket, az egyszerű változók és a tömbváltozók vége azonos tárcímen van, mutatójuk tehát azonos értékű. Ha lefuttatjuk a programot a RUN paranccsal, az A változó értéket kap.

```
***       T31.       ***

>M 0800 0810
>: 0800 00 09 08 0A 00 41 B2 31
>: 0808 00 00 00 41 00 81 00 00
>: 0810 00 00

>M 002B 0037
>: 002B 01 08 0B 08 12 08 12 08
>: 0033 00 A0 00 00 00 A0
```

A változókezdet ($2D/$2E) most is $080B, a változóterület vége azonban ($2F/$30) $0812. A változóterület hossza: $0812 − $080B = $0007 = 7 byte, és tartama a következő:

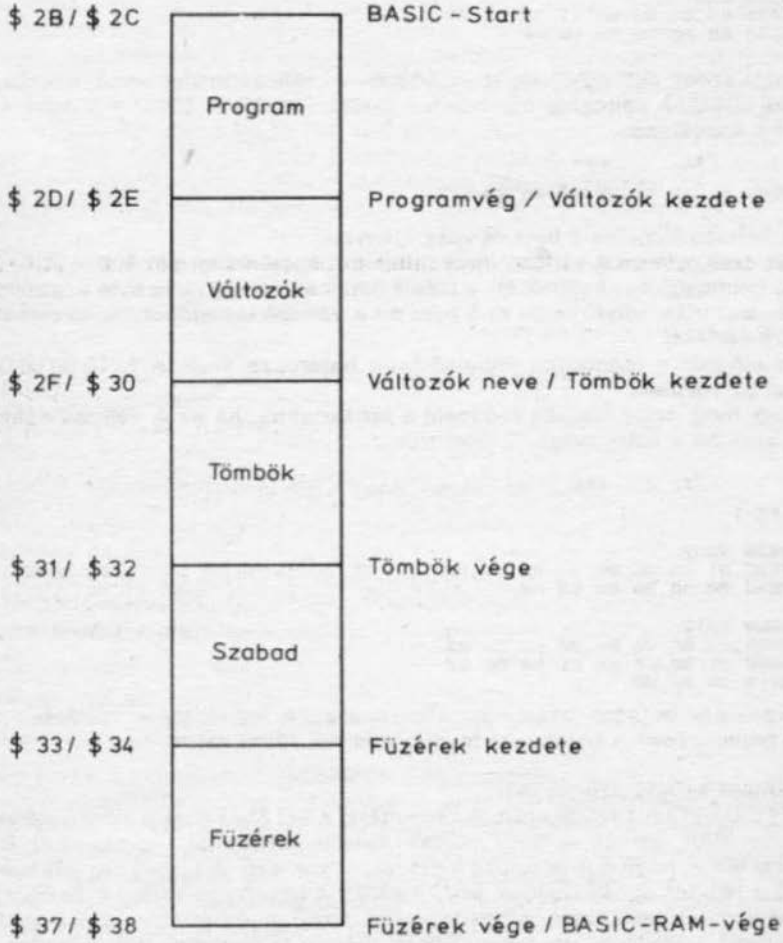
```
***       T32.       ***

>: 080B 41 00 81 00 00 00 00
```

Minden változó tárolása 7 byte-ot vesz igénybe.
Ebből az első byte-on a változó neve található, esetünkben $41 $00 = A.
Mivel az azonosító egy betűből áll, a másik betű helyén egy nulla byte található.
Az azonosító után helyezkedik el 5 byte-on a változó lebegőpontos formában tárolt számértéke.
A szám előjelét a mantissza legfelső bitje határozza meg. A 81 00 00 00 00 megfelel az 1-esnek.
Vizsgáljuk meg, hogy hogyan módosul a tár tartalma, ha az A változó egész típusú, azaz ha a neve mögé %-jelet írunk:

```
***       T33.       ***
 10 A%=1

>M 002B 0037
>: 002B 01 08 0C 08 13 08 13 08
>: 0033 00 A0 00 00 00 A0

>M 0800 0810
>: 0800 00 0A 08 0A 00 41 25 B2
>: 0808 31 00 00 00 C1 80 00 01
>: 0810 00 00 00
```

A program egy byte-tal, azaz egy jellel hosszabb lett. Hogyan módosult a változó bejegyzése? A bejegyzés továbbra is hét byte hosszú.

Felismerhető az A% azonosító?
Ha a $C1 $80 hexadecimális számok bitmintáját a $41 $00 számok bitmintájával összehasonlítjuk, azt látjuk, hogy mindkét byte legfelső bitje 1-re változott. Ez jelzi azt a tényt, hogy egész típusú változókról van szó. A következő két byte tartalma a 16 bites egész számok, értéke $0001. A következő három byte egész típusú változó tárolása esetén kihasználatlan. Azzal, hogy valós változó helyett egész típusú változóval dolgozunk, nem tudunk tárterületet megtakarítani. A program végrehajtási sebessége sem csökken, mivel minden művelet elvégzése előtt az értelmező az operandusokat lebegőpontossá alakítja, a végrehajtás több időt vesz igénybe, mintha valós típusú változókkal dolgoznánk.



A Commodore 64-es tárfelosztása

Milyen módon tárolja az értelmező a füzéreket?
Próbáljuk ezt is felderíteni egy programsor segítségével:

```
10 A$ = "STRING"
```

Futtassuk le ezt a programot és vizsgáljuk meg a monitorprogrammal a tár tartalmát.

```
***       T34.       ***
>M 002B 0037
>: 002B 01 08 13 08 1A 08 1A 08
>: 0033 00 A0 00 00 00 A0

>M 0800 0818
>: 0800 00 11 08 0A 00 41 24 B2
>: 0808 22 53 54 52 49 4E 47 22
>: 0810 00 00 00 41 80 06 09 08
>: 0818 00 00
```

A változóterület most is a $8013-as címen kezdődik és tartalma a következő:

```
***       T35.       ***
>: 0813 41 80 06 09 08 00 00
```

Az első két byte tartalma most is a változó azonosítója. A változó típusára (füzér) az értelmező azzal utal, hogy az azonosító második karakterének legfelső bitjét 1-re állítja. Az A betű hexadecimális megfelelője, a $41 és $00, füzérváltozó esetén $41 $80-ra módosul.
Az azonosító két byte-ja után következő három byte tartalmát az alábbiak szerint értelmezhetjük: az első byte ($06) a füzérváltozó hossza, az ezt következő két byte pedig azt a címet tartalmazza, ahol a tényleges füzér a tárban kezdődik, esetünkben a $0809-es.
Figyeljük meg, hogy ez a cím a programszöveg belsejében közvetlenül az idézőjel utáni byte címe. A valódi füzérterület tehát még üres, és mindaddig az is marad, amíg a füzért meg nem változtatjuk, azaz nem végzünk az A$ változóval valamilyen műveletet. Pl.:

```
10 A$ = "STRING"
20 A$ = LEFT$ (A$,3)
```

```
***       T36.       ***
>M 002B 0037
>: 002B 01 08 22 08 29 08 29 08
>: 0033 FD 9F 00 A0 00 A0

>M 0800 0828
>: 0800 00 11 08 0A 00 41 24 B2
>: 0808 22 53 54 52 49 4E 47 22
>: 0810 00 20 08 14 00 41 24 B2
>: 0818 C8 28 41 24 2C 33 29 00
>: 0820 00 00 41 80 03 FD 9F 00
>: 0828 00
```

A változóterület kezdete a $0822-es cím

```
***       T37.       ***
>: 0822 41 80 03 FD 9F 00 00
```

A változó neve után ismét a hossza következik, esetünkben 3, majd a füzér kezdőcíme: $9FFD, ami egyben a füzérterület alsó határa is. Ha betekintünk erre a tárterületre, azt látjuk, hogy tartalma valóban "STR".

```
***       T38.       ***
>: 9FFD 53 54 52
```

Hogyan kezeli az operációs rendszer a tömbváltozókat? Ennek felderítéséhez töröljük a régi programot, és írjunk helyette újat:

```
10 DIM A(500)
```

Vizsgáljuk meg a tár tartalmát:

```
***       T39.       ***
>M 002B 0037
>: 002B 01 0B 10 0B 10 0B E0 11
>: 0033 00 A0 00 00 00 A0
```

Mivel a programban egyszerű változót nem használtunk, a változóterület üres, kezdő- és végcíme azonos: $0810.
Ez a cím egyben a tömbváltozók kezdete.
A tömbváltozók által elfoglalt terület mérete jelenleg: $11E0 − $0810 = $09D0, azaz 2512 byte, a terület végcíme ugyanis $11E0.
Vizsgáljuk meg közelebbről ennek a területnek a tartalmát:

```
***       T40.       ***
>M 0810 0820
>: 0810 41 00 D0 09 01 01 F5 00
>: 0818 00 00 00 00 00 00 00 00
>: 0820 00 00 00 00 00 00 00 00
```

Az első két byte tartalma most is a változó azonosítóját tartalmazza (A). A következő két byte tartalma megadja a tömbváltozó által elfoglalt terület méretét ($09D0), amint fent kiszámoltunk. Az ezt követő 1-es jelzi, hogy egyméretű, azaz egyindexes tömb számára foglaltunk helyet. Végül az utolsó két byte tartalma a tömb elemeinek száma: $01F5 = 501.
A változó adatainak bejegyzése után kezdődnek az egyes elemek sorban, amelyek értéke a DIM utasítás végrehajtása után 0. Ha végrehajtjuk az A(0) = 10 : A(1) = 11 parancsokat, a tár tartalma megváltozik:

```
***       T41.       ***
>M 0810 0820
>: 0810 41 00 D0 09 01 01 F5 84
>: 0818 20 00 00 00 84 30 00 00
>: 0820 00 00 00 00 00 00 00 00

84 20 00 00 00 => 10; 84 30 00 00 00 => 11
```

Hogyan tárolódnak a többindexes tömbök?
Töröljük a programot és hajtsuk végre a

DIM B(1, 2, 3)

parancsot.

A dimenzionált tömb bejegyzése a $0803-as címen kezdődik:

```
***        T42.        ***

>M 002B 0037
>: 002B 01 0B 03 0B 03 0B 86 0B
>: 0033 00 A0 00 00 00 A0

>M 0803 0813
>: 0803 42 00 83 00 03 00 04 00
>: 080B 03 00 02 00 00 00 00 00
>: 0813 00 00 00 00 00 00 00 00
```

A tömb azonosítójának (B) kódját könnyen felismerhetjük: $ 42. A tömb hossza ez esetben $0083 = 131 byte. A következő byte tartalma (3) az indexek számát mutatja. Végül három byte-ot foglalnak el az indexek felső határai fordított sorrendben: $0004, $0003, ill. $0002, amelyek rendre megfelelnek a 3, 2 és 1 értékeknek.

A tömb elemeinek sorrendje a tárban:

B(0,0,0)
B(1,0,0)
B(0,1,0)
B(1,1,0)
B(0,2,0)
B(1,2,0)
B(0,0,1)
B(1,0,1)
B(0,1,1)
B(1,1,1)
B(0,2,1)
B(1,2,1)
B(0,0,2)
B(1,0,2)
B(0,1,2)
B(1,1,2)
B(0,2,2)
B(1,2,2)
B(0,0,3)
B(1,0,3)
B(0,1,3)
B(1,1,3)
B(0,2,3)
B(1,2,3)

A sorrendből látható, hogy az első index változik a leggyorsabban, az utolsó pedig a leglassabban.

Ha a tömböt egész típusú változóként dimenzionáljuk, akkor az egyes elemek csak két byte-ot foglalnak el, ami jelentős helymegtakarítás a valóskénti tárolással szemben. A füzértömbök tárolásánál minden elem számára a füzér tartalmának tárolásán kívül három byte-ra van szükség, ebből egy a füzér hosszát, a maradék kettő pedig a füzér tárbeli címét tartalmazza.

Eddigi ismereteink alapján általánosan is kiszámíthatjuk, hogy egy tömb mekkora helyet foglal el a tárban:

$$T = 5 + 2*N + E*SZOR\ (N_i + 1)$$

ahol T: a szükséges tárterület
N: az indexek száma
E: az elemenkénti helyigény
2 – egész típusú változónál (INTEGER)
5 – való típusú változónál (REAL)
3 – füzéreknél (STRING)
SZOR $(N_i + 1)$: az indexhatárok eggyel megnövelt értékeinek szorzata.

A konstans (5) az azonosító tárolásához szükséges 2 byte-ból, az elfoglalt terület méretének tárolásához szükséges 2 byte-ból, ill. az indexek számának tárolásához szükséges 1 byte-ból tevődik össze. A 2N byte az indexek felső határainak tárolására szolgál.
Az egyes elemek tárolási igényét (E) változó típusonként lehet meghatározni. A fenti program A(500) tömbjének tárolásához szükséges tárterület a képlet alapján:

$T = 5 + 2*1 + *(501)$ azaz
$T = 2512$ byte

A B(1,2,3) háromindexes tömb helyigénye:

$T = 5 + 2*3 + 5*(2*3*4)$ azaz

$T = 131$ byte

Az A% (10, 10, 10) egész típusú tömb helyigénye:

$T = 5 + 2*3 + 2*(11*11*11)$ azaz

$T = 2673$ byte

Az A$ (100, 100) füzértömb nem nagyon fér el a tárban.

$T = 5 + 2*2 + 3*(101*101)$ azaz

$T = 30603$ byte

A tömb bejegyzése már önmagában elfoglal 30 kbyte-nyi területet. Az 10201 elem tényleges tárolására mindössze 8 kbyte marad.

1.5 A BASIC ÉRTELMEZŐ LEBEGŐPONTOS FÜGGVÉNYEI

A lebegőpontos alapműveletek tárgyalása után térjünk át a BASIC értelmező lebegőpontos függvényeinek ismertetésére.
A függvény általános alakja:

$$Y = F(x)$$

ahol x a független változó (argumentum), Y pedig a függvény értéke.
A lebegőpontos függvények hívása előtt az argumentumot (x-et) az FAC-be kell beírni, és az értelmező az eredményt ugyanide tölti.

A lebegőpontos függvények:

***	T43.	***	
Nev	Cim	Vegrehajtasi idö	Müvelet
ABS	$BC58	0.0 ms	Abszolutertekfüggv.
ATN	$E30E	44.6 ms	Arcus tangens függv.
COS	$E264	27.9 ms	Cosinus függveny
EXP	$BFED	26.6 ms	Hatvanyra emeles (alap=e)
FRE	$B37D	0.6 ms	Szabad memoria vizsgalata
INT	$BCCC	0.9 ms	Egeszresz függveny
LOG	$B9EA	22.2 ms	Term.alapu logaritmus
POS	$B39E	0.3 ms	Kurzorpozicio
RND	$E097	3.5 ms	Veletlenszam-függveny
SGN	$BC39	0.4 ms	Előjel-függveny
SIN	$E26B	24.5 ms	Sinus függveny
SQR	$BF71	51.2 ms	Negyzetgyökvonas
TAN	$E2B4	49.8 ms	Tangens függveny

A számítási időket az $X = \pi$ argumentummal mértük le. A táblázatból kitűnik, hogy az egyes függvények számítási ideje meglehetősen eltérő.
Az ún. *transzcendens* függvények (COS, EXP, LOG, SIN, TAN, ATN) végrehajtása jóval több időt igényel, mint az egyéb függvényeké.
Ennek magyarázata abban rejlik, hogy a transzcendens függvények értékének kiszámítása nem vezethető vissza egyszerűen a négy alapműveletre. A számításokat az értelmező korlátozott pontosságú közelítő eljárásokkal végzi el. Legtöbb esetben a közelítés képlete az alábbi *polinom* formájában adható meg:

$$y = a_0 + a_1*x + a_2*x^2 + a_3*x^3 + a_4*x^4 + \ldots$$

Minél több tagot veszünk figyelembe, a számítás eredménye annál pontosabb, de a végrehajtás ideje is annál nagyobb.
A fenti polinom értékének kiszámításához

$$1+2+3+4+5 = 15$$

szorzási és 5 összeadási művelet elvégzésére van szükség.
A számítást egyszerűsíti a matematikából jól ismert Horner-elrendezés, amely szerint a polinom értéke az alábbi zárójeles kifejezésekkel is kiszámítható.

$$y = ((((a_5*x + a_4)*x + a_3)*x + a_2)*x + a_1)*x + a_6$$

Ezzel a módszerrel ugyanazon érték kiszámításához már csak 5 szorzás és 5 összeadás elvégzése szükséges.
Általánosan egy N-edfokú polinom értékének kiszámításához az első módszer szerint $N(N-1)/2$ szorzásra és N összeadásra van szükség, míg ha Horner-elrendezéssel számolunk, a szükséges szorzások és összeadások száma egyaránt N.
A módszer egyszerűségét szemlélteti az alábbi BASIC program is.

```
***       P13.       ***
100 Y=A(N)
110 FOR I=N-1 TO 0 STEP-1
120 Y=Y*X+A(I)
130 NEXT
```

A program kiszámítja az N-edfokú polinom helyettesítési értékét az X helyen és az eredményt az Y változóban tárolja.
A polinom együtthatóit az A indexes változó tartalmazza A(0)-tól A(N)-ig.
Ez az eljárás a transzcendens függvények kiszámításának leglényegesebb része.
A rutin önmagában is használható. Hívása előtt a független változó (x) értékét az FAC-ben kell tárolni. A polinom együtthatóinak tárolási rendje:

– a polinom fokszáma (n)
– az n-edfokú tag együtthatója
– az n – 1-edfokú tag együtthatója
⋮
– a 0-dfokú tag együtthatója

A polinom fokszáma egy byte-ot foglal el, az együtthatók pedig egyenként öt byte-ot.
A rutin hívásakor át kell adni az együtthatók tárbeli elhelyezkedésének kezdőcímét úgy, hogy a cím alsó byte-ja az akkumulátorban, felső byte-ja pedig az Y regiszterben legyen.
Egy átlagos assembler programmal kissé körülményes a lebegőpontos konstansok elhelyezése a programban.
Meg kell ugyanis keresni előzetesen monitorral a változóterületen az egyes változók bejegyzésének kezdőcímét, majd ezt a címet .BYT utasítással a programban el kell helyezni. A PROFI-MAT 2.0 program az eljárást megkönnyíti azzal, hogy rendelkezik egy olyan utasítással (.FLP), amely közvetlenül beépíti a konstansokat a programba, az assembler program pedig átalakítja ezeket öt byte-os lebegőpontos formátumra.
Alkalmazzuk eddigi ismereteinket az alábbi polinom értékének kiszámítására:

$$Y = 0.7 + 2.5*x + 8.2*x^2 - 2.3*x^3 + 0.5*x^4$$

```
***       P14.       ***
PROFI-ABS 64 V2.0

100:  033C                    .OPT P,00
110:                          ;
120:                          ; Polinomszamitas
130:                          ;
140:                          *=    828     ; Kazettapuffer
150:                          ;
160:  E059                    POLYNOM =   $E059
170:                          ;
180:  033C A9 43              LDA   #< KOEFF
```

```
190:  033E A0 03          LDY  #> KOEFF
200:  0340 4C 59 E0       JMP  POLYNOM
210:                      ;
220:  0343 04             KOEFF   .BYT 4        ; Fokszam
230:  0344 80 00 00       .FLP 0.5    ; A(4)
240:  0349 82 93 33       .FLP -2.3   ; A(3)
250:  034E 84 03 33       .FLP 8.2    ; A(2)
260:  0353 82 20 00       .FLP 2.5    ; A(1)
270:  0358 80 33 33       .FLP 0.7    ; A(0)
280:                      ;
 033C-035D
NO ERRORS
```

A fenti program feladata mindössze a paraméterek átadása és a belső polinomfüggvény hívása.
A program első részében átadjuk a függvény kezdőcímét, a második részben pedig a polinom együtthatóit a kitevők szerinti növekvő sorrendben.
A kérdés most már csak az, hogy hogyan használhatjuk a fenti gépi kódú rutint egy BASIC programban, ill. hogy adhatjuk át a független változó (x) értékét és hogyan kapjuk vissza a végeredményt? A válasz nagyon egyszerű. A saját készítésű függvényt a beépített függvényekhez (SIN, COS stb.) hasonlóan használhatjuk.
Erre szolgál az értelmező USR függvénye, amely kifejezetten a felhasználó által meghatározott (user) függvények hívását végzi el. A gépi kódú rutin kezdő címét hívás előtt közölni kell az értelmezővel a 785/786 ($0311/$0312) címeken alsó és felső byte-ra bontva. Esetünkben ezt a következő POKE utasításokkal tehetjük meg:

```
POKE 785, 828 AND 255 : POKE 786, 828/256
```

Ha a gépi kódú program már a tárban van, és a fenti BASIC sort végrehajtottuk, a

```
? USR(1)
```

parancs hatására megjelenik a képernyőn a polinom X = 1 helyen vett helyettesítési értéke.
Az eredményt ellenőrizhetjük:

$$y = 0.7 + 2.5 + 8.2 - 2.3 + 0.5 = 9.6$$

Az alábbi BASIC sorral tesztelhetjük a gépi kódú rutint:

```
FOR I = -5 TO 5 : PRINT USR (I) : NEXT
```

A futás eredménye:

```
 793.2
 397.1
 169.6
  54.9
   9.2
    .7
   9.6
```

28.1
60.4
122.7
243.2

A gépi kódú rutint feltétlenül érdemes beépiteni minden olyan programba, amelyben sokszor kell egy polinom értékét kiszámitani. Végrehajtási ideje (12.5 ms) ugyanis jóval kevesebb, mintha BASIC utasitásokkal irtuk volna meg (45 ms). Minél bonyolultabb egy képlet, annál számottevőbb a BASIC és a gépi kódú program végrehajtási ideje közötti különbség.
Ha a fenti programot másik polinom értékének kiszámitására is alkalmassá akarjuk tenni, az együtthatókat ki kell cserélnünk úgy, hogy ha valamely hatványérték a polinomban nem szerepel, a megfelelő helyre 0 értéket kell beirni.
A következő példában egy olyan függvényt készitünk, amely kiszámitja egy egész szám faktoriálisát. Az n egész szám faktoriálisán az 1-től n-ig terjedő egész számok szorzatát értjük:

$n! = 1*2*3* \ldots *n$

Pl.

$5! = 1*2*3*4*5 = 120$
$7! = 1*2*3*4*5*6*7 = 5040.$

A matematikában a faktoriális fogalmát valós számokra is kiterjesztik. Egy 0 és 1 közé eső szám faktoriálisát egy polinom közelitő értékeként határozzák meg. Az 1-nél nagyobb számok faktoriálisát pedig a következőképpen lehet visszavezetni az előző esetre:

$4.3! = 4.3*3.3*2.3*1.3*0.3!$

A 0.3! értékét az alábbi együtthatókat tartalmazó polinom helyettesitési értékeként számithatjuk ki:

$a_0 = 1$
$a_1 = -0.57719\ 1652$
$a_2 = .98820\ 6891$
$a_3 = -.89705\ 6937$
$a_4 = .91820\ 6857$
$a_5 = -.75670\ 4078$
$a_6 = .48219\ 9394$
$a_7 = -.19352\ 7818$
$a_8 = .03586\ 8343$

A polinom értékét meghatározó rutin a P.14 program alapján:

```
***       P15.       ***

PROFI-ASS 64 V2.0

100:   033C                    .OPT P1,00
110:                       ;
120:                       ; A faktorialis kiszamitasanak polinomja
130:                       ;
140:   033C                    *=   828      ; Kazettapuffer
150:                       ;
```

```
160:    E059              POLYNOM  =     $E059
170:                      ;
180:    033C A9 43                 LDA   #< KOEFF
190:    033E A0 03                 LDY   #> KOEFF
200:    0340 4C 59 E0              JMP   POLYNOM
210:                      ;
220:    0343 08           KOEFF    .BYT 8        ; 8-adfoku polinom
230:    0344 7C 12 EA              .FLP .035868343
240:    0349 7E C6 2C              .FLP -.193527818
250:    034E 7F 76 E2              .FLP .482199394
260:    0353 80 C1 B7              .FLP -.756704078
270:    0358 80 6B 0F              .FLP .918206857
280:    035D 80 E5 A5              .FLP -.897056937
290:    0362 80 7C FB              .FLP .988206891
300:    0367 80 93 C2              .FLP -.577191652
310:    036C 81 00 00              .FLP 1
 033C-0371
NO ERRORS
```

A P.15. program birtokában PRINT USR(X) utasítással kiírathatjuk a képernyőre bármely 0 és 1 közé eső valós szám faktoriálisát, pl:

? USR (.1) ⇒ 0.951350564
? USR (.5) ⇒ 0.886227246

A (0,1) számtartományon kívül eső számok faktoriálisának kiszámítása sem okoz gondot. A következő program tartalmazza a szükséges bővítéseket.

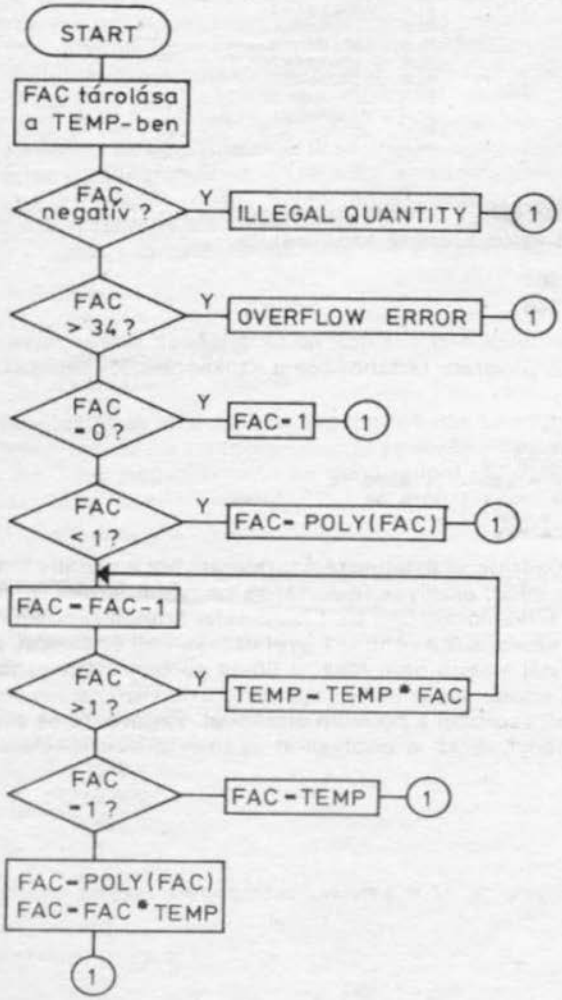
```
***      P16.      ***

10 INPUT "ARGUMENTUM";X
20 IF X<0 OR X>33 THEN 10
30 IF X=0 THEN Y=1 : GOTO 70
40 Y=X : IF X<1 THEN Y = USR(X) : GOTO 70
50 X=X-1 : IF X>1 THEN Y=Y*X : GOTO 50
60 IF X<>1 THEN Y = Y * USR(X)
70 PRINT "FAKTORIALIS = ";Y
```

A 20-as programsorban kizárjuk az értelmezési tartományból a negatív számokat, illetve azokat a számokat, amelyek faktoriálisa nagyobb lenne 1E38-nál. A 30-as sorban a 0-hoz definíciószerűen az 1 faktoriális értéket rendeljük. Az 50-es sorban mindaddig szorozzuk a rendre 1-gyel csökkentett értékeket, amíg a levonás eredménye 1-nél kisebb nem lesz. A 60-as sorban megvizsgáljuk, hogy a beolvasott szám egész szám volt-e. Ha nem, akkor a szorzás eddigi eredményét még meg kell szorozni a polinom értékével. Végül a 70-es sorban kiírathatjuk a végeredményt, azaz a beolvasott számértéknek megfelelően például a következőket:

0 ⇒ 1
1 ⇒ 1
1.5 ⇒ 1.32934087
2 ⇒ 2
3 ⇒ 3
0.5 ⇒ .886227246
7.35 ⇒ 10287.3151

Az elkészült programot érdemes lenne teljes egészében gépi kódban megírni, hiszen a leglényegesebb rész, a polinom kiszámítását végző rutin már készen van. A program átírása közben további belső aritmetikai rutinokkal ismerkedhetünk meg.

Készítsük el a feladat folyamatábráját.



Próbáljuk ki az elkészült programot, de ne feledkezzünk el a USR vektor kezdő címének beállításáról, enélkül ugyanis az ILLEGAL QUANTITY hibaüzenetet kapnánk.

```
***      P17.      ***

PROFI-ASS 64 V2.0

100:   033C                    .OPT P1,00
110:   033C          START     =    828      ; Kazettapuffer
120:   B97E          OVERFLOW  =    $B97E    ; OVERFLOW ERROR
130:   B248          ILLQUAN   =    $B248    ; ILLEGAL QUANTITY
140:                 ;
150:   BBD4          FACMEM    =    $BBD4    ; FAC tarolasa
160:   BBA2          MEMFAC    =    $BBA2    ; FAC betoltese
170:   BC5B          VERGLCH   =    $BC5B    ; Konstans osszehasonlitasa FAC-vel
180:   B867          MEMPLUS   =    $B867    ; Konstans + FAC
190:   BA28          MEMMULT   =    $BA28    ; Konstans * FAC
200:   E059          POLYNOM   =    $E059    ; Polinomszamitas
210:   0066          SIGN      =    $66      ; Elojel
220:   0061          EXP       =    $61      ; Exponens
230:                 ;
240:   033C                    *=   START
250:   033C A2 F5              LDX  #< TEMP
260:   033E A0 03              LDY  #> TEMP
270:   0340 20 D4 BB           JSR  FACMEM   ; A FAC tarolasa TEMP-ben
280:                 ;
290:   0343 24 66              BIT  SIGN     ; Elojelvizsgalat
300:   0345 10 03              BPL  OK1      ; Pozitiv ?
310:   0347 4C 48 B2           JMP  ILLQUAN
320:   034A A9 F0    OK1       LDA  #< MAX
330:   034C A0 03              LDY  #> MAX   ; Mutato a konstans 34-re
340:   034E 20 5B BC           JSR  VERGLCH
350:   0351 30 03              BMI  OK2      ; Kisebb ?
360:   0353 4C 7E B9           JMP  OVERFLOW
370:   0356 A5 61    OK2       LDA  EXP      ; FAC = 0
380:   0358 F0 52              BEQ  EGYENLO  ; akkor 1
390:   035A A9 E6              LDA  #< EINS
400:   035C A0 03              LDY  #> EINS
410:   035E 20 5B BC           JSR  VERGLCH  ; Osszehasonlitas 1-gyel
420:   0361 30 50              BMI  KISEBB
430:   0363 A9 EB    CIKLUS    LDA  #< MINUS1
440:   0365 A0 03              LDY  #> MINUS1
450:   0367 20 67 B8           JSR  MEMPLUS
460:   036A A9 E6              LDA  #< EINS
470:   036C A0 03              LDY  #> EINS
480:   036E 20 5B BC           JSR  VERGLCH  ; Osszehasonlitas 1-gyel
490:   0371 30 1F              BMI  NEXT     ; Mar nem nagyobb ?
500:   0373 A2 FA              LDX  #< TEMP2
510:   0375 A0 03              LDY  #> TEMP2
520:   0377 20 D4 BB           JSR  FACMEM   ; FAC atmeneti tarolasa
530:   037A A9 F5              LDA  #< TEMP
540:   037C A0 03              LDY  #> TEMP
550:   037E 20 28 BA           JSR  MEMMULT  ; FAC * TEMP
560:   0381 A2 F5              LDX  #< TEMP
570:   0383 A0 03              LDY  #> TEMP
580:   0385 20 D4 BB           JSR  FACMEM   ; Az eredmeny TEMP-be
590:   0388 A9 FA              LDA  #< TEMP2
600:   038A A0 03              LDY  #> TEMP2
610:   038C 20 A2 BB           JSR  MEMFAC   ; FAC visszatoltese
620:   038F 4C 63 03           JMP  CIKLUS
630:                 ;
640:   0392 A9 E6    NEXT      LDA  #< EINS
650:   0394 A0 03              LDY  #> EINS
660:   0396 20 5B BC           JSR  VERGLCH  ; FAC = 1 ?
670:   0399 D0 07              BNE  TOVABB
680:                 ;
690:   039B A9 F5              LDA  #< TEMP
700:   039D A0 03              LDY  #> TEMP
```

```
 710:  039F 4C A2 BB            JMP  MEMFAC  ; Betoltes FAC-ba
 720:                           ;
 730:  03A2 20 B6 03 TOVABB     JSR  POLY    ; A polinom szamitasa
 740:  03A5 A9 F5               LDA  #< TEMP
 750:  03A7 A0 03               LDY  #> TEMP
 760:  03A9 4C 28 BA            JMP  MEMMULT
 770:                           ;
 780:  03AC A9 E6    EGYENLO    LDA  #< EINS
 790:  03AE A0 03               LDY  #> EINS
 800:  03B0 4C A2 BB            JMP  MEMFAC  ; 1 a FAC-ban
 810:                           ;
 820:  03B3 4C B6 03 KISEBB     JMP  POLY    ; FAC = POLY(FAC)
 830:                           ;
 840:                           ; Faktorialis kiszamitasanak polinomja
 850:                           ;
 860:  03B6 A9 BD    POLY       LDA  #< KOEFF
 870:  03B8 A0 03               LDY  #> KOEFF
 880:  03BA 4C 59 E0            JMP  POLYNOM
 890:                           ;
 900:  03BD 08       KOEFF      .BYT 8       ; 8-adfoku polinom
 910:  03BE 7C 12 EA            .FLP .035868343
 920:  03C3 7E C6 2C            .FLP -.193527818
 930:  03C8 7F 76 E2            .FLP .482199394
 940:  03CD 80 C1 B7            .FLP -.756704078
 950:  03D2 80 6B 0F            .FLP .918206857
 960:  03D7 80 E5 A5            .FLP -.897056937
 970:  03DC 80 7C FB            .FLP .988206891
 980:  03E1 80 93 C2            .FLP -.577191652
 990:                           ;
1000:                           ; Tovabbi lebegopontos konstansok
1010:  03E6 81 00 00 EINS       .FLP 1
1020:  03EB 81 80 00 MINUS1     .FLP -1
1030:  03F0 86 08 00 MAX        .FLP 34
1040:                           ;
1050:                           ; Atmeneti tarolo
1060:  03F5         TEMP        *=   *+5
1070:  03FA         TEMP2       *=   *+5
]033C-03FF
NO ERRORS
```

Az előző BASIC programnál szemmel láthatóan hatékonyabb, gyorsabb gépi kódú függvényt kaptunk. Ismerkedjünk meg a felhasznált belső rutinokkal:

```
***       T44.       ***

?USR(0)          =>  1
?USR(1)          =>  1
?USR(2)          =>  2
?USR(3)          =>  6
?USR(.5)         =>  .886227246
?USR(4.5)        =>  52.3427967
?USR(-1)         =>  ILLEGAL QUANTITY ERROR
?USR(40)         =>  OVERFLOW ERROR
```

FACMEM – Ez a rutin tárolja a lebegőpontos akkumulátor (FAC) tartalmát azon a címen, amelynek alsó byte-ja az X, felső byte-ja pedig az Y regiszterben van. A tárolás 5 byte-os, rövidített formában történik.
MEMFAC – A fenti rutin ellentéte. Betölt egy lebegőpontos számot az FAC-be arról a címről, amelynek alsó byte-ja az akkumulátorban (A-ban), felső byte-ja pedig az Y regiszterben található.
VERGLCH – Ezzel a rutinnal az értelmező két valós számot hasonlít össze. Az egyik szám azon a tárcímen van, amelynek alsó byte-ja A-ban, felső byte-ja Y-ban található, a másik pedig az FAC-ben. Ha a két szám egyenlő, az akkumulátor tartalma 0, a Z kapcsoló értéke pedig 1 lesz.

Ha az első szám kisebb, akkor az akku tartalma −1 ($FF) és az N kapcsoló értéke 1, egyébként az akku tartalma 1, az N kapcsoló értéke pedig 0 lesz.

MEMPLUS – Ez a rutin két részből áll. Először betölti az A (alsó) és Y (felső) cím tartalmát az FAC-be, majd összeadja az FAC és az ARG tartalmát, és az eredményt az FAC-be helyezi el.

MEMMULT – Az előző rutinnal szerkezetileg azonos, de összeadás helyett szorzásra vonatkoztatva.

Az OVERFLOW és ILLQUAN címeken azokat a rutinokat hívtuk meg, amelyek kiírják a megfelelő hibaüzeneteket. Ha meggondoljuk, az $X>34$ esetet a programban fölösleges volt külön kezelni, hiszen az értelmező egyébként is a túlcsordulási hibaüzenetet írta volna ki.

Gyakran szükség van a következő alakú polinom értékének kiszámítására:

$$y = a_0*x+a_1*x^3+a_2*x^5+a_3*x^7+\ldots$$

A feladatot visszavezethetjük a polinom kiszámításának eredeti módszerére, mindössze annyit kell tenni, hogy x helyett az eredeti polinomba x^2-et kell helyettesíteni, majd az eredményt meg kell szorozni x-szel.

$$y = x*(a_0+a_1*(x^2)+a_2*(x^2)^2+a_3*(x^2)^3+\ldots)$$

A legtöbb beépített függvény használja ezt a rutint, ugyanis a közelítőpolinom legtöbbször a fenti alakú. A rutin hívása előtt az argumentumot általában a rutin által értelmezett tartományba kell transzformálni, majd az eredményt kiírás előtt az eredeti értéknek megfelelően vissza kell alakítani.

Számítsuk ki a rutinnal a következő polinom értékét:

$$y = 6*x+0.5*x^3-0.11*x^7$$

Az együtthatók bevitelénél ügyeljünk arra, hogy az ötödfokú tag hiányzik!

```
***      P18.      ***

PROFI-ASS 64 V2.0

100:    033C                    .OPT P,00
110:    033C                    *=   828
120:                    ;
130:    E043            POLY2   =    $E043
140:                    ;
150:    033C A9 43              LDA  #< KOEFF
160:    033E A0 03              LDY  #> KOEFF
170:    0340 4C 43 E0           JMP  POLY2
180:                    ;
190:    0343 03         KOEFF   .BYT 3        ; Fokszam
200:    0344 7D E1 47           .FLP -.11
210:    0349 00 00 00           .FLP 0
220:    034E 80 00 00           .FLP .5
230:    0353 83 40 00           .FLP 6
033C-0358
NO ERRORS
```

A program futtatásánál vegyük figyelembe, hogy az a polinom fokszámát nem a legmagasabb hatványkitevőből, hanem az együttható sorszámából határozza meg, hiszen a kitevők x^2-re vonatkoznak.

A program teszteléséhez néhány futtatási eredmény:

```
***       T45.       ***
USR(0)        =    0
USR(1)        =    6.39
USR(2)        =    1.92
USR(.75)      =    4.69625427
```

Végül a lebegőpontos számok kezelésével kapcsolatban kitérünk egy, a programozás során gyakran felmerülő feladat, nevezetesen a számhalmazok rendezésének megoldására.
Írjuk meg az alábbi rendezési algoritmust gépi kódban:

```
***       P19.       ***
100 FOR I=1 TO N : FL=0
110 FOR J=N TO I STEP -1
120 IF A(J-1)>A(J) THEN H=A(J):A(J)=A(J-1):A(J-1)=H:FL=1
130 NEXT J
140 IF FL=0 THEN RETURN
150 NEXT I : RETURN
```

Az A(N) egyindexes tömböt a 100-as sorban kezdődő szubrutinnal rendezzük, amelyet a GOSUB 100 utasítással hívunk meg.
A program az ún. „buborék-rendezés" algoritmussal dolgozik. Az algoritmus végrehajtása során rendre összehasonlítjuk az egymást követő elemeket. Ha a sorban az első elem nagyobb mint a második, a két elemet felcseréljük, és egy kapcsoló értékét 1-re állítjuk. Erre szolgál a fenti programban a két egymásba ágyazott ciklus. A belső ciklus lefutása után a kapcsoló (FL) értéke alapján megvizsgáljuk, hogy ebben a ciklusban volt-e elemcsere. Ha nem volt, akkor FL értéke nulla, ami egyben azt jelenti, hogy az elemek növő sorrendben követik egymást, tehát a rendezéssel készen vagyunk, az algoritmus végrehajtása itt megszakad. Egyébként az első ciklus befejeztével a legkisebb elem az első helyre, azaz az A(0) változóba kerül. A következő lépésben a belső ciklusnak már csak a 2 indexhatárig kell lefutnia, hiszen az első, azaz a legkisebb elem már a helyére került stb.
Ahhoz, hogy a gépi kódú rendező kellőképpen hatékony legyen, érdemes meggondolni, hogy hogyan kezeli az értelmező a tömböket.
A tömbök programbeli deklarálásakor az értelmező egy táblázatot készít, amely az egyes tömbök bejegyzéseit tartalmazza, és egy változóban tárolja a táblázat kezdőcímét.
Hogy rendezés előtt ne kelljen végigvizsgálni a táblázatot ahhoz, hogy a rendezendő tömböt megtaláljuk, érdemes a programban legelőször ezt a tömböt megadni, így biztosak lehetünk abban, hogy a bejegyzése a táblázat elejére kerül.

```
***       P20.       ***
PROFI-ASS 64 V2.0

100:   033C                    .OPT P,00
110:   002F          ARRTAB    =    $2F      ; Mutato a tombtablazatra
120:   0057                    *=   $57
130:   0057          IPNT      *=   *+2
140:   0059          JPNT      *=   *+2
150:   005B          JPNT1     *=   *+2
160:                 ;
170:   BBA2          MEMFAC    =    $BBA2
180:   BC5B          VERGLCH   =    $BC5B
190:                 ;
200:   033C                    *=   828      ; Kazettapuffer
```

```
210:                    ;
220:  033C A5 2F                LDA  ARRTAB
230:  033E 18                   CLC
240:  033F A0 02                LDY  #2
250:  0341 71 2F                ADC  (ARRTAB),Y ; A meret hozzaadasa
260:  0343 8D D9 03             STA  NPNT       ; Mutato a tomb vegere
270:  0346 C8                   INY
280:  0347 A5 30                LDA  ARRTAB+1
290:  0349 71 2F                ADC  (ARRTAB),Y
300:  034B 8D DA 03             STA  NPNT+1
310:  034E AD D9 03             LDA  NPNT
320:  0351 38                   SEC
320:  0352 E9 05                SBC  #5
330:  0354 8D D9 03             STA  NPNT
340:  0357 B0 03                BCS  L1
350:  0359 CE DA 03             DEC  NPNT+1
360:                    ;
370:  035C A5 2F          L1    LDA  ARRTAB
380:  035E 18                   CLC
390:  035F 69 07                ADC  #7
400:  0361 85 57                STA  IPNT       ; Az I mutato az A(0)-ra
410:  0363 A5 30                LDA  ARRTAB+1
420:  0365 69 00                ADC  #0
430:  0367 85 58                STA  IPNT+1
440:                    ;
450:  0369 A0 00          ILOOP LDY  #0
460:  036B 8C D8 03             STY  FLAG       ; A kapcsolo torlese
470:  036E AD D9 03             LDA  NPNT
470:  0371 85 59                STA  JPNT       ; J = N
480:  0373 AD DA 03             LDA  NPNT+1
480:  0376 85 5A                STA  JPNT+1
490:                    ;
500:  0378 A5 59          JLOOP LDA  JPNT
510:  037A 38                   SEC
510:  037B E9 05                SBC  #5
520:  037D 85 5B                STA  JPNT1      ; J-1, mutato
530:  037F AA                   TAX
540:  0380 A5 5A                LDA  JPNT+1
550:  0382 E9 00                SBC  #0
560:  0384 85 5C                STA  JPNT1+1
560:  0386 A8                   TAY
560:  0387 8A                   TXA
570:  0388 20 A2 BB             JSR  MEMFAC     ; A(J-1) FAC-ba
580:                    ;
590:  038B A5 59                LDA  JPNT
600:  038D A4 5A                LDY  JPNT+1
610:  038F 20 5B BC             JSR  VERGLCH ; Osszehasonlitasa A(J)-vel
620:  0392 30 12                BMI  NEMCSERE
630:                    ;
640:  0394 A0 04                LDY  #4
640:  0396 8C D8 03             STY  FLAG       ; A kapcsolo beallitasa
650:  0399 B1 59          SWAP  LDA  (JPNT),Y
660:  039B AA                   TAX
670:  039C B1 5B                LDA  (JPNT1),Y
680:  039E 91 59                STA  (JPNT),Y; A(J) es A(J-1)
690:  03A0 8A                   TXA             ; felcserelese
700:  03A1 91 5B                STA  (JNPNT1),Y
710:  03A3 88                   DEY
720:  03A4 10 F3                BPL  SWAP
730:                    ;
740:  03A6 A5 59          NEMCSERE LDA  JPNT
750:  03A8 38                   SEC
750:  03A9 E9 05                SBC  #5         ; J = J-1
760:  03AB 85 59                STA  JPNT
770:  03AD B0 02                BCS  TESTJ
780:  03AF C6 5A                DEC  JPNT+1
790:                    ;
800:  03B1 C5 57          TESTJ CMP  IPNT
810:  03B3 D0 C3                BNE  JLOOP
820:  03B5 A5 5A                LDA  JPNT+1  ; I = J ?
830:  03B7 C5 58                CMP  IPNT+1
```

```
840:    03B9 D0 BD                BNE  JLOOP
850:                      ;
860:    03BB AD D8 03             LDA  FLAG     ; Nem volt csere ?
870:    03BE F0 17                BEQ  ENDE
880:                      ;
890:    03C0 A5 57                LDA  IPNT
900:    03C2 18                   CLC
900:    03C3 69 05                ADC  #5       ; I = I+1
910:    03C5 85 57                STA  IPNT
920:    03C7 90 02                BCC  TESTI
930:    03C9 E6 58                INC  IPNT+1
940:                      ;
950:    03CB CD D9 03 TESTI       CMP  NPNT
960:    03CE D0 99                BNE  ILOOP
970:    03D0 A5 58                LDA  IPNT+1   ; I = N ?
980:    03D2 CD DA 03             CMP  NPNT+1
990:    03D5 D0 92                BNE  ILOOP
1000:                     ;
1010:   03D7 60           ENDE    RTS
1020:                     ;
1030:   03D8              FLAG    *=   *+1
1040:   03D9              NPNT    *=   *+2
]033C-03DB
NO ERRORS
```

A gépi kódú rutin elvégzi ugyanazt a feladatot, amit a P.19-es BASIC program. Hívása előtt az adatok helyes előkészítéséről a felhasználónak kell gondoskodnia, ugyanis a rutinba semmilyen ellenőrzést nem építettünk be. Nem vizsgálja meg azt sem, hogy milyen tömböt (hány indexes) deklaráltunk, és azt sem, hogy a tömböt valóban feltöltöttük-e adatokkal. Mindez a használó dolga. A rutint a SYS 828
utasítással hívhatjuk meg.
A gépi kódú rutin hatékonyságát bizonyítandó elkészítettük az alábbi összehasonlító táblázatot, amely tartalmazza az azonos méretű véletlenszám generátorral feltöltött tömbök rendezéséhez szükséges végrehajtási időt a gépi kódú, ill. a BASIC program esetében.

```
  ***       T46.       ***
   N         BASIC              GEPI RUTIN
-----------------------------------------
  10            1''                0.0''
  50           24''                0.4''
 100        1' 37''                1.5''
 200        6' 33''                6.3''
 500       41'                    38.7''
1000    2h 44'                 2' 33.4''
```

Megfigyelhetjük, hogy a rendezési idő az elemszám növekedtével négyzetesen nő, azaz 2-szer annyi elemű tömb rendezéséhez 4-szer annyi időre van szükség.
Ez a magyarázata annak, hogy amint viszonylag nagy elemszámú tömböt kell rendezni, a BASIC program kivárhatatlan ideig (órákig) dolgozik.
A gépi kódú program végrehajtási sebessége 60-szorosa a BASIC programénak.
Az is elképzelhető, hogy egy-egy feladat megoldása során olyan nagy tömböket kell rendezni, hogy már a gépi kódú rutin sebessége sem elegendő.
Ekkor a buborék algoritmust hatékonyabb rendezési eljárásra kell kicserélni.

A gyakorlás kedvéért módosítsuk most úgy a rutint, hogy az alkalmas legyen egész típusú tömb rendezésére. Mit kell ennek érdekében megváltoztatnunk? Először is figyelembe kell vennünk, hogy a tömb egy elemének helyigénye most csak két byte a korábbi öttel szemben. Másrészt az elemek összehasonlításához nincs szükségünk rendszerrutinra.
Elmaradhat a számok lebegőpontossá alakítása is, hiszen a kétbyte-os egészeket közvetlenül, átalakítás nélkül is összehasonlíthatjuk.
Ezek a módosítások jelentősen javítják a program hatékonyságát, az új rendező sokkal gyorsabb lesz mint az előző, amely valós számokkal dolgozott.
Azon Olvasóink számára, akik kézikönyvként szeretnék használni ezt a könyvet, közöljük a BASIC értelmező aritmetikai függvényeinek és műveleteinek táblázatát.

*** T46/1. ***

Nev	Cim	Mutato a konstansra	Elokeszites	FAC	Funkcio
MEMARG	$BA8C	A/Y	-	-	ARG := Konstansok
FACARG	$BBFC	-	-	+	FAC := ARG
DIV	$BB12	-	A = EXP	+	FAC := ARG / FAC
MEMDIV	$BB0F	A/Y	-	+	FAC := Konst./FAC
MAL10	$BAE2	-	-	+	FAC := FAC * 10
DURCH10	$BAFE	-	-	+	FAC := FAC / 10
PLUS05	$BB49	-	-	+	FAC := FAC + 0.5
MEMFAC	$BBA2	A/Y	-	+	FAC := Konstansok
FACARG	$BC0C	-	-	-	ARG := FAC
FACMEM	$BBD4	X/Y	-	-	Konstansok := FAC
MINUS	$B853	-	A = EXP	+	FAC := ARG - FAC
MEMMIN	$B850	A/Y	-	+	FAC := Konst./FAC
MULT	$BA2B	-	A = EXP	+	FAC := ARG * FAC
MEMMULT	$BA28	A/Y	-	+	FAC := Konst.*FAC
PLUS	$B86A	-	A = EXP	+	FAC := ARG + FAC
MEMPLUS	$B867	A/Y	-	+	FAC := Konst.+FAC
HOCH	$BF7B	-	A = EXP	+	FAC := ARG ^ FAC
HOCHMEM	$BF78	A/Y	-	+	FAC := ARG ^ Konst.
POLY	$E059	A/Y	-	+	FAC := Polinom
POLY2	$E043	A/Y	-	+	FAC := Polinom2
OR	$AFE6	-	-	+	FAC := ARG OR FAC
AND	$AFE9	-	-	+	FAC := ARG AND FAC
NOT	$AED4	-	-	+	FAC := NOT FAC
VERGLCH	$BC5B	A/Y	-	-	FAC osszehasonl. a konst.
ROUND	$BC1B	-	-	+	FAC kerekitese
CHGSGN	$BFB4	-	-	+	FAC := - FAC

A konvertáló rutinokat és az alapfüggvényeket a fenti táblázatból kihagytuk, hiszen ezeket már korábban ismertettük. A táblázat tartalmazza a rutinok paramétereit és a rutin pontos leírását.
Az FAC alatti oszlopban a „+" és „−" jelek arra utalnak, hogy az FAC értéke a rutin végrehajtása közben módosul vagy nem. Ha a rutin a műveletet az FAC és az ARG akkumulátorok tartalmával végzi el, hívás előtt be kell tölteni a kitevőt az FAC-ből az akkumulátorba.
Az AND, az OR, ill. a NOT logikai műveletek előtt az értelmező az operandusokat 16 bites egész számokká alakítja, elvégzi a műveletet, majd az eredményt ismét lebegőpontos alakban tárolja az FAC-ben.
A rutinok végrehajtásához szükséges valós állandókat az értelmező fix tárcímeken tárolja.

*** T46/2. ***

Cim	Konstansok	Decimalis ertek	Jelentes
$AEA8	82 49 0F DA A1	3.14159265	Pi
$B1A5	90 80 00 00 00	-32768	
$B9BC	81 00 00 00 00	1	
$B9C2	7F 5E 56 CB 79	.434255942	
$B9C7	80 13 9B 0B 64	.576584541	
$B9CC	80 76 38 93 16	.961800759	
$B9D1	82 38 AA 3B 20	2.88539007	
$B9D6	80 35 04 F3 34	.707106781	1/SQR(2)
$B9DB	81 35 04 F3 34	1.41421356	SQR(2)
$B9E0	80 80 00 00 00	-.5	
$B9E5	80 31 72 17 F8	.693147181	LOG(2)
$BAF9	84 20 00 00 00	10	
$BDB3	9B 3E BC 1F FD	99999999.9	
$BDB8	9E 6E 6B 27 FD	999999999	
$BDBD	9E 6E 6B 28 00	1E9	
$BFBF	81 38 AA 3B 29	1.44269504	1/LOG(2)
$BFC5	71 34 58 3E 56	2.14987637E-5	
$BFCA	74 16 7E B3 1B	1.4352314E-4	
$BFCF	77 2F EE E3 85	1.34226348E-3	
$BFD4	7A 1D 84 1C 2A	9.614011701E-3	
$BFD9	7C 63 59 58 0A	.0555051269	
$BFDE	7E 75 FD E7 C6	.240226385	
$BFE3	80 31 72 18 10	.693147186	
$BFE8	81 00 00 00 00	1	
$E08D	98 35 44 7A 00	11879546	
$E092	68 28 B1 46 00	3.92767774E-4	
$E2E0	81 49 0F DA A2	1.57079633	Pi/2
$E2E5	83 49 0F DA A2	6.28318531	Pi*2
$E2EA	7F 00 00 00 00	.25	
$E2F0	84 E6 1A 2D 1B	-14.3813907	
$E2F5	86 28 07 FB F8	42.0077971	
$E2FA	87 99 68 89 01	-76.7041703	
$E2FF	87 23 35 DF E1	81.6052237	
$E304	86 A5 5D E7 28	-41.3147021	
$E309	83 49 0F DA A2	6.28318531	Pi*2
$E33F	76 B3 83 BD D3	-6.84793912E-4	
$E344	79 1E F4 A6 F5	4.85094216E-3	
$E349	7B 83 FC B0 10	-.0161117015	
$E34E	7C 0C 1F 67 CA	.034209638	
$E353	7C DE 53 CB C1	-.054279133	
$E358	7D 14 64 70 4C	.0724571965	
$E35D	7D B7 EA 51 7A	-.0898019185	
$E362	7D 63 30 88 7E	.110932413	
$E367	7E 92 44 99 3A	-.142839808	
$E36C	7E 4C CC 91 C7	.19999912	
$E371	7F AA AA AA 13	-.333333316	
$E376	81 00 00 00 00	1	

2.1 A MEGSZAKÍTÁSOK PROGRAMOZÁSA

A gépi kódú programozásnak ezt a területét általában a legjobb programozók is nagy ívben elkerülik.
Az alábbi fejezetekben a megszakítástechnika alapelveinek lefektetésével szeretnénk bebizonyítani, hogy ez az idegenkedés teljes mértékben megalapozatlan.
Miután megvilágítottuk a „megszakítás" (interrupt) fogalom lényegét, az új technika alkalmazási területeit tárgyaljuk.
Az angol *interrupt* kifejezés lefordítva „megszakítás"-t jelent. Mi az, amit megszakítunk?
Egészen egyszerűen: az éppen működő gépi kódú program futását. A megszakítás hardver eredetű, és a program végrehajtásának bármely fázisában előidézhető. „Ki" vagy „mi" képes megszakítani egy gépi kódú program végrehajtását? A kérdés megválaszolásához elengedhetetlen, hogy a processzor hardverfelépítését némileg ismerjük.
A 6502-es, ill. a 6510-es mikroprocesszor egy negyvenpólusú tok, amelyben különleges szerepet töltenek be az

IRQ és NMI

jelű lábak.

A rövidítések eredete:

Interrupt Request = megszakítás kérés

Non Maskable Interrupt = nem maszkolható megszakítás

A lábakon fellépő impulzus a következőket idézi elő:

1. Impulzus az NMI lábon

A processzor befejezi az aktuális műveletet és a programszámláló (felső, alsó byte), és az állapotregiszter pillanatnyi értékét elhelyezi a veremben.
Betölti a $FFFA és $FFFB tárcímek tartalmát és ezeket a programszámláló aktuális értékeként kezeli, azaz végrehajt egy indirekt ugrást, azaz egy JMP ($FFFA) utasítást és az ugrás helyén talált programot futtatni kezdi.
A végrehajtott programról később esik szó.

2. Impulzus az IRQ lábon

Ez esetben is a fentiekhez hasonló folyamat zajlik le. A processzor az aktuális művelet befejezése után reagál a megszakításkérésre.
Azonban most mielőtt ténylegesen végrehajtaná a megszakítást, megvizsgálja az állapotregiszter harmadik bitjének, az interrupt (I) kapcsolónak az értékét.

Ha ez 1, akkor „visszautasítja" a megszakítási kérelmet és folytatja a futó program végrehajtását.
Ha a kapcsoló értéke 0 volt, akkor a processzor a fent leírtak szerint jár el: a programszámláló és az állapotregiszter aktuális értékét tárolja a veremben. Az I kapcsoló értékét 1-re állítja annak érdekében, hogy a „kitérő" program futását újabb megszakítási kérelem ne zavarja. Betölti a $FFFE és $FFFF címek tartalmát (felső, alsó byte), és az ezen a címen talált címet tekinti a programszámláló új értékének.
A megszakító programból hogyan lehet visszatérni?
Erre a célra egy speciális gépi kódú utasítás szolgál:

RTI – Return from Interrupt

Az utasítás a megszakításnál végrehajtott lépések ellentétét váltja ki. Az állapotregiszter, ill. a programszámláló értékét visszatölti a veremből, majd ezen a címen folytatja a program végrehajtását.
A megszakított program természetesen nem figyeli, hogy „őt" saját magát a megszakítás során megváltoztattuk, vagy sem. A processzor csak az állapotregiszter tartalmát menti, a többi regiszter értékével nem foglalkozik. Helyreállításukról a programozónak kell gondoskodnia az RTI utasítás végrehajtása előtt, pl. így:

```
***       P21.       ***

INTERRUPT       PHA   ; Az akkumulator mentese
                TXA
                PHA   ; Az X-regiszter mentese
                TYA
                PHA   ; Az Y-regiszter mentese

                ...   ; A megszakito rutin

                PLA
                TAY   ; Az Y-regiszter visszatoltese
                PLA
                TAX   ; Az X-regiszter visszatoltese
                PLA   ; Az akku visszatoltese
                RTI   ; Visszateres a megszakito programbol
```

A megszakító rutin szerkezetileg hasonló a szubrutinokhoz. A alapvető különbség az, hogy a szubrutint a főprogram egy meghatározott helyéről hívjuk, míg a megszakítást hardveresen váltjuk ki, és ez a főprogram végrehajtása közben bármikor történhet. A másik lényeges különbség, hogy a megszakítás során nemcsak a visszatérési címet, hanem az állapotregiszter tartalmát is megőrizzük, ami elengedhetetlen ahhoz, hogy a megszakított program futása hiba nélkül folytatódhasson.
Feltehetően az eddig leírtak megérlelték az Olvasóban is a következő, legfényegesebb kérdést:
Mi váltja ki a megszakítást?
A Commodore 64-esen a megszakításnak többféle oka lehet. Nézzünk egy példát az IRQ típusú megszakításokra: ezt kiválthatja a

VIC 6569 videovezérlő

és a

CIA 6526 I/O elemei (a $DC00-s címen).

Nem maszkolható megszakítást (NMI) válthat ki pl. a

CIA 6526 (a $DD00-s címen)

vagy pl. a

RESTORE billentyű

A saját megszakítások programozásához elengedhetetlenül fontos, hogy ismerjük az I/O elemek tulajdonságait, lehetőségeit. Ezekre olyan mélységig térünk most ki, amennyire a későbbi programokhoz szükséges. Bővebb leírást talál az Olvasó a „A Commodore 64-es belső felépítése" c. könyvben.

2.2 A CIA 6526-OS PROCESSZOR

A CIA (Complex Interface Adapter) 6526-os a 65XX-es processzorcsalád egy I/O eleme, amely egy soros 8 bites regisztert, két kaszkádolható 16 bites számlálót (timert), egy valós idejű órát és egy diverse vezérlővonalat tartalmaz.
A CIA további 16 regiszterét a processzor átmeneti tárolóként használhatja. A Commodore-ban két ilyen I/O elem található, az egyik a $DC00-s címtől a $DC0F címig, a másik pedig a $DD00-tól a $DD0F-ig terjedő tárterületet foglalja el.
A következő oldalakon ismertetjük a 16 vezérlőregiszter funkcióját.

A vezérlőregiszterek leírása

0. regiszter
A port
Hozzáférés: READ/WRITE
A regiszter tartalma az A (I/O port állapotát tükrözi.

1. regiszter
B pont
Hozzáférés: READ/WRITE
A regiszter tartalma a B (I/O) port állapotát tükrözi.

2. regiszter
A adatirányregiszter
Hozzáférés: READ/WRITE
Ezzel a regiszterrel az A port mind a 8 vonalát bevitelre, ill. kihozatalra kapcsolhatjuk. A kapcsoláshoz a regiszter megfelelő bitjét 0-ra (bevitel) vagy 1-re (kihozatal) kell állítani.

3. regiszter
B adatirányregiszter
Hozzáférés: READ/WRITE
A regiszter feladata azonos a 2. regiszter feladatával a B portra vonatkoztatva.

4. regiszter
A számláló alsó byte
Hozzáférés: READ
Olvasásnál a regiszter tartalma az A számláló pillanatnyi értékének alsó byte-ját adja vissza.
Hozzáférés: WRITE
Írásnál megadhatjuk annak az értéknek az alsó byteját, amelyről a számláló nullára visszaszámlál.

5. regiszter
A számlóló felső byte
Hozzáférés: READ
Visszakapjuk az A számláló pillanatnyi értékének alsó byte-ját
Hozzáférés: WRITE

Író utasítással megadhatjuk annak az értéknek a felső byte-ját, amelyről az A számláló nullára visszaszámlál

6. regiszter

B számláló alsó byte
Ua. mint a 4. regiszter a B számlálóra vonatkoztatva.

7. regiszter

B számláló felső byte
Ua. mint az 5. regiszter a B számlálóra vonatkoztatva

8. regiszter

A valós idejű óra (time of day) tizedmásodpercei
Hozzáférés: READ
Olvasásnál a 0-tól 3-ig terjedő bitek tartalma visszaadja a valós idejű óra pillanatnyi értékét tizedmásodpercekben, BCD kódban. A 4-től 7-ig terjedő bitek értéke mindig nulla.
Hozzáférés: WRITE
Írásnál a B vezérlőregiszter (15.) előzetes beállításával választhatunk: megadhatjuk az óra aktuális értékét tizedmásodpercekben, vagy a riasztás időpontját.
A 4-től 7-ig terjedő biteket nullára kell állítani, a tizedmásodperceket pedig BCD formátumban kell megadni.

9. regiszter

A valós idejű óra másodpercei
Hozzáférés: READ
Olvasásnál a regiszter tartalma visszaadja az óra pillanatnyi értékét másodpercekben, a 0-tól 3-ig terjedő bitek értéke az egyeneseket, a 4-től 7-ig terjedő bitek értéke pedig a tízeseket jelenti
Hozzáférés: WRITE
Ua. mint a 8. regiszternél. A másodpercek formátuma is BCD kód.

10.regiszter

A valós idejű óra percei.
Ua. mint a 9. regiszternél a percekre vonatkoztatva.

11. regiszter

A valós idejű óra órái
Hozzáférés: READ
Olvasásnál visszakapjuk az idő aktuális értékét órában. A 0-tól 3-ig terjedő bitek értéke most is az egyeseket jelenti. Mivel az óra legnagyobb értéke 12 lehet, a tízesek jelölésére elég egyetlen bit, nevezetesen a 4. bit.
A 7. bit az amerikai időnek megfelelően a délelőtt (AM, 7. bit = 0), ill. délután (PM, 7. bit = 1) jelölésére szolgál.
Hozzáférés: WRITE
Ua. mint a fenti regisztereknél, de a biteket az olvasási hozzáférésnél leírtak szerint kell kezelni.

12. regiszter

Soros eltolási regiszter
Kiírásnál a soros buszra kerülő, olvasásnál a soros buszról érkező adatokat tárolja bitenként.

13. regiszter

Megszakítást vezérlő regiszter
(Interrupt Control Register)
Hozzáférés: READ
0. bit : az A számláló lefutása
1. bit : a B számláló lefutása
2. bit : az aktuális idő és a riasztási idő értéke azonos
3. bit : az eltolási regiszter megtelt (beolvasásnál) vagy üres (kiírásnál)
4. bit : a FLAG lábon impulzus érkezett
5–6. bit : mindig nulla
7. bit : értéke 1, ha a megszakítást vezérlő és a megszakítást maszkoló regiszterek 0-tól 4-ig terjedő bitjei közül legalább egy értéke 1.

Figyelem! Olvasásnál a regiszter tartalma törlődik!
Hozzáférés: WRITE (Megszakítási maszk)
A 0-tól 4-ig terjedő bitek jelentése azonos. Ha ráadásul a 7. bit értéke 1, a megszakítást tetszőleges művelet számára kihasználhatóvá tehetjük. Ha a 7. bit nulla, a megszakítás nem engedélyezett.

14. regiszter

A vezérlő regiszter
Hozzáférés: READ/WRITE
0. bit : 0 = A számláló STOP = 1 A számláló START
1. bit : 1 = az A számláló lefutását jelzi a PB6
2. bit : 0 = az A számláló minden lefutása előállít egy magas impulzust a PB6 lábon.
1 = az A számláló minden lefutása megfordítja a PB6 állapotát.
3. bit : 1 = az A számláló visszaszámlál a kezdőértéktől nulláig, majd leáll (one shot).
0 = az A számláló minden lefutás után automatikusan újraindul.
4. bit : 1 = az A számláló új értékének betöltése feltétel nélkül
5. bit : 0 = az A számláló számlálja a rendszerütemet, 1 = számlálja a CNT ütemet.
6. bit : 0 = a soros port bemenet, 1 = a soros port kimenet.
7. bit : 0 = a valós óra 60 Hz-cel üzemel
1 = a valós óra 50 Hz-cel üzemel

15. regiszter

B vezérlő regiszter
Hozzáférés: READ/WRITE

0–4. bitek: Az A vezérlő regiszter bitjeivel azonos jelentésűek a B számlálóra és a PB7-re vonatkoztatva.

5–6. bitek: Ezek a bitek határozzák meg a B számláló triggerforrását:
00 = a számláló a rendszerütemet számlálja, 01 = a számláló a CNT ütemet számlálja
10 = a B számláló az A számláló lefutásait számlálja
11 = a B számláló az A lefutásait számlálja, ha CNT = 1

7. bit: 0 = valós idő beállítása.
1 = riasztási idő beállítása.

2.3 A RENDSZERMEGSZAKÍTÁSOK FELHASZNÁLÁSA

A legegyszerűbb megszakító rutin nem tesz egyebet, mint „csatlakozik" a rendszermegszakításhoz.
Mi váltja ki a rendszermegszakítást és milyen célból?
A rendszermegszakítást a CIA 1 számlálója vezérli.
A számláló értéke minden gépi ütemben, ami kb. egy mikrosekundum hosszú, eggyel csökken. Ha az érték nullára csökkent, a számláló lead egy impulzust a processzor IRQ bemenetére.
Az aktuális program megmarad, és a processzor végrehajt egy elágazást a $EA31-es címen kezdődő megszakítási rutinba. A számláló két nyolcbites regiszterből áll, így maximum 2^{16} mikrosekundumig, azaz 65 millisekundumig képes számlálni. A rendszermegszakítások folyamatosan minden 60-ad másodpercben (azaz minden 16 millisekundumban) követik egymást.
Milyen feladatokat lát el a megszakítási rutin?
Ellenőrzi, hogy a STOP billentyű le van-e nyomva. Ha igen, a nulláslapon beállít egy kapcsolót 1-re. A kapcsolót a rendszer minden BASIC utasítás végrehajtása előtt megvizsgálja.
Ha értéke 1, a program futása megszakad.
A rutin a STOP billentyű lekérdezéséhez a belső órát (TI) megnöveli, amely minden hatvanad másodpercben változik.
A rutin következő feladata a kurzor vizsgálata. Ha a gép parancsmódban bevitelre várakozik, a kurzor villog. A kurzor alatti karaktert a rendszer minden huszadik megszakítása közben invertálja. A kurzor tehát 60/20 = 3 alkalommal villog minden másodpercben. A rutin további feladata a kazettás egység figyelése. Ha a kazettás egység éppen nincs programvezérlés alatt (LOAD, SAVE), akkor attól függően, hogy egy bizonyos billentyű az egységen le van nyomva vagy nincs, a motort ki-, ill. bekapcsolja.
A megszakítási rutin utolsó és egyben legfontosabb feladata a billentyűzet lekérdezése. Ha egy billentyű le van nyomva, a rendszer átveszi a billentyű kódját, és elhelyezi a billentyűzet pufferében. A puffer tíz karakter hosszúságú. Így lehetséges az, hogy előre leüthetünk olyan billentyűket, amelyek a képernyőn csak akkor jelennek meg, amikor a program „fogadja" azokat. A pufferben elhelyezett karakterek számát a rendszer tárolja. Miután minden feladatot elvégzett, a rutinból visszatérünk és a megszakított program végrehajtása folytatódik.
Amint már említettük, a processzor a $FFFE és $FFFF tárcímeiről tölti be a megszakítási rutin címét. Ezek a címek a ROM területén találhatók. Hogyan lehet megváltoztatni az értéküket? Mi történik a megszakítás után?
Az a cím, ahová a megszakítási vektor mutat, a $FF48.

```
***      P22.      ***
*****************************   IRQ belepesi pont
FF48   48          PHA
FF49   8A          TXA
FF4A   48          PHA          A regiszter mentese
FF4B   98          TYA
FF4C   48          PHA
```

```
FF4D    BA          TSX
FF4E    BD 04 01    LDA $0104,X     A BREAK-kapcsolo beolvasasa a verembol
FF51    29 10       AND #$10        es vizsgalata
FF53    F0 03       BEQ $FF58       Erteke = 0 ?
FF55    6C 16 03    JMP ($0316)     BREAK-rutin
FF58    6C 14 03    JMP ($0314)     Megszakito rutin
```

Először a regiszterek tartalma a verembe kerül.
Ekkor a megszakításnál automatikusan „mentett" állatregiszter tartalmát a rutin visszatölti, és leválasztja a 4. bitet. Ez a BREAK kapcsoló, amelynek értékét a BRK utasítás állítja 1-re.
A BRK utasítás szoftveresen szimulál egy megszakítást.
A BREAK kapcsoló alapján a rendszer meg tudja különböztetni ezt a megszakítást a hardveres megszakítástól. Értékétől függően két indirekt ugrás következik. Ha a kapcsoló értéke 1, akkor az ugrási címet a $316/$317 cím, ha értéke 0, az ugrási címet a $314/$315 cím tartalma szabja meg.
A $314/$315 vektor a tényleges megszakítási rutinra, azaz a $EA31-es címre mutat.
Ha a megszakítási rutint egy további, saját feladattal kívánjuk felruházni, a következőket kell tenni:
Megváltoztatjuk a vektor tartalmát úgy, hogy az a saját rutinunkra mutasson. Ebből a programrészből viszont a valódi megszakítási rutinra térünk vissza, hogy az is elvégezhesse a saját feladatait. Ezzel az eljárással egy főprogramtól független munkával (JOB-bal) ruháztuk fel a rendszert, amit minden másodpercben 60-szor végre kell hajtani. Természetesen ez a rutin nem tarthat tovább, mint egyhatvanad másodperc, hiszen egyébként nem marad idő a főprogram végrehajtására. Egy hosszú megszakítási rutin jelentősen megnövelheti a főprogram futási idejét.
Mi az, amit érdemes minden hatvanad másodpercben elvégeztetni a géppel? Ezen a téren szinte semmi sem szab határt az Olvasó fantáziájának.
Például villogtathatjuk a képernyőt, vagy azon egy feliratot éppúgy, mint ahogyan a kurzort a rendszer villogtatja. Természetesen ha nem akarjuk, hogy a villogás túl szapora legyen, be kell építenünk a programba egy változót, amely figyeli, hogy pl. csak minden 30. végrehajtásnál kerüljön sor a színek kikapcsolására.

```
***       P23.       ***
PROFI-ASS 64 V2.0

100:    C000                        .OPT P,00
110:                        ;
120:                        ; Hatter/keret villogtatasa
130:                        ;
140:    C000                        *=    $C000
150:                        ;
160:    D020                KERET   =     $D020    ; Kepernyokeret
170:    D021                HATTER  =     $D021    ; Kepernyohatter
180:    EA31                IRQROUT =     $EA31
190:    0314                IRQVEC  =     $314
200:                        ;
210:    001E                SZAM    =     30       ; Minden fel mp.-ben
220:                        ;
230:    C000 78             INIT    SEI            ; A megszakitas letiltasa
240:    C001 A9 0D                  LDA   #<VILLOG
250:    C003 A0 C0                  LDY   #>VILLOG
260:    C005 8D 14 03               STA   IRQVEC   ; IRQ-vektor a villogtato rutinra
```

```
270:    C008 8C 15 03           STY   IROVEC+1
280:    C00B 58                 CLI
290:    C00C 60                 RTS
300:                    ;
310:    C00D CE 26 C0 VILLOG    DEC   COUNT    ; Szamlalo csokkentese
320:    C010 D0 11              BNE   KESZ
330:    C012 A9 1E              LDA   #SZAM
340:    C014 8D 26 C0           STA   COUNT    ; A szamlalo beallitasa
350:                    ; A szin kicserelese
360:    C017 AE 21 D0           LDX   HATTER
370:    C01A AD 20 D0           LDA   KERET
380:    C01D 8D 21 D0           STA   HATTER
390:    C020 8E 20 D0           STX   KERET
400:                    ;
410:    C023 4C 31 EA KESZ      JMP   IROROUT
420:                    ;
430:    C026 1E       COUNT     .BYT SZAM      ; Szamlalo
 C000-C027
NO ERRORS
```

Elemezzük a programot. Az INIT rutin gondoskodik a kezdőértékek beállításáról (inicializálásról) és beállítja a megszakítási rutin kezdőcímét úgy, hogy az a villogtató rutinra mutasson. A SEI utasítással letiltjuk a megszakítási vektor változtatása közben esetlegesen fellépő megszakításokat. Gondoljuk meg ugyanis, milyen hibákhoz vezethetne, ha abban a pillanatban lépne fel egy megszakítás, amikor a vektor alsó byte-ját már megváltoztattuk, de a felső byte-ot még nem. A végrehajtás egy előre meghatározhatatlan helyen folytatódna, és ez beláthatatlan következményekhez vezetne. Az alsó és felső byte megváltoztatása után a CLI utasítással újra engedélyezzük a megszakításokat. Ettől kezdve azonban az új megszakítási rutin lesz aktív.

A következő megszakítás az alábbiak szerint zajlik le: A COUNT változó értéke 1-gyel csökken. Ha az így kapott érték nem 0, akkor a program elágazik a KÉSZ címkére és egy normális megszakítást hajt végre. Ha a számláló értéke 0 volt, a program ismét 30-at tölt bele. Ekkor felcseréli a keret és a háttér színét, ami a képernyőn a villogás hatását kelti.

A saját gépi kódú rutint a SYS 12*4096 utasítással hívhatjuk. A hívás pillanatától kezdve a képernyő minden másodpercben kétszer felvillan. A villogás, azaz a megszakítási rutin végrehajtása teljesen független az éppen futó BASIC vagy gépi kódú program végrehajtásától. A rendszer ezt a háttértevékenységet mindaddig végzi, amíg a megszakítási vektor eredeti értékét vissza nem állítjuk, vagy a RUN/STOP – RESTORE billentyűket le nem ütjük.

A villogás sebességét a SZÁML változó értékének megváltoztatásával szabályozhatjuk, ez adja meg ugyanis, hogy hány hatvanad másodperc eltelte után cserélje fel a rutin a háttér és a keret színét.

Második példánkban megváltoztatjuk a kurzor megjelenési formáját. Azt akarjuk elérni, hogy a kurzor ne villogjon, hanem helyette a kurzor alatti inverz karakter jelenjen meg. A megoldáshoz nem elég az új rutint egyszerűen beilleszteni a megszakítási rutin elé, hanem teljesen ki kell cserélnünk a kurzor villogtatását végző rutint.

```
***       P24.       ***
*****************************  Megszakitorutin
EA31   20 EA FF    JSR $FFEA   STOP-billentyu, az ido noveles
EA34   A5 CC       LDA $CC     A kurzor vill. kapcsoloja
EA36   D0 29       BNE $EA61   Ha nem villog, akkor tovabb
EA38   C6 CD       DEC $CD     A szamlalo csokkentese
EA3A   D0 25       BNE $EA61   Nem 0, akkor tovabb
EA3C   A9 14       LDA #$14    A szamlalo = 20
EA3E   85 CD       STA $CD     Tarolas
```

```
EA40   A4 D3      LDY $D3          A kurzor oszlopa
EA42   46 CF      LSR $CF          A kapcsolo = 0, C = 1
EA44   AE 87 02   LDX $0287        A kurzor alatti szin
EA47   B1 D1      LDA ($D1),Y      Karakterkod beallitasa
EA49   B0 11      BCS $EA5C        Ha a kapcsolo = 1, akkor tovabb
EA4B   E6 CF      INC $CF          A kapcsolo be
EA4D   85 CE      STA $CE          A karakter tarolasa
EA4F   20 24 EA   JSR $EA24        A szin-RAM mutato meghatarozasa
EA52   B1 F3      LDA ($F3),Y      A szinkod betoltese,
EA54   8D 87 02   STA $0287        es tarolasa
EA57   AE 86 02   LDX $0286        A kurzor alatti szin
EA5A   A5 CE      LDA $CE          A kurzor alatti karakter
EA5C   49 80      EOR #$80         Az RVS-bit megforditasa
EA5E   20 1C EA   JSR $EA1C        A kurzor karakter es a szin beallitasa
```

A kurzor villogtatásának folyamata a következő:
Először megvizsgáljuk, hogy a kurzor aktív-e. Ha nem, akkor a következő programrészt átugorjuk.
Ha a kurzor aktiv, akkor a számlálót 1-gyel csökkentjük. Ha a kapott érték nem 0, akkor a következő programrészt ismét átugorjuk. Egyébként megvizsgáljuk, hogy a kurzor éppen inverz fázisban volt-e. Ettől függően a pillanatnyi, vagy a korábban tárolt értéket invertáljuk és megjelenítjük. Az utóbbi eljárást megismételjük a színramban is, a karakter és a kurzor pillanatnyi színével.
Az alábbi program lefuttatása után a képernyőn nem villogó, hanem álló kurzort láthatunk.

```
***       P25.       ***
PROFI-ASS 64 V2.0

100:   C000                        .OPT P,00
110:                          ;
120:                          ; A kurzor modositasa
130:                          ;
140:   FFEA           STOP    =    $FFEA     ; A STOP billentyu lekerdezese
150:   00CC           CURSFLAG =   $CC       ; A lathato kurzor kapcsoloja
160:   00CF           INVERS  =    $CF       ; Az inverz karakter kapcsoloja
170:   0287           CURSCOL =    $287      ; A kurzor alatti szin
180:   00CE           CURSCHAL =   $CE       ; A kurzor alatti karakter
190:   00D1           CHAR    =    $D1       ; Mutato a videoramban
200:   00F3           COLOR   =    $F3       ; Mutato a szinramban
210:   EA24           SETCOL  =    $EA24     ; A szinram mutatoja
220:   00D3           SPALTE  =    $D3       ; A kurzor oszlopa
230:   0286           COLSTR  =    $286      ; A kurzor szine
240:   0314           IRQVEC  =    $314      ; IRQ-vektor
250:   EA61           CONTIRQ =    $EA61     ; Az IRQ vegrehajtasa
260:                          ;
270:   C000 78        INIT     SEI            ; A megszakitas letiltasa
280:   C001 A9 0D              LDA  #<NEWCURS
290:   C003 A0 C0              LDY  #>NEWCURS
300:   C005 8D 14 03           STA  IRQVEC    ; IRQ-vektor az uj rutinra
310:   C008 8C 15 03           STY  IRQVEC+1
320:   C00B 58                 CLI
330:   C00C 60                 RTS
340:                          ;
350:   C00D 20 EA FF NEWCURS   JSR  STOP       ; A STOP billentyu vizsgalata
360:   C010 A5 CC              LDA  CURSFLAG  ; A kurzor lathato ?
370:   C012 D0 1D              BNE  NOCURSOR  ; Nem
380:   C014 A4 D3              LDY  SPALTE    ; A kurzor oszlopa
a90:   C016 A5 CF              LDA  INVERS    ; A karaktert invertaltuk ?
400:   C018 D0 17              BNE  NOCURSOR  ; Igen
410:   C01A E6 CF              INC  INVERS    ; Kapcsolo az invertalasra
420:   C01C 20 24 EA           JSR  SETCOL    ; Mutato a szinramban
430:   C01F B1 D1              LDA  (CHAR),Y  ; A kurzorpozicio karakterenek
430:   C021 85 CE              STA  CURSCHAR  ; tarolasa
440:   C023 49 80              EOR  #$80      ; Az RVS bit megforditasa es a
450:   C025 91 D1              STA  (CHAR),Y  ; szin tarolasa a videoramban
460:   C027 B1 F3              LDA  (COLOR),Y
```

```
460:   C029 8D 87 02          STA  CURSCOL
470:   C02C AD 86 02          LDA  COLSTR     ; A kurzor szinenek
480:   C02F 91 F3             STA  (COLOR),Y  ; beallitasa
490:   C031 4C 61 EA NOCURSOR JMP  CONTIRQ    ; Az IRQ vegrehajtasa
]C000-C034
NO ERRORS
```

Ha a rutint a SYS 12*4096 utasítással aktivizáljuk, a kurzor helyén egy inverz karaktert látunk.
A kész rutint alakíthatjuk úgy, hogy az saját elképzelésünk szerint működjön; pl. a karakter a kurzor színe helyett fehér színű legyen. Megoldhatjuk pl. azt is, hogy a kurzor karakterét ne az invertálás, hanem egy aláhúzás, vagy egy másik szín különböztesse meg a többi karaktertől.
Persze ez a rutin igen egyszerű alkalmazása a megszakítási rutinnak, amelyet azzal a szándékkal ismertettünk, hogy felkeltse az Olvasó érdeklődését, és további ötletek megvalósítására sarkallja.
Röviden még egy felhasználási lehetőséget említünk; a STOP billentyű letiltásának módját.
Emlékezzünk viszont arra, hogy a STOP billentyűt a megszakítási rutin közvetlenül hívása után megvizsgálja – ez a rutin által végrehajtandó első feladat.
Ha a beugrási címet úgy módosítjuk, hogy az a lekérdezést követő első utasításra mutasson, akkor az éppen futó program futását nem lehet a STOP billentyűvel megállítani. A megoldás nagyon egyszerű:

```
POKE 788, PEEK(788)+3
```

A megszakítási vektor címének alsó byte-ját 3-mal megnöveltük, így a megszakítás során a STOP billentyű lekérdezése elmarad.
A módszer hátránya, hogy ilyenkor a belső óra, azaz a TI és TI$ változók értéke nem módosul.
A továbbszámlálást ugyanis szintén a programrész végezné el.
A rendszermegszakítások egy további hasznos alkalmazása az egyes billentyűk programozása. Egy-egy billentyűhöz tetszőleges feladatot rendelhetünk, például egy funkcióbillentyű kiválthat egy hardcopy-t, azaz a képernyő tartalmának kinyomtatását.
A billentyűket nagyon egyszerűen felruházhatjuk speciális feladatokkal. Mindössze annyit kell tennünk, hogy a megszakítási rutinban lekérdezzük a billentyű állapotát, és ha az le van nyomva, végrehajtjuk a kérdéses műveletet.
A lehetőségek száma szinte végtelen. Gyakran van szükségünk például arra, hogy az egyes képernyőoldalak között lapozgassunk.
Írjuk meg azt a programot, amely lehetővé teszi, hogy egy billentyű lenyomásával átkapcsoljunk az aktuálisról egy másik képernyőlapra.

```
***      P26.      ***
PROFI-ASS 64 V2.0

100:   033C                  .OPT P1,00
110:                   ;
120:                   ; A kepernyolap bekapcsolasa
130:                   ;
140:   0003            PNT1      =   3
150:   0005            PNT2      =   5
160:   DD00            VIDEOMAP  =   $DD00     ; 16K video-terulet
170:   0288            VIDEOPGE  =   648
180:   0314            IROVEC    =   $314
190:   EA31            IRQALT    =   $EA31
200:   D000            CHARGEN   =   $D000     ; A karaktergenerator
```

```
D800            COLOR   =   $D800     ; A szinram
C000            COLOR2  =   $C000     ; A szinram cime
0001            PORT    =   1         ; A processzor port
028D            CTRL    =   653       ; CONTROL kapcsolo
00C5            KEY     =   $C5       ; Az utolso billentyu
0004            F1      =   4         ; Az F1 billentyu matrixszama
                ;
033C                    *=  828
                ;
033C 78         INIT    SEI
033D 20 94 03           JSR SETCHAR   ; A karaktergenerator masolasa
0340 A9 4C              LDA #< TEST
0342 A0 03              LDY #> TEST
0344 8D 14 03           STA IRQVEC    ; Mutato az uj rutinra
0347 8C 15 03           STY IRQVEC+1
034A 58                 CLI
034B 60                 RTS
                ;
034C AD 8D 02   TEST    LDA CTRL      ; A CONTROL le van nyomva ?
034F 29 04              AND #%100
0351 F0 09              BEQ NOSWITCH  ; Nem
0353 A5 C5              LDA KEY       ; F1 le van nyomva ?
0355 C9 04              CMP #F1
0357 D0 03              BNE NOSWITCH  ; Nem
0359 20 5F 03           JSR SWITCH    ; Az oldal felcserelese
035C 4C 31 EA NOSWITCH  JMP IRQALT
                ;
035F A0 00      SWITCH  LDY #0
0361 84 03              STY PNT1
0363 84 05              STY PNT2
0365 A9 D8              LDA #>COLOR   ; Mutato a szinramban
0367 85 04              STA PNT1+1
0369 A9 C0              LDA #>COLOR2  ; A szin tarcime
036B 85 06              STA PNT2+1
036D A2 04              LDX #4        ; A lapok szama
036F B1 03      SWAP    LDA (PNT1),Y
0371 48                 PHA
0372 B1 05              LDA (PNT2),Y
0374 91 03              STA (PNT1),Y  ; A szin tarcimenek csereje
0376 68                 PLA
0377 91 05              STA (PNT2),Y
0379 C8                 INY
037A D0 F3              BNE SWAP
037C E6 04              INC PNT1+1
037E CA                 DEX           ; A kovetkezo oldal
037F D0 EC              BNE SWAP
0381 AD 00 DD           LDA VIDEOMAP
0384 49 03              EOR #%11      ; A VIC hozzaferesi cime
0386 8D 00 DD           STA VIDEOMAP
0389 AD 88 02           LDA VIDEOPGE
038C 49 C0              EOR #$C0      ; Kepernyolap
038E 8D 88 02           STA VIDEOPGE
0391 60                 RTS
                ;
0392 A0 00      SETCHAR LDY #0
0394 84 03              STY PNT1
0396 A9 D0              LDA #> CHARGEN
0398 85 04              STA PNT1+1
039A A2 10              LDX #$10
039C A9 33      LOOP    LDA #$33
039E 85 01              STA PORT      ; Karaktergenerator bekapcsolasa
03A0 B1 03              LDA (PNT1),Y
03A2 48                 PHA
03A3 A9 30              LDA #$30
03A5 85 01              STA PORT      ; A RAM bekapcsolasa
03A7 68                 PLA
03A8 91 03              STA (PNT1),Y
03AA C8                 INY
03AB D0 EF              BNE LOOP
03AD E6 04              INC PNT1+1    ; A kovetkezo oldal
03AF
```

```
880:   03B1 CA          DEX
890:   03B2 D0 EA       BNE  LOOP
900:   03B4 A9 37       LDA  #$37       ; Alapallapot
910:   03B6 85 01       STA  PORT
920:   03B8 60          RTS
.033C-03B9
NO ERRORS
```

A programban két képernyőoldalt váltogatunk. Az első lap a megszokott helyén a $0400-as, a másik pedig a $C400-as címen kezdődik.
Az is megoldható, hogy a második lapon egy sprite-ot aktivizáljunk. A sprite-ok mutatóit a $C7F8-as címtől kezdve kell tárolni, és a $C000-s báziscímre vonatkoztatni.
A sprite-okat elhelyezhetjük pl. a $C800-tól a $CFFF-ig terjedő tárterületen, amely 32 különböző sprite-minta tárolására elegendő (32-től 63-ig). Mivel a videovezérlő a színramot mindig azonos kezdőcímen, a $D800-as címen keresi, tároljuk a színeket az éppen nem látható lapon, a $C000-tól $C3FF-ig terjedő tárterületen. Ügyelnünk kell arra is, hogy a felső 16 K területen ($C000–$FFFF) a VIC nem tudja elérni a karaktergenerátort. Ezért a karaktergenerátort a program indításakor átmásoljuk a ROM-ból az ugyanazon a tárterületen levő RAM-ba.
A megszakítási rutin a CONTROL billentyűt a kapcsoló 2. bitjének vizsgálatával teszteli. Ha a bit értéke 1, a billentyű le van nyomva. Ha ezen kívül az F1 billentyűt is lenyomva találja, meghívja azt a rutint, amely kicseréli a színtárat és átállítja a mindenkori (látható) képernyőlap mutatóját. Végül elágazik a valódi megszakítási rutinba. Ha a programot lefordítjuk (assembláljuk), és SYS 828 utasítással elindítjuk, a CONTROL és az F1 billentyű együttes lenyomásával képernyőlapot válthatunk. A legelső aktivizáláskor törölni kell a képernyő tartalmát, nehogy határozatlan értékek maradjanak a videotárban. A billentyűk ismételt lenyomásával mindig az eredeti képernyőre ugorhatunk vissza. A kurzor a képernyőváltás közben a helyén marad.
További alkalmazási feladat az óra mindenkori értékének kijelzése. A megszakítási rutinba építve ezt a funkciót, az óra az éppen futó program végrehajtásától függetlenül mindig látható a képernyőn. Hasonló megoldással már találkozhatott az Olvasó a „Tippek és trükkök a Commodore 64-eshez" c. könyvben.

Nagyon sok érdekes alkalmazást kínálnak a megszakítások felhasználásában a sprite-ok. Minden megszakítás közben mozgathatunk a képernyőn egy vagy több sprite-ot. Hasonló lehetőségek adódnak a hangvezérlés területén. A megszakításokba beépítve különböző hanghatásokat idézhetünk elő, dallamokat szólaltathatunk meg a program futása közben.
A lehetőségek száma valóban korlátlan. Szaporítsuk az eddigi példák sorát még két rutinnal.
Ha az alapgéphez a user porton keresztül egy külső készüléket csatlakoztatunk, hasznos lehet a következő program, amelyet ismét a megszakítási rutinhoz fűztünk. A program kiírja a képernyőre a user port egyes bitjeinek aktuális értékét. Az első képernyősorban kiírtuk az adatirányregiszter tartalmát.
A képernyőről minden pillanatban leolvashatjuk, hogy a port egyes vonalai bemenetre (0), vagy kimenetre (1) vannak kapcsolva. A következő sor tartalma a user port vonalainak állapotáról tájékoztat; a 0 az alacsony, az 1 a magas jelszintet jelzi. Mindkét sor tartalmát kiírjuk hexadecimális alakban is.

```
***      P27.      ***
PROFI-ASS 64 V2.0

100:   033C                        .OPT P1,00
110:                               ;
120:                               ; Az USER-PORT kijelzese
130:                               ;
140:   DD00          CIA2         =    $DD00
150:   DD01          USERPORT     =    CIA2+1
160:   DD03          FELSOSOR     =    CIA2+3
170:                               ;
180:   0288          VIDEOPGE     =    648
190:   D800          COLORRAM     =    $D800
200:                               ;
210:   0007          SZIN         =    7          ; Sarga
220:                               ;
230:   0314          IRQVEC       =    $314
240:   EA31          IRQALT       =    $EA31
250:   00FB          PNT          =    $FB
260:                               ;
270:   033C                        *=   828
280:   033C 78        INIT         SEI
290:   033D AD 14 03               LDA  IRQVEC
300:   0340 49 7E                  EOR  #< IRQALT ^ KIIRAS
310:   0342 8D 14 03               STA  IRQVEC
320:   0345 AD 15 03               LDA  IRQVEC+1
330:   0348 49 E9                  EOR  #> IRQALT ^ KIIRAS
340:   034A 8D 15 03               STA  IRQVEC+1
350:   034D 58                     CLI
360:   034E 60                     RTS
370:                               ;
380:   034F A5 FB     KIIRAS       LDA  PNT
390:   0351 48                     PHA             ; A mutato tarolasa
400:   0352 A5 FC                  LDA  PNT+1
410:   0354 48                     PHA
420:   0355 A9 1C                  LDA  #28
430:   0357 85 FB                  STA  PNT        ; Mutato a videoramban
440:   0359 AD 88 02               LDA  VIDEOPGE
450:   035C 85 FC                  STA  PNT+1
460:   035E AD 03 DD               LDA  FELSOSOR
470:   0361 A0 00                  LDY  #0         ; A felso sor
480:   0363 20 77 03               JSR  DISPLAY    ; kijelzese
490:   0366 AD 01 DD               LDA  USERPORT
500:   0369 A0 28                  LDY  #40        ; Az utolso sor
510:   036B 20 77 03               JSR  DISPLAY    ; kijelzese ( USER-PORT )
520:   036E 68                     PLA
530:   036F 85 FC                  STA  PNT+1      ; A mutato visszatoltese
540:   0371 68                     PLA
550:   0372 85 FB                  STA  PNT
560:   0374 4C 31 EA               JMP  IRQALT     ; Vissza az eredeti IRQ-hoz
570:                               ;
580:   0377 48        DISPLAY      PHA             ; A hexadecimalis kijelzes
590:   0378 A2 08                  LDX  #8
600:   037A 0A        LOOP         ASL             ; A felso bit a CARRY-be
610:   037B 48                     PHA
620:   037C A9 30                  LDA  #"0"       ; A 0 kiirasa,
630:   037E 90 02                  BCC  NULL
640:   0380 A9 31                  LDA  #"1"       ; ha C=1, akkor 1 kiirasa
650:   0382 91 FB     NULL         STA  (PNT),Y
660:   0384 A9 07                  LDA  #SZIN      ; es a szin beallitasa
670:   0386 99 1C D8               STA  COLORRAM+28,Y
680:   0389 C8                     INY
690:   038A 68                     PLA
700:   038B CA                     DEX             ; Kovetkezo bit
710:   038C D0 EC                  BNE  LOOP
720:                               ;
730:                               ; Hexaszam
740:                               ;
750:   038E C8                     INY
```

```
 760:   038F 68                    PLA
 770:   0390 48                    PHA
 780:   0391 4A                    LSR
 780:   0392 4A                    LSR             ; A felso fel-byte eldobasa
 780:   0393 4A                    LSR
 780:   0394 4A                    LSR
 790:   0395 20 99 03              JSR  ASCII      ; A felso fel-byte es
 800:   0398 68                    PLA             ; az also fel-byte
 810:   0399 29 0F      ASCII      AND  #%1111
 820:   039B C9 A0                 CMP  #10
 830:   039D 90 02                 BCC  KISEBB
 840:   039F 69 06                 ADC  #6
 850:   03A1 69 30      KISEBB     ADC  #"0"       ; Atvaltas ASCII-ba
 860:   03A3 29 3F                 AND  #%111111   ; Atvaltas kepernyokodra
 870:   03A5 91 FB                 STA  (PNT),Y
 880:   03A7 A9 07                 LDA  #SZIN      ; es a szin beallitasa
 890:   03A9 99 1C D8              STA  COLORRAM+28,Y
 900:   03AC C8                    INY
 910:   03AD 60                    RTS
  033C-03AE
NO ERRORS
```

A kezdőértékek beállítása (inicializálás) a korábbiaktól kicsit eltér. Az IRQ vektor régi és új tartalma között végrehajtunk egy megengedő vagy logikai műveletet, ezzel elérjük, hogy minden SYS 828 utasítás kiváltson egy átkapcsolást a régi és az új megszakítási rutin között (a régi rutin kezdőcíme: $EA31, az új rutin kezdőcíme: KIJELZES).
Ha a kijelzést már nem akarjuk tovább folytatni, a SYS 828 utasítással visszaállíthatjuk a megszakítási vektor kezdőcímét $EA31-re.
A program maga is tartalmaz egy főprogram részt, amely induláskor a fontos tárcímek tartalmát a verembe helyezi, így ezeket egy másik program is felhasználhatja. Ezután a PNT mutatóit az első sor 28-ik oszlopára állítjuk, betöltjük az adatregiszter tartalmát és meghívjuk a kiíró rutint. Kiírás után az Y regiszter tartalmát megnöveljük 40-nel, így a következő kiírás egy sorral lejjebb kezdődik.
Most kiírjuk a user port tartalmát. Végül a mutatókat újra visszaolvassuk és ugrunk az eredeti megszakítási rutinba.
A program az akkumulátor tartalmát egyszer bináris, egyszer pedig hexadecimális alakban írja ki. A bináris alakot egy nyolc lépéses ciklusban határozzuk meg.
Minden lépésben leválasztjuk a mindenkori legfelső bitet, azaz az ASL utasítással áttöltjük az átviteli (carry) kapcsolóba. Ha a bit értéke 1 volt, akkor a C kapcsoló értéke is 1, tehát ez kerül a képernyőre, egyébként pedig 0.
A bináris kiírás után az eredeti értéket a veremből visszatöltjük, és kiírjuk hexadecimális alakban is. Ehhez először eltoljuk a felső félbyte-ot négy bittel jobbra, a kapott érték ASCII kódja alapján az eredményt kiírjuk a képernyőre. Ugyanezt végrehajtjuk az alsó félbyte-tal is.
Ha a rutint SYS 828 utasítással elindítjuk, a képernyőn az alábbiak jelennek meg:

```
00000000 00
11111111 FF
```

A kiírás a bekapcsolás utáni állapotot mutatja. Ha a user portot bemenetre kapcsoljuk, a nyitott bemeneti lábak magas jelszintet közvetítenek.

Kapcsoljuk a user portot kimenetre, és írjunk bele 100-at.

```
POKE 56579, 255
POKE 56577, 100
```

Most a képernyő tartalma a következő:

```
11111111 FF
01100100 64
```

A 2-es, 5-ös és 6-os bitek tartalma 1, így a hexadecimális számérték $64.
A következő rutin is az előzőhöz hasonlóan működik.
Minden pillanatban tájékoztat bennünket arról, hogy mennyi szabad tárterület áll még a rendelkezésünkre.
Minden megszakítási lépésben végrehajtjuk a FRE műveletet, azaz meghatározzuk a tömbváltozók vége és a karakteres változók kezdete közötti tárterület nagyságát.
Ellentétben a FRE függvénnyel, a megszakítási rutinban nem végzünk „nagytakarítást", azaz nem ugrunk a Garbage Collect rendszerrutinra. Ez egyrészt nagyon hosszadalmas lenne, másrészt meghamisítaná a valós helyzetet.
Ha a szabad tárkapacitást decimális alakban szeretnénk kiírni, az adott értéket először lebegőpontos számmá, majd ASCII formátumúvá kellene alakítanunk. Az ilyen eljárás alapvető hátránya az lenne, hogy a kezelt tárcím tartalmát átmenetileg mindig tárolnunk kellene a veremben, hiszen a BASIC program futása bármikor megszakadhat. Ez legalább 20 tárcím tartalmának állandó mentését jelentené, ami egyrészt idő-, másrészt helypazarlás, és az sem biztos, hogy ennyi szabad hely egyáltalán rendelkezésünkre áll a veremben. Ezért a szabad tárkapacitást célszerűbb hexadecimális alakban kiírni. A kiírás így is értelmes, a megoldás pedig jóval gyorsabb.

```
***      P28.      ***

PROFI-ASS 64 V2.0

100:    033C                    .OPT P,00
110:                    ;
120:                    ; A szabad tarkapacitas kijelzese
130:                    ;
140:    0031            ARRAYEND  =    $31
150:    0033            STRGSTRT  =    $33
160:                    ;
170:    0400            VIDEO     =    1024
180:    D800            COLOR     =    $D800
190:                    ;
200:    0007            SZIN      =    7          ; Sarga
210:                    ;
220:    0314            IRQVEC    =    $314
230:    EA31            IRQALT    =    $EA31
240:                    ;
250:    033C            INIT      *=   828
260:    033C 78                   SEI
270:    033D A9 49                LDA  #< FREE
280:    033F A0 03                LDY  #> FREE
290:    0341 8D 14 03             STA  IRQVEC
300:    0344 8C 15 03             STY  IRQVEC+1
310:    0347 58                   CLI
320:    0348 60                   RTS
330:                    ;
340:    0349 38         FREE      SEC
```

```
350:   034A A5 33              LDA  STRGSTRT
360:   034C E5 31              SBC  ARRAYEND
370:   034E 08                 PHA
380:   034F A0 25              LDY  #37
390:   0351 20 61 03           JSR  KIIRAS
400:   0354 28                 PLP
410:   0355 A5 34              LDA  STRGSTRT+1
420:   0357 E5 32              SBC  ARRAYEND+1
430:   0359 A0 23              LDY  #35
440:   035B 20 61 03           JSR  KIIRAS
450:   035E 4C 31 EA           JMP  IRQALT
470:   0361 48           KIIRAS PHA
480:   0362 4A                 LSR
480:   0363 4A                 LSR
480:   0364 4A                 LSR
480:   0365 4A                 LSR
490:   0366 20 6A 03           JSR  ASCII
500:   0369 68                 PLA
510:   036A 29 0F        ASCII AND  #%1111
520:   036C C9 0A              CMP  #10
530:   036E 90 02              BCC  KISEBB
540:   0370 69 06              ADC  #6
550:   0372 69 30        KISEBB ADC  #"0"
560:   0374 29 3F              AND  #%111111
570:   0376 99 00 04           STA  VIDEO,Y
580:   0379 A9 07              LDA  #SZIN
590:   037B 99 00 D8           STA  COLOR,Y
600:   037E C8                 INY
610:   037F 60                 RTS
 033C-0380
NO ERRORS
```

A rutin elindítása után (SYS 828) a képernyőn állandóan látható a szabad tárkapacitás.
Próbáljuk ki azt a következő BASIC programmal:

```
***      P29.      ***

100 DIM A$(200)
110 FOR I=1 TO 200 : A$(I)=CHR$(I) : NEXT
```

A képernyőn megjelenik a rendelkezésre álló tárterület hexadecimális értéke. Írjuk be most a ? FRE (0) utasítást. Kb. 4 másodpercet igényel a függvény értékének kiszámítása, miközben a képernyőn állandóan figyelhetjük, hogy hogyan változik a szabad tárkapacitás.
Ha az Olvasó a PROFI-ASS 64 programmal dolgozik, előzetesen elindítva a P.28-as gépi kódú rutint, az első menetben (PASS 1) megfigyelheti, hogy a fordítóprogram hogyan építi fel a szimbólumtáblázatot, ugyanis a BASIC program és a fordító azonos tármutatókat használ.

2.4 A VIDEOVEZÉRLŐ MEGSZAKÍTÁSAI

Miután megismerkedtünk a számlálók által vezérelt rendszermegszakítással és láttuk ennek gazdag felhasználási lehetőségeit, nézzük meg, hogyan lehet programból megszakításokat előidézni.
Mielőtt a feladat megoldásához hozzákezdünk, érdemes megvizsgálni, hogy hogyan dolgoznak azok a chipek, amelyek képesek rendszermegszakítást kiváltani.
A 6526-os chipek kiválthatnak rendszermegszakításokat, a CIA 1 az IRQ vonalon, a CIA 2 pedig az NMI vonalon keresztül. A 6569-es videovezérlő is rendelkezik ezzel a tulajdonsággal. A megszakítások kezelése az alábbi regiszterek feladata:

18. regiszter

Hozzáférés: READ
Olvasásnál a regiszterből megkapjuk annak a rasztersornak a számát, amelynek kijelzése éppen folyamatban van.
Íráskor megadhatjuk azt a rasztersort, amelynek kijelzése közben a VIC IRQ megszakítást vált ki.

25. regiszter

Interrupt Request Register
Ez a regiszter jelzi a VIC megszakítási kérelmét. Az egyes bitek tartalma utal a megszakítási kérelem forrására.

0. bit A vezérlő felírja a 18-as regiszterben megadott rasztersort.
1. bit Egy sprite megérintett egy háttérkaraktert.
   A sprite száma a 31-es regiszterben található.
2. bit Két sprite ütközött egymással.
   A sprite-ok címe a 30-as regiszterben található.
3. bit A fényceruza (lightpen)
   STROBE megszakítást váltott ki. A képernyő pozíció X- és Y-koordinátái a 19-es és a 20-as regiszterben találhatók.
7. bit Értéke mindig egy másik bit értékével együtt változik.

26. regiszter

Interrupt Mask Register
A regiszter bitjeinek jelentése azonos a 25-ös regiszter bitjeinek jelentésével.
A megszakítás engedélyezett, ha az IMR megfelelő bitje 1.

30. regiszter

Sprite – sprite ütközés
Ha két sprite ütközik, akkor a regiszterben a nekik megfelelő bit, ill. a 25-ös regiszter 2. bitje 1 lesz.

31. regiszter	Sprite – háttér ütközés Ha egy sprite ütközik egy háttérkarakterrel, akkor a sprite-nak megfelelő bit értéke a 31-es regiszterben és az 1. bit értéke a 25-ös regiszterben 1-re változik.

A videovezérlő által kiváltott megszakítások típusai:

Rasztersor
Sprite – háttér ütközés
Sprite – sprite ütközés
Fényceruza

A vezérlő a 25-ös regiszter tartalma alapján értesül arról, hogy fellépett-e a négy kiváltó ok valamelyike?
Hogy az esemény megszakítás-kérelemhez vezethet-e, azt az Interrupt Mask Register tartalma dönti el. A megszakítás csak abban az esetben jöhet szóba, ha az IMR első bitje 1 értékű. A közönséges RAM címekkel szemben az IMR tartalmát nem lehet a szokott módon sem beolvasni, sem felülírni. A megszakítás engedélyezését jelző biteket a következőképpen lehet törölni, ill. beállítani:

Egy bit beállítása

A fentiek szerint a megszakítás a rendszer az IMR első (azaz 7-es sorszámú) ill. az eseményhez hozzárendelt bit értéke alapján engedélyezi. Ha pl. a megszakítást sprite – sprite ütközés váltotta ki, akkor ezt a 2. bit 1 értéke jelzi.

```
***       P29/1.       ***

LDA #%10000100
STA IMR
```

A fenti utasításpár hatása megfelel az elvárásoknak, a megfelelő bitek értéke 1 lesz, a 0 bitek értéke pedig nem változik.

Egy bit törlése

Ha szeretnénk egy megszakítási típust letiltani, akkor az eseményhez rendelt bit értékét ismét 1-re, de az engedélyezést jelző bitet 0-ra kell állítani.

```
***       P30.       ***
LDA #%00000100,
STA IMR
```

A bitek értéke ismét nem változik. Az IMR tartalmát nem lehet programból olvasni. Ha a megszakítási maszk ismeretére mégis szükségünk van, nincs más megoldás, mint az, hogy egy tárcím tartalmát a RAM-ban vele párhuzamosan változtatjuk, így az mindig az IMR tartalmát tükrözi.
A videovezérlő megszakításainak másik sajátossága, a „megszakítási kérelem regiszter" (Interrupt Request Register) kezelése.

Valahányszor lezajlott egy megszakítás, az IRR eredeti állapotát vissza kell állítanunk, ellenkező esetben ugyanis visszatérve a megszakítási rutinból, azonnal kiváltódna egy újabb (nem tervezett) megszakítás. Az 1 értékű bitet most is az előző módszerrel törölhetjük. A legegyszerűbb, ha az IRR tartalmát beolvassuk, és azonnal vissza is írjuk.

```
***       P31.       ***

LDA IRR
STA IRR
```

Mivel a bitminta az akkumulátorban van, maszkolással vizsgálhatjuk az egyes biteket. Erre minden olyan esetben szükség van, amikor egyszerre több megszakítási forrás aktív, például a számláló rendszermegszakítása, és valamelyik, a videovezérlő által generált megszakítás. Mindkét megszakítás azonos vektort használ, így elágazás előtt el kell dönteni, hogy mi volt a megszakítás forrása, és az elágazást a forrás alapján kell végrehajtani. Ha a fentiek némileg bonyolultnak tűnnének az Olvasó számára, egy példa elemzése biztosan segít a tájékozódásban:
Tegyük fel, hogy egy rasztermegszakítást akarunk előidézni annak érdekében, hogy a képernyőn egyszerre 16 sprite-ot tudjunk ábrázolni. A videovezérlő csak 8 sprite egyidejű kezelésére van felkészítve, a feladatot tehát csak úgy tudjuk megoldani, ha egy ügyes trükkel 8–8 sprite-ot jelenítünk meg egymás után.

A folyamat a következő:
A képernyő felső felén ábrázolunk 8 sprite-ot.
Miután a videovezérlő előállította a képernyő felső felét, kiváltunk egy megszakítást. A megszakítási rutinban beállítjuk annak a 8 sprite-nak a paramétereit, amelyeket a képernyő alsó felén szeretnénk láthatóvá tenni. Ezzel egyidejűleg előkészítettünk egy újabb megszakítást, amellyel visszakapcsolunk az eredeti 8 sprite-ra.

```
***       P32.       ***

PROFI-ASS 64 V2.0

100:   033C                    .OPT P,00
110:                    ;
120:                    ; Rasztermegszakito
130:                    ;
140:   D000             VIC     =    $D000      ; Videovezerlo
150:   D001             SPRITEY =    VIC+1      ; Sprite Y-koordinata
160:   D012             RASTER  =    VIC+18     ; Rasztersor
170:   D019             IRR     =    VIC+25     ; Interrupt request regiszter
180:   D01A             IMR     =    VIC+26     ; Interrupt maszk regiszter
190:   0064             SOR1    =    100        ; Elso sor
200:   00C8             SOR2    =    200        ; Masodik sor
202:   005A             YKOORD1 =    90         ; Elso Y-koordinata
203:   00AA             YKOORD2 =    170        ; Masodik Y-koordinata
210:                    ;
220:   0314             IRQVEC  =    $314
230:   EA31             IRQREGI =    $EA31
240:                    ;
300:   033C                    *=828
310:   033C 78          INIT    SEI
320:   033D A9 64               LDA  #SOR1      ; Az elso megszakitas
330:   033F 8D 12 D0            STA  RASTER     ; A 100-as sornal
340:   0342 AD 11 D0            LDA  RASTER-1
350:   0345 29 7F               AND  #%01111111 ; A felso byte
360:   0347 8D 11 D0            STA  RASTER-1
```

```
370:   034A A9 81          LDA  #%10000001  ; A rasztersor altali
380:   034C 8D 1A D0       STA  IMR         ; megszakitas
390:   034F A9 5B          LDA  #< TESTIRQ
400:   0351 A0 03          LDY  #> TESTIRQ
410:   0353 8D 14 03       STA  IRQVEC      ; Vektor az uj
420:   0356 8C 15 03       STY  IRQVEC+1    ; rutinra mutat
430:   0359 58             CLI
440:   035A 60             RTS
450:                ;
460:   035B AD 19 D0 TESTIRQ LDA  IRR        ; A regiszter beolvasasa
470:   035E 8D 19 D0       STA  IRR         ; es torlese
480:   0361 29 01          AND  #%1         ; Rasztersor generalta az IRQ-
490:   0363 D0 03          BNE  OK          ; Igen
500:   0365 4C 31 EA       JMP  IRQREGI     ; Normal IRQ
510:                ;
520:   0368 AD 12 D0 OK     LDA  RASTER      ; Aktualis sor
530:   036B C9 C8          CMP  #SOR2       ; >= mint a 2.sor
540:   036D B0 16          BCS  SECOND      ; Igen
545:                ;
550:   036F A0 C8          LDY  #SOR2       ; A kovetkezo IRQ a 2.sorban
555:   0371 A9 AA          LDA  #YKOORD2    ; Uj SPRITE koordinata
560:   0373 8C 12 D0 VISSZA STY  RASTER     ; A rasztersor beallitasa
570:   0376 A2 0E          LDX  #14
590:   0378 9D 01 D0 CIKLUS1 STA  SPRITEY,X  ; SPRITE koordinatak
600:   037B CA             DEX               ; modositas
600:   037C CA             DEX
610:   037D 10 F9          BPL  CIKLUS1
620:                ;
630:   037F 68             PLA               ; A regiszter visszatoltese
640:   0380 A8             TAY
650:   0381 68             PLA
660:   0382 AA             TAX
670:   0383 68             PLA
680:   0384 40             RTI
690:                ;
700:   0385 A0 64    SECOND LDY  #SOR1        ; AZ elso sor parametere
710:   0387 A9 5A          LDA  #YKOORD1
720:   0389 4C 73 03       JMP  VISSZA
 033C-038C
NO ERRORS
```

Az elkészült rutint csak úgy tudjuk kipróbálni, ha előzetesen 8 sprite-ot aktivizálunk. Erre szolgál a P.33-as BASIC program. Ha ezt lefuttatjuk, majd SYS 828 utasitással elindítjuk a P.32-es rutint, hirtelen felvillan a képernyőn 16 sprite. Közülük 8-nak az y-koordinátája 80, a többié pedig 170. Amint a képernyön megjelenik a felső 8 sprite, a paramétereket a megszakitási rutinok úgy változtatják meg, hogy ugyanez a 8 sprite most a képernyő alsó felén legyen látható.

```
***       P33.       ***
100 FOR I=0 TO 7 : POKE 2040+I,12 : NEXT
110 V=53248
120 POKE V+21,255
130 FOR I=0 TO 7 : POKE V+2*I,(I+1)*30 : POKE V+2*I+1,70 : NEXT
140 FOR I=0 TO 7 : POKE V+39+I,1 : NEXT
```

A koordinátákon kívül természetesen más paramétereket is módosithatunk, például a sprite szinét vagy nagyságát. Sőt, ha a sprite mutatóját változtatjuk meg úgy, hogy az egy másik sprite-mintára mutasson, akkor az eredetiektől egészen eltérő figurákat is előállithatunk a képernyő alsó felén.
A rasztermegszakitási rutinba beépitve a képernyő üzemmódjának megváltoztatását, osztott képet is előállithatunk, aminek eredményeként pl. a képernyő felső felén egy finom felbontású grafika, az alsó felén egy szöveg látható.
A kétféle ábrázolási üzemmód hatását akár egy BASIC programból is állandóan változtathatjuk, ha a megszakitást kiváltó rasztersor értékét egy RAM-beli címen tároljuk. Az alkalmazási lehetőségek száma itt is végtelen.

2.5 A CIA 6526-OS MEGSZAKÍTÁSAI

A CIA 6526-os processzor nagyon sokféle megszakítás forrása lehet.
A CIA 6526-os egy univerzális input/output elem, amelybe két párhuzamos 8 bites port, egy soros eltolási regiszter, két 16 bites számláló, egy valós idejű óra és több handshake vonal van beépítve.
A 8 bites párhuzamos portok bonyolítják le az adatforgalmat, azaz az adatok beolvasását és kiírását. A két CIA összesen 4 portot tartalmaz, amelyből hármat használ az operációs rendszer; a CIA 1 két portja felel a billentyűzet és a botkormányok lekérdezéséért. A CIA 2 A portja lebonyolítja a videovezérlő 16 K-s tartományának kiválasztását (a 0. és 1. bit), 2. bitje szabad, míg a 3-tól a 7-ig terjedő bitjeit a soros busz használja. A B port user portként teljesen a használó rendelkezésére áll, hacsak nem csatlakoztattunk ide a soros adatátvitelhez egy RS 232-es betétet (cartridge-ot).
Az operációs rendszer a számlálókat az alábbiak szerint használja:

CIA 1	A számláló	60 Hz-es rendszermegszakítások
	B számláló	soros busz (time out) kazetta olvasás, írás
CIA 2	A számláló	továbbítás az RS 232-eshez
	B számláló	fogadás az RS 232-esről

A CIA 2 számlálóit bármilyen saját célra igénybevehetjük. A CIA 2 nem IRQ, hanem NMI típusú megszakítást vált ki. A CIA 1 B számlálóját is használhatjuk, ha munka közben nincs szükségünk a soros buszra, ez esetben IRQ megszakítást is létrehozhatunk. Sőt, ha a rendszermegszakításokról lemondunk, az A számlálót is saját elképzelésünk szerint használhatjuk.
A valós idejű órát az operációs rendszer egyáltalában nem használja, ami számunkra a következő lehetőségeket nyújtja: riasztó idővel tetszés szerint kiválthatunk egy IRQ (CIA 1) vagy egy NMI (CIA 2) megszakítást.
A soros tolóregiszterrel is szabadon gazdálkodhatunk. A FLAG vonal, amely handshake-bemenetként szolgál, lefutó éllel 1-re állítja a CIA 2 ICR (Interrupt Control Register) megfelelő bitjét.
Az input/output – és a handshake-vonalak elsősorban speciális külső egységek csatlakoztatására szolgálnak. Ezzel a témakörrel foglalkozik a „A Commodore 64-es és a világ másik fele" c. könyv*. Ezen a területen a megszakítások programozása szintén főszerepet játszik. Mi is foglalkozunk példaként egy nyomtató user portra csatlakoztatásával, azonban ez a könyv elsősorban a rendszerrutinok elemzését adja, így a külső egységek kezelését csak az ide vonatkozó BASIC utasítások kapcsán (OPEN, PRINT stb.) említjük meg. Következő példánk egy ébresztőórát készít a Commodore 64-esből.
A programban a valós idejű órát használjuk, és a megfelelő időben bekövetkező ébresztést a CIA 2 NMI megszakításán keresztül generáljuk.

* Eredeti cím: Der Commodore 64 und der Rest der Welt. Magyarul előkészületben (a szerk.)

PROFI-ASS 64 V2.0

```
100:  033C                        .OPT P,00
110:                              ;
120:                              ; EBRESZTOORA A CIA2-vel
130:                              ;
140:  DD00                        CIA2      =     $DD00       ; A CIA baziscime
150:  DD08                        TOD10     =     CIA2+8      ; Tized masodperc
160:  DD09                        TODSEK    =     CIA2+9      ; Masodperc
170:  DD0A                        TODMIN    =     CIA2+10     ; Perc
180:  DD0B                        TODSTD    =     CIA2+11     ; Ora
190:                              ;
200:  DD0D                        ICR       =     CIA2+13     ; Interrupt kontroll register
210:  DD0E                        CRA       =     CIA2+14     ; Kontroll regiszter A
220:  DD0F                        CRB       =     CIA2+15     ; Kontroll regiszter B
230:  D020                        BORDER    =     $D020       ; A keret szine
240:  0002                        ROT       =     2
250:                              ;
260:  0318                        NMI       =     $318        ; NMI vektor
270:  FE56                        CONTNMI   =     $FE56       ; A regi NMI
280:                              ;
290:                              ; ORAIDO            12h 00' 00.0''
300:  0000                        TIZED     =     0
310:  0000                        SEC       =     $00
320:  0000                        MIN       =     $00
330:  0012                        ORA       =     $12
340:                              ;
350:                              ; EBRESZTESI IDO  12h 00' 05.0''
360:  0000                        EBR.10    =     0
370:  0005                        EBR.SK    =     $05
380:  0000                        EBR.MN    =     $00
390:  0012                        EBR.ST    =     $12
400:                              ;
410:  033C                              *=    828
420:                              ;
430:                              ; AZ IDO BEALLITASA
440:  033C AD 0E DD                     LDA   CRA
450:  033F 09 80                        ORA   #$80          ; 50 Hz-s oratrigger
460:  0341 8D 0E DD                     STA   CRA
470:                              ;
480:  0344 AD 0F DD                     LDA   CRB
490:  0347 29 7F                        AND   #$7F          ; Az ido beallitasa
500:  0349 8D 0F DD                     STA   CRB
510:                              ;
520:  034C A9 12                        LDA   #ORA
530:  034E 8D 0B DD                     STA   TODSTD
540:  0351 A9 00                        LDA   #MIN
550:  0353 8D 0A DD                     STA   TODMIN
560:  0356 A9 00                        LDA   #SEC
570:  0358 8D 09 DD                     STA   TODSEK
580:  035B A9 00                        LDA   #TIZED
690:  035D 8D 08 DD                     STA   TOD10
600:                              ;
610:  0360 AD 0F DD                     LDA   CRB
620:  0363 09 80                        ORA   #$80          ; Ebresztesi ido
630:  0365 8D 0F DD                     STA   CRB
640:                              ;
650:  0368 A9 12                        LDA   #EBR.ST
660:  036A 8D 0B DD                     STA   TODSTD
670:  036D A9 00                        LDA   #EBR.MN
680:  036F 8D 0A DD                     STA   TODMIN
690:  0372 A9 05                        LDA   #EBR.SK
700:  0374 8D 09 DD                     STA   TODSEK
710:  0377 A9 00                        LDA   #EBR.10
720:  0379 8D 08 DD                     STA   TOD10
730:                              ;
740:  037C A9 84                        LDA   #%10000100 ; Ebresztes
750:  037E 8D 0D DD                     STA   ICR          ; NMI engedelyezes
760:                              ;
```

```
0370:  0381 A9 8C          LDA  #< TEST
0380:  0383 A0 03          LDY  #> TEST
0390:  0385 8D 18 03       STA  NMI        ; Az uj NMI-vektor
0A00:  0388 8C 19 03       STY  NMI+1
0A10:  038B 60             RTS
0A20:                ;
0A30:  038C 48       TEST  PHA
0A40:  038D 8A             TXA
0A50:  038E 48             PHA             ; A regiszter mentese
0A60:  038F 98             TYA
0A70:  0390 48             PHA
0A80:  0391 AC 0D DD       LDY  ICR
0A90:  0394 98             TYA
0AA0:  0395 29 04          AND  #%100      ; A bit 1 ?
0AB0:  0397 D0 03          BNE  EBR.       ; Igen
0AC0:  0399 4C 56 FE       JMP  CONTNMI
0AD0:                ;
0AE0:  039C A9 02    EBR.  LDA  #ROT
0AF0:  039E 8D 20 D0       STA  BORDER     ; A keret piros
0B00:                ;
0B10:  03A1 68             PLA
0B20:  03A2 A8             TAY
0B30:  03A3 68             PLA
0B40:  03A4 AA             TAX
0B50:  03A5 68             PLA
0B60:  03A6 40             RTI
1033C-03A7
NO ERRORS
```

A program először rögzíti a valós idejű óra és a CIA 2 vezérlőregiszterének címét. Ezután beállítja az időt 12 órára, az ébresztési időt pedig 12 óra 5 percre. Az óratrigger értéke 50 Hz lesz, így az óra pontosan fog számolni. Ezután a program törli a B vezérlőregiszter 7. bitjét, ezzel jelezve a CIA számára, hogy szeretnénk a pontos időt beállítani. Most 1-re állítjuk a 7. bitet, meghatározzuk az ébresztési időt és az ICR regiszterben a megszakítást (NMI) engedélyezzük. Ehhez a 7. és a 2. bitek értékét kell 1-re állítani. Végül az NMI vektor címét úgy módosítjuk, hogy az a mi új rutinunkra mutasson, és ezzel a kezdeti értékek meghatározását (inicializálást) befejeztük.

A tulajdonképpeni NMI rutin elkészítése nem jelent nehéz feladatot. Először a regiszterek tartalmát a verembe mentjük, majd beolvassuk az ICR tartalmát és megvizsgáljuk a 2. bit értékét. Ha ez nem 1, akkor ideje ébreszteni.

Az ébresztést szimbolikusan a képernyő színének pirosra váltásával jelezzük. Most visszaállítjuk a regiszterek eredeti tartalmát és RTI utasítással visszatérünk a megszakított programba.

Ha az NMI kiváltója nem az volt, hogy az idő azonos az ébresztési időponttal, akkor visszatérünk az operációs rendszer NMI rutinjára. Ez a rutin megvizsgálja, hogy a RESTORE billentyűvel együtt a STOP billentyű is le van-e nyomva (az NMI-t ugyanis a RESTORE billentyű váltja ki). Ha igen, akkor egy melegstart végrehajtása következik.

A program kidolgozásánál nem tartottuk fontosnak azt a részt, amely a riasztási időpontban lejátszódó eseményt generálja. Az Olvasó ízlése és igénye szerint készítheti el ezt a programrészt. Ha valóban ébresztőórát akarunk programozni, akkor egy hangkeltő rutint célszerű ide beiktatni, és érdemes egy kényelmes beolvasó rutint készíteni a kezdeti idő és az ébresztési idő bevitelére.

2.6 A SZÁMLÁLÓK (TIMEREK) FELHASZNÁLÁSA

Mindkét CIA tartalmaz két 16 bites számlálót, amelyek lefutása kiválthat egy megszakítást. A számlálók értéke minden processzor-ütemben eggyel csökken. Ha a tárolt számérték nullára csökkent, az ICR (Interrupt Control Register) megfelelő bitjének értéke 1 lesz – és ha ugyanakkor a bitmaszk megszakítást engedélyező bitje is 1, akkor létre is jön az IRQ vagy az NMI megszakítás.
A Commodore 64-es a nyugat-német PAL rendszer szerint kb. 985 kHz ütemfrekvenciával működik, így egy ütemciklus 1,015 s, ill. kerekítve 1 s ideig tart. A 16 bites számlálók maximális számértéke 65535, így a lehetséges megszakítások közötti időtartam maximálisan 65535 ütemciklus, azaz 65 ms, a másodpercnek kb. egytizenötöd része. Pl. ha a CIA 1 számlálóba a $4025 számértéket töltöttük, az 16421 ütemciklusnak kb. egyhatvanad másodperc felel meg.
Az amerikai NTSC rendszerben az ütemfrekvencia 1.02 MHz. Ott a számlálóba töltött $4295, azaz decimálisan 17045 felel meg a hatvanad másodpercenkénti lefutásnak.
A számlálók különböző üzemmódban működhetnek, megkülönböztetünk egy ún. „egy lövés" (one shot) és egy folytonos (continuous) üzemmódot. Az első esetben a számláló a kezdőértékről nulláig számol, majd leáll, a folytonos üzemmódban pedig amint elérte a nullát, automatikusan felveszi a kezdőértéket és a számlálás újra indul. A megszakítások kiváltásán kívül a számlálók arra is képesek, hogy egy impulzust küldjenek. Ezt felhasználhatjuk pl. egy külső egység ütemjeleként. A számlálók kezdőértékét kívülről is meghatározhatjuk, a számlálását (csökkentését) is végezheti egy kívülről érkező jel (nemcsak a rendszerütem). A számlálók a kapcsolatteremtés eszközei is lehetnek.- Képesek együttműködni olyan módon, hogy amint az egyik értéke nullára csökkent, a másik megkezdi a számlálást. Ez lehetőség arra, hogy egy 32 bites számlálót imitáljunk, amely 2^{32}, azaz 4 294 967 296 számú ütemciklust, tehát 4360 másodperc (= 1 óra 12 perc) hosszúságú időtartamot jelent.
A megszakítások programozásáról szóló fejezetünket egy olyan gépi kódú program ismertetésével zárjuk, amely lehetővé teszi, hogy BASIC szubrutint futtassunk megszakításvezérléssel. Eközben megismerkedünk a számlálók programozásának módszerével és ugyanakkor a BASIC értelmező működéséből is újabb ízelítőt kapunk.
A feladat egy olyan új BASIC utasítás előállítása, amellyel egy tetszőleges szubrutint a programból bizonyos idő elteltével ismételten meghívhatunk.
A feladat megoldásához még szükségünk van némi elméletre.
A BASIC értelmező egy BASIC program feldolgozásának főciklusában egy olyan rész, amely értelmezi és végrehajtja az utasítást. Minden utasítás után megvizsgálja, hogy a STOP billentyűt időközben lenyomták-e. Ha igen, kilép a főciklusból és visszatér közvetlen üzemmódra. A STOP billentyű vizsgálatát egy ugrással végzi el. Ezt ugrási vektorral változtatjuk meg úgy, hogy az egy másik rutinra mutasson. Ebben a rutinban megvizsgáljuk, hogy a megszakítás feltételei teljesülnek-e, vagyis a számlálónk lefutott-e már. Ha igen, akkor egy valódi megszakítási rutin egy kapcsoló értékét 1-re állítja, ennek alapján a mi rutinunk is tájékozódhat.

Az új BASIC utasítás paramétereként megadhatjuk, hogy melyik az a BASIC rutin, amelyet időközként ismételni szeretnénk. Az utasítás második paramétere megadja a megszakítások közötti időtartamot.

```
! GOSUB 1000,100
```

A felkiáltójel megkülönbözteti az új utasítást a régi megszokott GOSUB utasítástól. Az időközönként végrehajtandó rutin az 1000-es sorban kezdődik, az időtartam pedig 100. Az időegység kb. egyötvened másodperc. Ezt az értéket töltjük az egyik számlálóba. A második értéket a következő számlálókba töltjük, amelyeket 16 bites számlálóként használunk. Így a programozható időhöz egyötvened másodperctől 65535 ötvenedmásodpercig, azaz 0.02–1311 másodpercig (1311 másodperc = 21 perc 51 másodperc) tarthat.
A program három rutin inicializálásával indul. Az első úgy módosítja a BASIC értelmezőt, hogy az „megértse" az új BASIC utasítást.
A második minden utasítás végrehajtása után megvizsgálja a számláló-figyelő kapcsoló értékét, és ha szükséges, elágazik a szubrutinba.
A harmadik program a saját megszakítási vagy NMI rutinunk, amely a másik két rutin számára a számláló lefutása után beállítja a kapcsolót.

```
***       P35.      ***
PROFI-ASS 64 V2.0

100:   CC00                    .OPT P,00
110:   CC00                    .TIT "BASIC-IRO"
120:   CC00                    .SYM              ; Szimbolumtablazat kiirasa
130:                    ;
140:                    ; A BASIC megszakitasi rutin
150:                    ;
160:   0308    EXEC    =      $308              ; Az utasitas vegrehajt. vekto
170:   0318    NMI     =      $318              ; NMI-vektor
180:   0328    STOP    =      $328              ; STOP-vektor
190:                    ;
200:   DD00    CIA2    =      $DD00
210:   DD04    TIMERA  =      CIA2+4            ; Timer A
220:   DD06    TIMERB  =      CIA2+6            ; Timer B
230:   DD0D    ICR     =      CIA2+13           ; Interrupt Control Register
240:   DD0E    CRA     =      CIA2+14           ; Control Register A
250:   DD0F    CRB     =      CIA2+15             Control Register B
260:                    ;
270:   FE56    CONTNMI =      $FE56             ; A regi NMI visszaallitasa
280:                    ;
290:   4CF9    TIME    =      19705             ; =20 Millisec.
300:   0014    LO      =      $14               ; A sorszam also byte-ja
310:   0015    HI      =      LO+1
320:   005F    LINEADR =      $5F               ; A BASIC sor cime
330:   0039    LINENO  =      $39               ; Futo sorszam
340:   0073    CHRGET  =      $73
350:   0079    CHRGOT  =      CHRGET+6
360:   007A    TXTPTR  =      CHRGOT+1
370:   008D    GOSUB   =      $8D               ; GOSUB-token
380:   AF08    SYNTAX  =      $AF08             ; SYNTAX ERROR
390:   A8E3    UNDEFD  =      $A8E3             ; UNDEF'D STATEMENT ERROR
400:   B248    ILLQUAN =      $B248             ; ILLEGAL QUANTITY ERROR
410:   A7AE    INTER   =      $A7AE             ; Az interpreter-ciklus
420:   A96B    GETLIN  =      $A96B             ; A sorszam betoltese
430:   A613    GETADR  =      $A613             ; Sor keresese
440:   AEFD    CHKCOM  =      $AEFD             ; A vesszo vizsgalata
450:   A7E7    EXECOLD =      $A7E7             ; Az utasitas vegrehajtasa
460:   AD8A    FRMNUM  =      $AD8A             ; Numerikus ertek betoltese
470:   B7F7    INTEGER =      $B7F7             ; es atvaltasa egeszre
```

```
480:  A3FB              TESTSTACK=  $A3FB       ; Hely keresese a veremben
490:  F6ED              TESTOLD   =  $F6ED       ; A STOP billentyu vizsgalata
500:  FE47              NMIOLD    =  $FE47       ; A regi NMI vektor
510:                    ;
520:  CC00                        *=  $CC00
530:  CC00 A9 10        INIT      LDA #< TESTSTAT
540:  CC02 A0 CC                  LDY #> TESTSTAT
550:  CC04 8D 08 03               STA EXECIN     ; Ugrasdekodolas rutinja
560:  CC07 8C 09 03               STY EXEC+1     ; A '!' hozzafuzese
570:  CC0A A9 00                  LDA #0
580:  CC0C 8D F7 CC               STA FLAG       ; A kapcsolo torlese
590:  CC0F 60                     RTS
600:                    ;
610:  CC10 20 73 00 TSTGOSUB JSR  CHRGET         ; A kovetkezo karakter betoltese
620:  CC13 C9 21                  CMP #"!"
630:  CC15 F0 06                  BEQ TSTGOSUB
640:  CC17 20 79 00               JSR CHRGOT     ; A kapcsolo helyreallitasa,
650:  CC1A 4C E7 A7               JMP EXECOLD    ; es tovabb
660:                    ;
670:  CC1D 20 73 00 TSTGOSUB JSR  CHRGET         ; A kovetkezo kapcsolo
680:  CC20 C9 8D                  CMP #GOSUB     ; GOSUB-kod ?
690:  CC22 F0 03                  BEQ OK         ; Igen
700:  CC24 4C 08 AF               JMP SYNTAX     ; SYNTAX ERROR
710:  CC27 20 73 00 OK            JSR CHRGET     ; A kovetkezo sor
720:  CC2A F0 68                  BEQ IRQOFF     ; Ha sorvege, akkor IRQ
730:  CC2C 20 6B A9               JSR GETLIN     ; A sorszam betoltese
740:  CC2F 20 13 A6               JSR GETADR     ; A sor cimenek betoltese
750:  CC32 B0 03                  BCS FOUND      ; Megvan ?
760:  CC34 4C E3 A8               JMP UNDEFD     ; Nem, UNDEF'D STATEMENT ERROR
770:  CC37 A5 5F        FOUND     LDA LINEADR    ; A sor cime
780:  CC39 E9 01                  SBC #1         ; Minusz 1
790:  CC3B 8D F8 CC               STA LINESTR    ; Tarolasa
800:  CC3E A5 60                  LDA LINEADR+1
810:  CC40 E9 00                  SBC #0         ; Felso BYTE
820:  CC42 8D F9 CC               STA LINESTR+1
830:  CC45 20 FD AE               JSR CHKCOM     ; A vesszo vizsgalata
840:  CC48 20 8A AD               JSR FRMNUM     ; A szam betoltese,
850:  CC4B 20 F7 B7               JSR INTEGER    ; Es atalakitasa egeszre
860:  CC4E A5 14                  LDA LO
870:  CC50 05 15                  ORA HI         ; Felso es also BYTE 0 ?
880:  CC52 D0 03                  BNE OK1        ; Nem
890:  CC54 4C 48 B2               JMP ILLQUAN    ; ILLEGAL QUANTITY ERROR
900:  CC57 A5 15        OK1       LDA HI
910:  CC59 8D 07 DD               STA TIMERB+1
920:  CC5C A5 14                  LDA LO         ; Az ertek betoltese a B timerbe
930:  CC5E 8D 06 DD               STA TIMERB
940:  CC61 A9 4C                  LDA #> TIME    ; Az A timerbe
950:  CC63 8D 05 DD               STA TIMERA+1
960:  CC66 A9 F9                  LDA #< TIME    ; 20 ms. betoltese
970:  CC68 8D 04 DD               STA TIMERA
980:  CC6B A9 11                  LDA #%00010001 ; Az A timer indul
990:  CC6D 8D 0E DD               STA CRA
1000: CC70 A9 51                  LDA #%01010001 ; A B timer indul,
1010: CC72 8D 0F DD               STA CRB        ; az A timer triggerezi
1020: CC75 AD 0D DD               LDA ICR        ; Az ICR torlese
1030: CC78 A9 82                  LDA #%10000010 ; A B timer NMI-je
1040: CC7A 8D 0D DD               STA ICR        ; Engedelyezett
1050: CC7D A9 C9                  LDA #< TESTTIME
1060: CC7F A0 CC                  LDY #> TESTTIME
1070: CC81 8D 28 03               STA STOP       ; STOP-vektor beallitasa
1080: CC84 8C 29 03               STY STOP+1
1090: CC87 A9 B0                  LDA #< NMIROUT
1100: CC89 A0 CC                  LDY #> NMIROUT
1110: CC8B 8D 18 03               STA NMI        ; NMI-vektor beallitas
1120: CC8E 8C 19 03               STY NMI+1
1130: CC91 4C AE A7               JMP INTER      ; Az interpreter ciklushoz
1140:                   ;
1150: CC94 A9 7F        IRQOFF    LDA #%01111111
1160: CC96 8D 0D DD               STA ICR        ; Minden megszakitas ki
1170: CC99 A9 ED                  LDA #< TESTOLD
```

```
1180:  CC9B A0 F6          LDY  #> TESTOLD
1190:  CC9D 8D 28 03       STA  STOP        ; A STOP-vektor a regi ertekre
1200:  CCA0 8C 29 03       STY  STOP+1
1210:  CCA3 A9 47          LDA  #< NMIOLD
1220:  CCA5 A0 FE          LDY  #> NMIOLD
1230:  CCA7 8D 18 03       STA  NMI         ; NMI-vektor a regi ertekre
1240:  CCAA 8C 19 03       STY  NMI+1
1250:  CCAD 4C AE A7       JMP  INTER       ; Az interpreter ciklushoz
1260:                ;
1270:  CCB0 48       NMIROUT PHA
1280:  CCB1 8A             TXA
1280:  CCB2 48             PHA
1290:  CCB3 98             TYA
1290:  CCB4 48             PHA
1300:  CCB5 AC 0D DD       LDY  ICR
1310:  CCB8 98             TYA
1320:  CCB9 29 02          AND  #%10        ; A B timer lefutott ?
1330:  CCBB D0 03          BNE  TIMEOUT     ; Igen
1340:  CCBD 4C 56 FE       JMP  CONTNMI     ; Egyebkent normal NMI
1350:                ;
1360:  CCC0 EE F7 CC TIMEOUT  INC  FLAG       ; A kapcsolo beallitasa
1370:  CCC3 68             PLA
1370:  CCC4 A8             TAY
1380:  CCC5 68             PLA
1380:  CCC6 AA             TAX
1390:  CCC7 68             PLA
1390:  CCC8 40             RTI
1400:                ;
1410:  CCC9 AD F7 CC TESTTIME LDA  FLAG       ; A kapcsolo erteke 1 ?
1420:  CCCC D0 03          BNE  TIMEIRQ     ; Igen
1430:  CCCE 4C ED F6       JMP  TESTOLD
1440:                ;
1450:  CCD1 CE F7 CC TIMEIRQ  DEC  FLAG       ; A kapcsolo torlese
1460:  CCD4 68             PLA
1460:  CCD5 68             PLA              ; A visszaugrasi cim a verembol
1470:  CCD6 A9 03          LDA  #3
1480:  CCD8 20 FB A3       JSR  TESTSTACK   ; Van meg eleg hely a veremben ?
1490:  CCDB A5 7B          LDA  TXTPTR+1
1500:  CCDD 48             PHA              ; CHRGET-mutato a veremre
1510:  CCDE A5 7A          LDA  TXTPTR
1520:  CCE0 48             PHA
1530:  CCE1 A5 3A          LDA  LINENO+1
1540:  CCE3 48             PHA              ; Az aktualis sorszam a veremben
1550:  CCE4 A5 39          LDA  LINENO
1560:  CCE6 48             PHA
1570:  CCE7 A9 8D          LDA  #GOSUB
1580:  CCE9 48             PHA              ; A GOSUB-kod a verembe
1590:  CCEA AD F8 CC       LDA  LINESTR
1600:  CCED 85 7A          STA  TXTPTR      ; A szubrutin cime
1610:  CCEF AD F9 CC       LDA  LINESTR+1
1620:  CCF2 85 7B          STA  TXTPTR+1
1630:  CCF4 4C B1 A7       JMP  INTER+3     ; Az interpreter ciklushoz
1640:                ;
1650:  CCF7          FLAG     *=   *+1
1660:  CCF8          LINESTR  *=   *+2
 CC00-CCFA
NO ERRORS


BASIC-IRQ      PROFI-ASS 64 V2.0


SYMBOLTABLE:
LINESTR   CCF8   FLAG      CCF7   TIMEIRQ   CCD1   TESTTIME  CCC9
TIMEOUT   CCC0   NMIROUT   CCB0   IRQOFF    CC94   OK1       CC57
FOUND     CC37   OK        CC27   TSTGOSUB  CC1D   TESTSTAT  CC10
INIT      CC00   NMIOLD    FE47   TESTOLD   F6ED   TESTSTAC  A3FB
INTEGER   B7F7   FRMNUM    AD8A   EXECOLD   A7E7   CHKCOM    AEFD
GETADR    A613   GETLIN    A96B   INTER     A7AE   ILLQUAN   B248
UNDEFD    ABE3   SYNTAX    AF08   GOSUB     080D   TXTPTR    007A
```

```
CHRGOT    0079   CHRGET    0073   LINENO    0039   LINEADR   005F
HI        0015   LO        0014   TIME      4CF9   CONTNMI   FE56
CRB       DD0F   CRA       DD0E   ICR       DD0D   TIMERB    DD06
TIMERA    DD04   CIA2      DD00   STOP      0328   NMI       0318
EXEC      0308
45 SYMBOLS DEFINED
```

A részletes leírás előtt nézzük meg a gépi kódú program hívását egy BASIC programban!

```
***       P36.      ***

100 SYS 52224 : REM A BASIC BOVITES INICIALIZALASA
110 !GOSUB 200,50
120 I=I+1 : PRINT I : IF I<100 GOTO 120
130 !GOSUB
140 END
200 J=J+1 : PRINT J ".IRQ-HIVAS !" : RETURN
```

Amikor a programot RUN paranccsal elindítjuk, első lépésként a 100-as sorban elhelyezett SYS utasítás inicializálja a BASIC bővítését. A 110-es sorban megszakítási alprogramként kijelöljük a 200-as rutint, amelyet minden másodpercben (50 ötvenedmásodperc) ismételten végre kell hajtani. A tulajdonképpeni főprogram a 120-as sorban kezdődik, és semmi egyebet nem tesz, minthogy számol 1-től 100-ig, és a soron következő számot kiírja a képernyőre. Amikor a ciklus végetér, végrehajtunk egy paraméter nélküli !GOSUB utasítást, és ezzel a program futása is befejeződik. A megszakítási rutint a 200-as sorban helyeztük el. Ez a sor minden hívásnál kiír egy szöveget a hívás sorszámával együtt a képernyőre, majd a RETURN utasítás hatására visszatér a főprogramba.
Futtatás közben a program kiírja a képernyőre 1-től 100-ig a számokat, közben ötször megszakítja a kiírást az alábbi sorokkal:

1. IRQ – HIVAS!
.
.
.
5. IRQ – HIVAS

Ha megfelelően megváltoztatjuk a 110-es sorban a második paramétert, megmérhetjük a hívási frekvenciát. A második paraméter értéke 1 és 65535 közé eső szám lehet. Minél kisebb ez az érték, annál gyakoribb lesz a megszakítási rutin hívása. A megszakítási rutin végrehajtási ideje természetesen nem lehet nagyobb, mint a két hívás között eltelt idő, egyébként a szubrutin a saját futását szakítaná félbe, és a ciklikus megszakítástól a verem hamar megtelne. A

```
110 !GOSUB 200,1
```

utasítás a program futási eredményét alaposan megváltoztatná:

```
1
1. IRQ – HIVAS!
2. IRQ – HIVAS!
...
```

22. IRQ – HIVAS!
23. IRQ – HIVAS!
?OUT OF MEMORY ERROR IN 200

A megszakítási rutin hosszát megszabja a maximális hívási ütem.
Térjünk vissza a gépi kódú program ismertetésére.
A 100-tól 500-ig terjedő sorokban rögzítjük a program által használt konstansok értékét. Itt adjuk meg az NMI és a BASIC vektorok címét, majd a CIA 2 regisztereket, amelyek a számláló – megszakításokhoz szükségesek. A 290-es sorban rögzítjük az időegységet. Ezután következnek a nullaslap BASIC (pl. a hibaüzenetek) címei, és az értelmező ROM címei.
Az 520-tól 590-ig terjedő sorokban végrehajtjuk az inicializálást. Ez azt jelenti, hogy itt végzünk el minden olyan rendszermódosítást, amely ahhoz szükséges, hogy a program a mi elképzelésünk szerint működjön. Azt a vektort, amely a BASIC utasítások dekódolását és végrehajtását tartalmazó rutinra mutat, úgy változtatjuk meg, hogy a saját rutinunkra mutasson.
A saját rutin beolvas egy karaktert a BASIC szövegből, és megvizsgálja, hogy az felkiáltójel-e?
Ha nem, akkor a CHRGOT rutinnal a kapcsoló eredeti értékét visszaállítjuk és az értelmezőnek arra a részére ugrunk, ahol az az utasításokat a szokott módon feldolgozza. Ha a karakter felkiáltójel, akkor beolvassuk a következő karaktert és megvizsgáljuk, hogy az megfelel-e a GOSUB utasításnak. Ha nem, akkor kiírjuk a SYNTAX ERROR hibaüzenetet.
Ha azonosítottuk a GOSUB utasítást, akkor beolvassuk a következő karaktert. Ha sor végét észlelünk, akkor elágazunk arra a programrészre, amely kikapcsolja a megszakításokat és a vektorok eredeti értékét helyreállítja.
Egyébként beolvassuk a megszakítási rutin sorszámát. Ellenőrizzük, hogy ez a sorszám valóban létezik-e (a C kapcsoló értéke alapján), majd a sor címének eggyel csökkentett értékét későbbi felhasználásra tároljuk. Ezután következik a két paramétert egymástól elválasztó vessző vizsgálata.
Végül betöltjük a második paramétert, amely a megszakítási időközök hosszát adja meg. Miután meggyőződtünk arról, hogy a beolvasott érték nem nulla, betöltjük a B számlálóba. Az A számlálóba betöltjük az ötvenedmásodpercnek megfelelő értéket, és mindkét számlálót elindítjuk.
A B számlálót úgy programozzuk, hogy értéke az A számláló minden lefutása után eggyel csökkenjen. Végül engedélyeztetjük a B számláló NMI típusú megszakítását úgy, hogy az ICR-be a megfelelő bitmintát írjuk be.
Végül a STOP és az NMI vektorokat az új rutin címére állítjuk és visszatérünk az értelmező – ciklushoz.
Az 1150-től 1250-ig terjedő sorokban található az a rutin, amely egy paraméter nélküli !GOSUB utasítás hatására kikapcsolja a megszakításokat, és mindent visszaállít az eredeti értékre.
A tényleges NMI rutin az 1270-től 1390-ig terjed. Mint általában minden megszakítási rutinban, itt is először tároljuk a regiszterek tartalmát, majd az ICR értékét megvizsgálva eldöntjük, hogy az NMI megszakítás forrása valóban a B számláló volt-e. Ha igen, egy kapcsoló értékét beállítjuk és visszatérünk a megszakításból, egyébként pedig elágazunk a valódi rutinra.
A legfontosabb szubrutin, amelyet az értelmezőből minden utasítás végrehajtá-

sa után hívunk – az 1410-es sorban kezdődik. Itt vizsgáljuk meg, hogy a kijelölt időtartam eltelt-e már, és a kitüntetett kapcsoló értékét az NMI rutin beállította-e. Ha az eredmény negatív, akkor a szokott módon ugrunk a STOP billentyű vizsgálatát végző rutinra. Ha azonban a kijelölt idő eltelt, a kapcsoló értékét töröljük, és a veremből visszaolvassuk a saját ugrási címet. Itt végezzük el mindazt, amit az értelmezőnek minden GOSUB utasítás hatására el kell végeznie. Megvizsgáljuk, hogy van-e még elég hely a veremben, majd a BASIC szöveg aktuális mutatóját és a sorszámot a verembe helyezzük. A FOR- NEXT utasítás végrehajtásától eltérően – amelynek paraméterei is bekerülnek a verembe –, itt csak a GOSUB kód kerül a verembe. Most betöltjük a szubrutin definíció szerint kiszámított és tárolt címet a BASIC szövegmutatóba, majd ismét elágazunk az értelmező-ciklushoz. Az értelmező végrehajtja a szubrutint, és a RETURN utasítás hatására visszatér a megszakított programba.
A program két változó értékétől függően érhet véget.A 120-as sorban található .SYM utasítás hatására kaphatunk a programbeli változók azonosítóit, ill. értékeit tartalmazó táblázatot.
Az elkészült gépi kódú programmal tetszőleges, a számláló által vezérelt alprogramot hívhatunk meg egy BASIC programból. Az időtartamot 20 ms és 21 min között tetszőlegesen megválaszthatjuk.
Példaként nézzük meg a P.37-es programot, amely a képernyőt villogtatja a háttér és a keret színének felcserélésével.

```
***       P37.       ***

100 SYS 52224
110 F1 = 53280 : F2 = F1 + 1
120 !GOSUB 1000,30
130 FOR I=1 TO 1000 : PRINT I, : NEXT
140 !GOSUB : END
1000 A=PEEK(F1) : POKE F1,PEEK(F2) : POKE F2,A : RETURN
```

A BASIC megszakítási rutint mindig a !GOSUB utasítással fejezzük be, amely helyreállítja a vektorok eredeti értékét. Ellenkező esetben ugyanis, ha pl. egy programot listázni vagy tárolni szeretnénk, az utasítás végrehajtását a saját készítésű rutin meg fogja szakítani.
A következő mintaprogram kiírja a Commodore 64-es teljes karakterkészletét normál és inverz alakban, majd minden megszakítási lépésben átkapcsol normál karakter üzemmódról kibővített színes üzemmódra és megfordítva. Az átkapcsolás a videovezérlő 17-es regiszterében a 6. bit módosításával történik. Kibővített üzemmódban az eredeti 256 helyett csak 64 karaktert lehet ábrázolni. A képernyőkódok így felszabaduló felső két bitje a karakterek háttérszínének kiválasztására szolgál, amelyeket egyébként a videovezérlő 33-tól 36-ig terjedő regisztereiben (az 53281-től 53284-ig terjedő címeken) találhatunk meg.

```
***       P38.       ***

100 SYS 52224
110 !GOSUB 170,25
120 X=18
130 PRINT CHR$(X); : X=X+128 AND 255
140 FOR I=32 TO 127 : PRINT CHR$(I); : NEXT
150 FOR I=160 TO 255 : PRINT CHR$(I); : NEXT
160 PRINT : GOTO 130
170 A=PEEK(53248+17) : POKE 53248+17,(A OR 64) ANDNOT (A AND 64)
180 RETURN
```

3.1 AZ OPERÁCIÓS RENDSZER ÉS A BASIC BŐVÍTÉSEI

A Commodore 64-es gép óriási előnye nagyobb testvéreihez, a CBM 2000, 3000, 4000 és 8000 gépekhez képest az, hogy szemben a többi géppel, a Commodore 64-es BASIC értelmezőjét és operációs rendszerét nagyon egyszerűen lehet saját elképzelésünk szerint működő gépi kódú rutinokkal kibővíteni úgy, hogy ráadásul ezek a rutinok az értelmezőbe vagy az operációs rendszerbe tökéletesen beilleszkedjenek.
A „beilleszkedés" alatt azt értjük, hogy a rutin inicializálása után az új lehetőséget egy új utasítással éppen olyan kényelmesen használhatjuk, mint a többi BASIC utasítást.
Nincs szükség arra, hogy az új utasítás minden végrehajtását PEEK, POKE esetleg SYS utasítással készítsük elő.
Mivel a Commodore 64-es a teljes 64 Kbyte RAM területet meg tudja címezni, a BASIC vagy az operációs rendszer módosítása mindössze annyiból áll, hogy a kívánt részt a ROM-ból a RAM azonos címtartományába átmásoljuk, ott tetszőlegesen megváltoztatjuk, majd az 1-es címen található porton keresztül átkapcsolunk a RAM-ra. A módszernek előnyei mellett természetesen van némi hátránya is.
Az előnye elsősorban abban rejlik, hogy a módosítási lehetőségek száma korlátlan. Ez azt jelenti, hogy a BASIC helyett akár egy teljesen új programozási nyelvet is kialakíthatunk, és az operációs rendszert is teljesen átírhatjuk. Például, ha úgy tetszik, a teljes RAM területet a grafikus ábrázolásmód szolgálatába állíthatjuk. A módszer hátránya az, hogy eközben elvész számunkra a RAM tartomány.
A módosítások egyik változata az, hogy a $8000-től $9FFF-ig vagy a $8000-től $BFFF-ig terjedő területre elhelyezünk egy vagy két olyan EPROM-ot, amely tartalmaz egy BASIC bővítést, egy új nyelvet, vagy egy, a felhasználó által meghatározott programot.Ehhez a megoldáshoz szükség van egy, a bővítő portra (expansion port) illeszthető kártyára.
A második megoldáshoz nincs szükség kiegészítő ROM-ra, csak arra, hogy az új módosított függvényeknek előre meg tudjuk határozni a beugrási pontjait. Ebben az esetben főszerepet kapnak az ún. beugrási vektorok, amelyeket a használó könnyen megváltoztathat. Az ilyen változtatásokat megoldhatjuk olyan indirekt ugróutasítással, mint pl.

JMP (VEKTOR)

A VEKTOR címen található a tényleges ugrási cím alsó és felső byte-ja. A gép bekapcsoláskor mindig újraállítja (inicializálja) ezeket a vektorokat és ezek a BASIC értelmező esetében mindig közvetlenül az ugróutasítások mögé mutatnak.
Amikor egy függvényt módosítunk, úgy kell eljárnunk, hogy először megírjuk a saját új függvényrutinunkat, majd az eredeti függvényhez tartozó ugróvektort átírjuk úgy, hogy az új függvényre mutasson. Az alapelv ugyanaz, mint amit már megismertünk a megszakítási vektorok esetében.

A „RAM – módszer" alkalmazásához közöljük az alábbi táblázatot, amely tartalmazza azokat a bitmintákat, amelyeket az 1-es címre be kell írni ahhoz, hogy a megfelelő tárkonfigurációt kiválaszthassuk.

*** T46/3. ***

Bit 2	1	0	Dec.	$A000-$BFFF	$D000-$DFFF	$E000-$FFFF
1	1	1	7	BASIC	I/O	KERNAL
1	1	0	6	RAM	I/O	KERNAL
1	0	1	5	RAM	I/O	RAM
1	0	0	4	RAM	RAM	RAM
0	1	1	3	BASIC	CHAR-GEN	KERNAL
0	1	0	2	RAM	CHAR-GEN	KERNAL
0	0	1	1	RAM	CHAR-GEN	RAM
0	0	0	0	RAM	RAM	RAM

A táblázat minden olyan kombinációt tartalmaz, amelyeket a programból közvetlenül elő lehet állítani. A negyedik és a nyolcadik kombináció azonos eredményt ad, azaz a teljes címezhető területet átkapcsolja a RAM-ra. A táblázatból kitűnik, hogy akár a teljes BASIC ROM-ot is kicserélhetjük a RAM-mal, de a KERNAL ROM-ot csak a BASIC-kel együtt lehet kicserélni. Erre feltétlenül ügyelnünk kell, amikor az operációs rendszert akarjuk helyettesíteni. A $D000 – $DFFF terület háromszorosan foglalt; egyrészt itt található az I/O tartomány a következő tagolással:

*** T46/4. ***

$D000	– $D3FF	VIC 6569
$D400	– $D7FF	SID 6581
$D800	– $DBFF	Színram
$DC00	– $DCFF	CIA 1 6526
$DD00	– $DDFF	CIA 2 6526
$DE00	– $DEFF	I/O 1 bővítésekhez
$DF00	– $DFFF	I/O 2 bővítésekhez

Másrészt ezen a területen található a karaktergenerátor. Végül ez a terület a RAM-hoz tartozik abban az értelemben, hogy csak akkor érhetjük el, amikor a teljes tárat a RAM-ra kapcsoltuk át.

3.2 A BASIC VEKTOROK

A BASIC értelmező hat olyan vektorral rendelkezik, amelyek kapcsolódási pontot jelenthetnek a saját készítésű rutinokhoz. Ez a hat vektor a 3-as lapon található és jelentésük az alábbi:

```
***        T46/5.        ***

  Vektor        Cim        Jelentes
-------------------------------------------------------------------
$0300/$0301   $E38B     BASIC melegstart es hibarutin beugr.pont
$0302/$0303   $A438     Beviteli varakozo ciklus
$0304/$0305   $A57C     Atalakitas interpreterkodda
$0306/$0307   $A71A     Interpreterkod atalakitasa szovegge
$0308/$0309   $A7E4     A BASIC utasitas vegrehajtasa
$030A/$030B   $AE86     A BASIC kifejezes kiertekelese
```

A hat vektor segítségével a BASIC értelmezőt messzemenően átalakíthatjuk. Az alábbiakban mindegyiket részletesen ismertetjük és példát adunk a felhasználási lehetőségekre.
Ha az Olvasó rendelkezik a „Commodore 64-es belső felépítése" c. könyvvel, érdemes ezt a fejezetet a gép ROM listájával párhuzamosan tanulmányoznia, az eredmény biztosan erőteljesebb lesz.

A melegstart- és hibavektor $300/$301

Mind az END utasítás végrehajtása, mind pedig egy esetleges hiba fellépte olyan ugrást eredményez, amelynek ez a vektor a kiindulópontja. Ha hiba lépett fel, akkor az X regiszter tartalmazza a hiba számát. A számok jelentése 1-től 29-ig:

```
***        T46/6.        ***

Ssz.   Hibauzenet
-------------------------------------------------------------------
 1 TOO MANY FILES                - Tul sok file
 2 FILE OPEN                     - A file nyitva van
 3 FILE NOT OPEN                 - A file nincs nyitva
 4 FILE NOT FOUND                - A file nem letezik
 5 DEVICE NOT PRESENT            - Az egyseg nincs jelen
 6 NOT INPUT FILE                - Nem INPUT file
 7 NOT OUTPUT FILE               - Nem OUTPUT file
 8 MISSING FILENAME              - Hianyzo file-nev
 9 ILLEGAL DEVICE NUMBER         - Illegalis egysegszam
10 NEXT WITHOUT FOR              - NEXT FOR nelkul
11 SYNTAX                        - Szintaktikus
12 RETURN WITHOUT GOSUB          - RETURN GOSUB nelkul
13 OUT OF DATA                   - Nincs tobb DATA
14 ILLEGAL QUANTITY              - Illegalis mennyiseg
15 OVERFLOW                      - Tulcsordulas
16 OUT OF MEMORY                 - Betelt a tarterulet
17 UNDEF'D STATEMENT             - Hatarozatlan utasitas
18 BAD SUBSCRIPT                 - Illegalis index
19 REDIM'D ARRAY                 - Ujradimenzionalas
20 DIVISION BY ZERO              - Osztas nullaval
```

```
21 ILLEGAL DIRECT          - Parancs uzemmodban nem hasznalhato
22 TYPE MISMATCH           - Tipus keveredes
23 STRING TOO LONG         - A szoveg tul hosszu
24 FILE DATA               - Adathiba a file-ban
25 FORMULA TOO COMPLEX     - A kifejezes tul osszetett
26 CAN'T CONTINUE          - Folytathatatlan
27 UNDEF'D FUNCTION        - Hatarozatlan fuggveny
28 VERIFY                  - Ellenorzes
29 LOAD                    - Betoltes
```

Az 1-től 9-ig terjedő hibák a beolvasással és a kiírással kapcsolatosak, ezeket az üzeneteket az operációs rendszer küldi, míg a 10-től 29-ig terjedő üzenetek a BASIC értelmezőtől származnak.
Ha az értelmező felismer egy hibát, a hibaszámot elhelyezi az X regiszterben, és ugrik a $A437-es címre, ahol az indirekt JMP ($0300) utasítás áll. Ha az ugrást nem hiba, hanem az END utasítás váltotta ki, akkor hibaszám helyett egy negatív érték ($80) található az X regiszterben. A hibarutin megvizsgálja az X regiszter tartalmát, és annak értéke szerint vagy egy hibaüzenetet, vagy a READY üzenetet írja ki, majd elágazik a beolvasási várakozó ciklusra.
A hibavektort különböző célokra hasznosíthatjuk.Egyfelől megváltoztathatjuk a hibaüzenetek szövegét, lefordíthatjuk pl. magyarra.
A másik, sokkal érdekesebb lehetőség az, hogy egy hiba bekövetkezésekor nem engedjük, hogy a program futása megszakadjon, hanem ehelyett elágazunk egy meghatározott BASIC sorra, ahol a hibát valamilyen módon korrigáljuk. Egy ilyen utasítást elnevezhetünk pl.

ON ERROR GOTO...

utasításnak, hiszen ez a név feltehetően sokak számára ismerős más programnyelvekből.
Az új utasítással kezelhetjük pl. azokat a hibákat, amelyeket egy külső egység váltott ki.

A beolvasási várakozó ciklus $302/$303

Amikor a gép a hiba-, ill. melegstart vektoron keresztül kiírt egy hibaüzenetet vagy a READY szöveget, akkor a $302/$303 vektoron keresztül elágazik egy másik rutinra. Itt addig várakozik, amíg egy sor beadását egy RETURN le nem zárta. Közben figyeli, hogy a sor hossza nem haladta-e meg a 88 karaktert, amennyi a beviteli puffer területe (a $200-tól a $258-ig). Ha igen, akkor kiírja a STRING TOO LONG üzenetet. A beírt sor első karaktere határozza meg a sor feldolgozásának módját. Ha az első karakter számjegy, akkor az értelmező tudja, hogy egy új BASIC sort gépeltünk be. Ha a számjegy(ek) után semmi nem következik, akkor a sort törli a programból, ezzel ezt a munkafázist befejezi és visszatér a ciklus elejére. Ha a sorszámot szöveg követi, akkor azt átalakítja értelmező kódokká és a programsort beilleszti a programszövegbe, majd ismét visszaugrik a várakozó ciklus elejére.
Ha az első karakter nem számjegy volt, akkor a sort mint BASIC utasítást közvetlen módban értelmezi és végrehajtja. A megfelelő értelmező kódokat kialakítva elugrik arra a rutinra, amelynek feladata a BASIC utasítás végrehajtása. Ezt a vektort is fel tudjuk használni saját céljainkra. Tegyük fel például,

hogy az adatok nem a billentyűzetről, hanem egy lemezen tárolt soros file-ból, vagy a user porton keresztül egy másik számítógépből érkeznek. Megoldhatjuk, hogy az egyik gépen elkészült programot ne kelljen sok munkával és esetleg sok hibával újra begépelni, hanem átvehessük közvetlenül a géphez csatlakoztatott külső egységről. A közvetlen csatlakoztatásnál az adatokat küldő egységnek szüksége van egy illesztőre, ez lehet pl. egy beépített RS 232-es illesztő, amely a legtöbb típusú gépre alkalmazható.
Egy másik lehetőség pl. az AUTO utasítás megvalósítása. Ez az utasítás megkönnyíti a programok begépelését azzal, hogy minden BASIC sor begépelése után automatikusan adja a következő sorszámot, és a kurzort a következő sor elejére pozicionálja.

Átalakítás értelmező kódokká $304/$305

Az Olvasó biztosan tudja, hogy a programsort a gép nem abban a formában tárolja, ahogyan azt begépeltük, hanem minden utasítást lerövidít egy egybyteos értékké. Ennek a tárolási módszernek két előnye van. Az egyik a takarékos helykihasználás, hiszen pl. a PRINT szó tárolására öt byte helyett csak egy byte-ra van szükség. A másik előny az utasítások végrehajtása során mutatkozik meg. Amikor az értelmező egy program feldolgozása közben egy értelmező kódot talál, azonnal végre tudja hajtani a megfelelő utasítást. Ha kód helyett teljes szavakkal kellene dolgoznia, akkor be kellene olvasnia a teljes szót, azután ellenőriznie kellene, hogy a szó létezik-e a szótárában. A program futását ezek a tevékenységek nagyon lelassítanák.
Ugyanakkor a kódolást csak egyszer kell elvégezni (begépeléskor), nem pedig minden sor végrehajtása közben.
Egyébként is begépelés közben a kódolás sokkal kevésbé időigényes, mint maga az adatbevitel, a legtöbb idő a várakozással telik el.
Ha egy új utasítást értelmező kóddá akarunk alakítani, akkor ezt a vektort kell megváltoztatnunk.
Az új rutinnak a beolvasott szót az új utasítástáblázat szavaival kell összehasonlítania. Ha az új szót sikerült a táblázatban megtalálni, a beviteli sorba beírhatjuk helyette a kódot.

Az interpreter kódok visszaalakítása szöveggé $306/$307

Ennek a vektornak az előzővel éppen ellentétes feladata van. Amikor egy programot listázunk, a kódot vissza kell alakítani szövegekké.
A visszaalakításhoz a kódot az utasítástáblázat mutatójaként használjuk. Ez a vektor LIST vektor néven is ismert. Természetesen ezt a vektort is meg kell változtatnunk, ha új értelmező kódokat használunk.
Egy másik felhasználási lehetőség a LIST utasítás módosítása. Pl. a jobb olvashatóság kedvéért minden utasítás után beszúrhatunk egy szóköz karaktert a programlistába, vagy például a kettőspontokkal elválasztott utasításokat külön sorba írhatjuk stb.

Egy BASIC utasítás végrehajtása $308/$309

Minden kétséget kizáróan ez az egyik legfontosabb vektor. Az értelmezőnek arra a részére mutat, ahol az a BASIC utasításokat végrehajtja. A rutin először betölt egy karaktert a BASIC szövegből és megvizsgálja, hogy az értelmező kód vagy sem. A kódokat gyakran *token*nek is nevezzük.
Ha a karakter nem token, akkor az értelmező megpróbálja a sort "A = ..." értékadó utasításként kezelni és elágazik a LET utasításhoz.
Egyébként a tokent annak a táblázatnak az indexeként használja, amely az utasítások címeit tartalmazza. Az utasítást a megfelelő szubrutinban végrehajtja és visszatér az ún. értelmező ciklus elejére, ahol a következő utasítást hasonló módon kezdi feldolgozni. A vektor segítségével tetszőleges saját BASIC utasítást illeszthetünk a meglévő utasítások közé. Az új utasítást valamilyen karakterrel, pl. egy felkiáltójellel meg kell különböztetni az eredeti BASIC utasításoktól. A saját rutinunk erről a karakterről ismerheti majd fel a különleges utasítást.
Ha azonban a $304/$305 vektort is módosítjuk, és ezáltal az új utasításhoz egy saját értelmező kódot rendelünk, akkor nincs szükség a megkülönböztető karakterre. Így a kódot magát megvizsgálva dönthetünk arról, hogy az eredeti, vagy az új rutinra kell elágazni.

Egy BASIC kifejezés kiértékelése $30A/$30B

Míg a fenti vektor a BASIC utasítások, addig ez a BASIC függvények vektora. Az értelmező ezt a vektort akkor használja, amikor egy kifejezés valamely elemét pontosan ki akarja számítani. Ez az elem lehet egy szám, egy változó vagy egy BASIC függvény.
Ha a meglévők közé egy új függvényt szeretnénk beilleszteni, ezt a vektort kell módosítanunk. Az eljárás vonatkozhat numerikus és füzér függvényre egyaránt. Akkor is ezen a ponton kell az értelmezőt módosítanunk, ha a megszokottól eltérően akarjuk a változókat tárolni, vagy új módszerrel (pl. hexadecimális alakban) szeretnénk adatokat beolvastatni.

```
***      P39.      ***

PROFI-ASS 64 V2.0

100:   033C                    .OPT P1,00
110:                    ;
120:                    ; A hexadecimalis es binaris szamok beolvasasa
130:                    ;
140:   030A             AUSRUCK   =   $30A      ; Kifejezest kiertekelo vektor
150:   AE8D             VEKTALT   =   $AE8D     ; A regi rutin
160:                    ;
170:   000D             TYP       =   13        ; A valtozo tipusa
180:   0073             CHRGET    =   $73
190:   0079             CHRGOT    =   CHRGET+6
200:                    ;
210:   BD7E             ADDZIFFER =   $BD7E     ; Egy byte-os szam + FAC
220:                    ;
230:   005D             FLOAT     =   $5D       ; Valos szamok tartomanya
240:   0061             EXP       =   $61       ; A FAC kitevoje
250:                    ;
260:   B97E             OVERFLOW  =   $B97E     ; OVERFLOW ERROR
```

```
270:                    ;
280:  033C              *=  828
290:                    ;
300:  033C A9 47  INIT  LDA #< TEST
310:  033E A0 03        LDY #> TEST
320:  0340 8D 0A 03     STA AUSDRUCK  ; A vektor az uj rutinhoz
330:  0343 8C 0B 03     STY AUSDRUCK+1
340:  0346 60           RTS
350:                    ;
360:  0347 A9 00  TEST  LDA #0
370:  0349 85 0D        STA TYP.      ; A tipuskapcsolo numerikus
380:  034B 20 73 00     JSR CHRGET    ; A kovetkezo kar. beolvasasa
390:  034E C9 24        CMP #"$"      ; Hexadecimalis ?
400:  0350 F0 0A        BEQ HEXZAHL
410:  0352 C9 25        CMP #"%"      ; Binaris jegy ?
420:  0354 F0 41        BEQ BINZAHL
430:                    ;
440:  0356 20 79 00     JSR CHRGOT    ; A kapcsolo vissza
450:  0359 4C BD AE     JMP VEKTALT   ; es ugras a regi kiertekelesre
460:                    ;
470:  035C 20 8D 03 HEXZAHL JSR CLRFAC  ; A FAC torlese
480:  035F 20 73 00 GETNEXT JSR CHRGET  ; A kovetkezo karakter betoltese
490:  0362 90 0B        BCC ZIFFER    ; Szamjegy ?
500:  0364 C9 41        CMP #"A"
510:  0366 90 1F        BCC END       ; Kisebb mint "A" ?
520:  0368 C9 47        CMP #"F"+1
530:  036A B0 1B        BCS END       ; Nagyobb mint "F" ?
540:  036C 38           SEC
550:  036D E9 07        SBC #7        ; Az OFFSET figyelembevetele
560:  036F 38     ZIFFER SEC
560:  0370 E9 30        SBC #"0"      ; Atvaltas hexadecimalisra
570:  0372 48           PHA           ; A karakter tarolasa
580:  0373 A5 61        LDA EXP
580:  0375 F0 07        BEQ NOCHNULL  ; A FAC = 0 ?
590:  0377 18           CLC
600:  0378 69 04        ADC #4        ; Kitevo + 4 => szam * 16
610:  037A B0 0E        BCS OVER      ; A szam tul nagy ?
620:  037C 85 61        STA EXP
630:  037E 68     NOCHNULL PLA        ; A szamjegy visszatoltese
640:  037F F0 DE        BEQ GETNEXT   ; Ha 0, => hozzaadas felesleges
650:  0381 20 7E BD     JSR ADDZIFFER ; A szamjegy hozzaadasa FAC-hoz
660:  0384 4C 5F 03     JMP GETNEXT
670:                    ;
680:  0387 4C 79 00 END JMP CHRGOT
690:                    ;
700:  038A 4C 7E B9 OVER JMP OVERFLOW
710:                    ;
720:  038D A9 00  CLRFAC LDA #0
730:  038F A2 0A        LDX #10
740:  0391 95 5D  LOOP  STA FLOAT,X   ; A valos tartomany torlese
750:  0393 CA           DEX
760:  0394 10 FB        BPL LOOP
770:  0396 60           RTS
780:                    ;
790:  0397 20 8D 03 BINZAHL JSR CLRFAC  ; A FAC torlese
800:  039A 20 73 00 GETBIN JSR CHRGET   ; A kovetkezo karakter betoltese
810:  039D C9 32        CMP #"2"
820:  039F B0 E6        BCS END       ; Nagyobb mint 1 ?
830:  03A1 C9 30        CMP #"0"
840:  03A3 90 E2        BCC END       ; Kisebb mint 0 ?
850:  03A5 E9 30        SBC #"0"      ; ASCII-rol hexadecimalisra
860:  03A7 48           PHA
870:  03A8 A5 61        LDA EXP       ; A szam meg 0 ?
880:  03AA F0 04        BEQ NULL
890:  03AC E6 61        INC EXP       ; Duplazni !
900:  03AE F0 DA        BEQ OVER      ; Tul nagy ?
910:  03B0 68     NULL  PLA
920:  03B1 F0 E7        BEQ GETBIN    ; A 0-t nem kell hozzaadni
930:  03B3 20 7E BD     JSR ADDZIFFER ; A szamjegy hozzaadasa
940:  03B6 4C 9A 03     JMP GETBIN    ; A kovetkezo kar. betoltese
]033C-03B9
NO ERRORS
```

A rutin a decimális számjegyeket feldolgozó rutinhoz hasonlóan dolgozik, de annál egyszerűbb és áttekinthetőbb, mivel nem kell tizedes törtekkel és kitevőkkel bajlódnia. Ha a programot SYS 828 utasítással lefuttatjuk, a decimális írásmódhoz hasonlóan hexadecimális, ill. bináris számokat is begépelhetünk. Pl. a

```
?$FFFF
```

utasítás eredménye: 65535, a

```
?%101010
```

utasítás eredménye pedig 42.
A beírt hexadecimális szám nem csak négyjegyű lehet. A program úgy van megírva, hogy tetszőleges, a gép által kezelhető lebegőpontos számtartományba eső hexadecimális számokat fel tud dolgozni, azaz olyanokat, amelyek maximálisan 31 hexadecimális számjegyet tartalmaznak. Pl:

```
?$FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
```

Az eredmény:

```
2.12676479E+37
```

A teljes lebegőpontos számtartományt bináris számokkal nem tudjuk kihasználni, hiszen még egy 78 jegyű szám decimális értéke is csak 3E+23.
A bővítést nemcsak a PRINT utasításokban lehet alkalmazni. A programban ezután minden olyan helyre, ahová eddig decimális számot írtunk, írhatunk bináris vagy hexadecimális számokat. Különösen érdekes lehet ez a POKE és PEEK, ill. a SYS utasításokban. Könnyebb megjegyezni például a videovezérlő kezdőcímét hexadecimális ($D000) alakban, mint decimálisan (53248).
Pl.: A 3-as sprite bekapcsolása az új módszerrel:

```
POKE $D015, PEEK($D015) OR %1000
```

Az utasítás régi alakjából sokkal kevésbé szembetűnő a dolog logikája.

```
POKE 53248+21, PEEK(53248+21) OR 8
```

Van azonban az új módszernek egy furcsa sajátossága. Az alábbi utasítás

```
?$ABCDEF
```

hatására hibajelzést kapunk. Vajon miért? Ha a hexadecimális számot alaposabban szemügyre vesszük, felismerjük benne a DEF utasításszót. Mivel az értelmező először minden lehetséges szöveget átalakít kódokká, a DEF szóból is kódot képez, amit természetesen a saját gépi kódú rutinunk már nem tud

értelmezni. Ezt a hibalehetőséget egyszerűen kiküszöbölhetjük, ha az esetlegesen előforduló utasításszavak közé egy szóköz karaktert írunk, pl:

```
?$ABCD EF
```

Így megkapjuk a helyes 11259375 decimális értéket. A CHRGET rutin a szöveg közbeni szóköz karaktert éppúgy átlépi, mint a számjegyek közötti szóközt. Gondoljuk végig a program működésének logikáját.
Első lépés a szokásos inicializálás, amelyben a vektort a saját rutinunkra állítjuk. Ezután az értelmező rutinjaihoz hasonlóan töröljük a változótípust jelölő kapcsolót (a nulla érték numerikus változóra utal). Ezután beolvassuk a következő karaktert. Ha ez dollár vagy felkiáltójel, elágazunk az új rutinra. Egyébként a CHRGOT utasítással visszaállítjuk a kapcsolót, és visszatérünk az értelmező eredeti kiértékelő rutinjára.
Az új rutin az alábbi programlépéseket végzi el:
Először törli a lebegőpontos akkumulátort, az eredményt ugyanis ide fogja beírni. Ezután beolvassa a következő karaktert és megvizsgálja, hogy az karakterjegy, vagy egy A és F közé eső betű. Ha ez a feltétel teljesül, a karaktert átváltja a megfelelő hexadecimális alakra, tehát pl. az "1"-ből $01, az "A"-ból $0A lesz. Most megszorozza a lebegőpontos akkumulátor tartalmát 16-tal, ha az nem nulla. A szorzással érjük el, hogy a számjegyek helyiértéke megfelelő legyen. A szorzást nagyon egyszerűen végezzük el. Az igazi lebegőpontos szorzást végző rutin hívása helyett megnöveljük a 2-re vonatkozó kitevőt 4-gyel.
Az összeadás jóval kevesebb ideig tart, mint a szorzás. Miután ellenőriztük, hogy az összeadásnál nem keletkezett átvitel, ismét visszatöltjük az előbb leolvasott számjegyet és hozzáadjuk az FAC-hez. Ha a számjegy 0, akkor az összeadást nem kell elvégezni. Ez a ciklus mindaddig ismétlődik, amíg a CHRGET rutin a számhoz tartozó karaktert olvas be a szövegből.
A bináris számokat a fentiekhez hasonlóan dolgozzuk fel. Az eljárást azonban lényegesen egyszerűsíti, hogy a 2-vel való szorzás nem egyéb, mint a kitevő növelése eggyel. A továbbiakban semmi eltérés nincs a hexadecimális és a bináris számok kezelése között.

3.3 A STRUKTURÁLT PROGRAMOZÁS

Ebben a könyvben már nagyon sok szó esett a BASIC értelmező működéséről, és különösen hangsúlyozottan tárgyaltuk a BASIC utasításokat végrehajtó interpreter részt. Arról azonban még egyáltalán nem esett szó, hogy hogyan kezeli az értelmező a programstruktúrákat. A strukturált programozást az értelmező két utasítással támogatja; az egyik a

GOSUB... RETURN

a másik a

FOR... NEXT

utasítás.

Az ilyen szerkezetű utasítások végrehajtása közben az értelmezőnek tudnia kell, hogy pl. a RETURN hatására a program melyik utasítására kell visszatérnie, azaz a programnak mely részéről hívták a szubrutint.
A NEXT utasítás végrehajtásához tárolni kell a ciklusmag első utasításának címén kívül a ciklus végértékét és a növekményt is, hiszen minden lépésben dönteni kell arról, hogy folytatódik vagy véget ér-e a ciklus.
A RETURN és NEXT utasítások szükséges paramétereit elhelyezzük a tár egy rögzített területére.
De mi történik akkor, ha több ciklust vagy több szubrutint skatulyázunk egymásba?
Minden RETURN és NEXT utasítás számára lehetővé kell tenni, hogy hozzáférjen az utolsó szerkezeti egységből származó paraméterekhez. Az elvet már ismerjük a veremmel kapcsolatban:
– amit utoljára tároltunk, azt vesszük legelőször elő (LAST IN – FIRST OUT). Ez a tárolási mód tökéletesen megfelel a fenti szerkezetű egymásba ágyazott rutinok, ill. ciklusok egymás utáni végrehajtására.

Milyen paraméterekre van szüksége a GOSUB utasításnak?
Először is annak az utasításnak a címére, ahonnan a hívás érkezett. Másrészt az aktuális sorszámra, hiszen visszatérés után ennek is vissza kell állítani az eredeti értékét. Végül azért, hogy meg lehessen különböztetni a GOSUB utasítás verembeli adatait a FOR ciklus adataitól, az értelmező elhelyezi a verembe a GOSUB kódját is. A verem teljes tartalma tehát az alábbi:

Veremmutató	
a GOSUB előtt a	programmutató felső byte-ja
	programmutató alsó byte-ja
	sorszám felső byte-ja
	sorszám alsó byte-ja
	GOSUB kód $8D byte-ja

Veremmutató

a GOSUB után

A GOSUB utasítás paraméterei öt byte-ot foglalnak el a veremben. Mivel a 6510-es processzor veremmutatója nyolc bites, a verem mérete maximum 1 lapnyi lehet: a $100-tól $1FF-ig. A korlátozott veremméret miatt nem lehet tetszőleges mélységben rutinokat és ciklusokat egymásba skatulyázni. Maximálisan 256/5 = 51 szubrutin paraméterei férnének el a veremben, ha az értelmező semmi mást nem tárolna itt. Mivel azonban ez nem így van, a verem tényleges kapacitása ennél valamivel kevesebb.
Egy GOSUB utasítás végrehajtása előtt az értelmező mindig megvizsgálja, hogy a veremben van-e még elegendő hely a paraméterek számára. A vizsgálatot elvégző rutin hívása előtt az akkumulátor tartalma a szükséges helyek számának fele. Például, ha az akkumulátor tartalma 3, akkor a veremben 6 szabad tárcím szükséges a paraméterek számára.
Ha már nincs annyi hely a veremben, akkor az OUT OF MEMORY (kevés a tárterület) hibaüzenetet kapjuk, ami nem egészen felel meg a tényleges helyzetnek, hiszen a gép akkor is ezt a hibaüzenetet küldi, ha egy változó elhelyezésére már nem talál szabad tárterületet. A helyes üzenet a STACK OVERFLOW (verem túlcsordulás) lenne.
A BASIC értelmező csak a $013E-től $1FA-ig terjedő területet használja veremként. A $0100-tól $0110-ig terjedő terület a lebegőpontos számok füzérré alakításának munkaterülete, a $0100-tól $013E-ig terjedő területet pedig a szalagos olvasásnál bekövetkező hibák kezelése közben használja az értelmező.
Hogyan hajtja végre az értelmező a RETURN utasítást? Először megvizsgálja, hogy a verem legtetején GOSUB kód van-e.
Ha nem, akkor kiírja a RETURN WITHOUT GOSUB (GOSUB nélküli RETURN) hibaüzenetet. Egyébként beolvassa a következő négy byte tartalmát, és ezekből visszaállítja a programmutató és a sorszám értékét. A veremmutatót visszaállítja arra a címre, ahová az a GOSUB utasítás végrehajtása előtt mutatott. Ezután visszaugrik az értelmező ciklusra, amely a program végrehajtását automatikusan a GOSUB utasítást követő utasításon folytatja.

A FOR – NEXT ciklus végrehajtásának elve teljesen azonos az előzőekkel, de annál egy kicsit bonyolultabb, hiszen több paraméter átmeneti tárolására van szükség.

Veremmutató

a FOR utasítás előtt a
- programmutató felső byte-ja
- programmutató alsó byte-ja
- sorszám felső byte-ja
- sorszám alsó byte-ja
- 4. mantissza
- 3. mantissza
- 2. mantissza
- 1. mantissza
- Kitevő

} a ciklusváltozó végértéke

Előjel
4. mantissza
3. mantissza } a lépésköz
2. mantissza
1. mantissza
Kitevő
a ciklusváltozó felő byte-ja
a ciklusváltozó alsó byte-ja
a FOR kódja $81

Veremmutató a

FOR utasítás után

A fenti táblázat alapján látjuk, hogy a FOR-NEXT ciklushoz a veremben 18 byte-ra van szükség.
A végrehajtás lépései: Az értelmező először megvizsgálja, hogy a verem legfelső byte-ja FOR kód-e. Ha nem, akkor kiírja a NEXT WITHOUT FOR (FOR nélküli NEXT) hibaüzenetet. Egyébként megvizsgálja, hogy a NEXT utasítás után van-e változónév. Ha igen, akkor megkeresi a változó címét és összehasonlítja a verembeli változó-címmel. Ha a két cím egyenlő (vagy ha nincs a NEXT után változó), akkor beolvassa a változó értékét az FAC-be, és ehhez hozzáadja a veremben tárolt lépésközt. A kapott értéket összehasonlítja a ciklusváltozó végértékével. Az eredménytől függ, hogy a ciklus véget ér vagy folytatódik. Ha a ciklus lefutott, akkor megnöveli a veremmutató értékét 18-cal, ezzel eltávolítva a veremből a ciklus paramétereit, és visszatér az értelmező ciklushoz, ahol folytatódik a program végrehajtása. Ha a ciklusváltozó értéke még nem érte el a végértéket, akkor betölti a veremben tárolt programmutatót és sorszámot, a veremmutató értékét pedig változatlanul hagyja, hiszen a paraméterekre a következő NEXT utasításnál ismét szüksége lesz.
Ha a NEXT utasításban van változónév, de a változó címe nem egyezik meg a veremben tárolt címmel, akkor az értelmező megnöveli a veremmutatót 18-cal és megvizsgálja, hogy ott szintén ciklusparaméterek vannak-e. A belső ciklus minden ilyen lépésben automatikusan lezáródik.
Eddigi ismereteink elegendőek ahhoz, hogy új programszerkezetet építsünk az értelmezőbe. Azok az Olvasók, akik már programoztak PASCAL nyelven, biztosan ismerik a REPEAT...UNTIL típusú ciklust.
A ciklus ismétlődését ebben a programszerkezetben egy feltétel vezérli. A ciklusmag végrehajtása mindaddig ismétlődik, amíg a feltétel teljesül. Pl.

```
REPEAT
I = I+1
UNTIL I = 10
```

A ciklusmag I = I+1 utasítását mindaddig végrehajtjuk, amíg az I értéke el nem éri a 10-et.
A FOR – NEXT ciklushoz hasonlóan az ilyen szerkezetű ciklus is lefut legalább egyszer a feltétel teljesülésétől függetlenül.
Beépítése sok esetben egyszerűsíti a programírást.

Pl. a billetyűre várakozás az új utasítással:

```
REPEAT : GET A$ : UNTIL A$<>""
```

vagy még egyszerűbben

```
REPEAT : UNTIL PEEK (197) <>64
```

A 197-es tárcím tartalma ugyanis mindaddig 64, amíg le nem ütünk egy billentyűt.
A P.40-es gépi kódú program beépíti az értelmezőbe a REPEAT...UNTIL szerkezetű ciklusutasítást.

```
***      P40.      ***
PROFI-ASS 64 V2.0

100:  033C                    .OPT P,00
110:  033C                    .TIT "REPEAT - UNTIL"
110:                ;
120:                ; REPEAT - UNTIL - CIKLUS
130:                ;
140:  0308          UTAS      =    $308      ; Az utasitas vegreh. vektora
150:  A7E7          REGIUT    =    $A7E7     ; A regi rutin
160:                ;
170:  0022          ADR       =    $22       ; A hibauzenet cime
180:  0039          SORSZ     =    $39       ; Aktualis sorszam
190:  0073          CHRGET    =    $73
200:  0079          CHRGOT    =    CHRGET+6
210:  007A          TXTPTR    =    CHRGOT+1
220:  0100          VEREM     =    $100      ; Processzor verem
230:  A445          ERROR     =    $A445     ; A hibauzenet kiirasa
240:                ;
250:  A3FB          VEREMELL  =    $A3FB     ; A szabad hely vizsgalata
260:  ADBA          FRMNUM    =    $ADBA     ; Numerikus kifejezes betoltese
270:  A7AE          INTER     =    $A7AE     ; Az interpreterciklus
280:  AF08          SYNTAX    =    $AF08     ; SYNTAX ERROR
290:  A906          NEXTUT    =    $A906     ; A kovetkezo utasitas keresese
300:                ;
310:  033C                    *=   828
320:  033C A9 47    INIT      LDA  #< TEST
330:  033E A0 03              LDY  #> TEST
340:  0340 8D 08 03           STA  UTAS       ; Vektor az uj rutinra
350:  0343 8C 09 03           STY  UTAS+1
360:  0346 60                 RTS
370:                ;
380:  0347 20 73 00 TEST      JSR  CHRGET     ; Kovetkezo karakter beolvasasa
390:  034A C9 21              CMP  #"!"
400:  034C F0 06              BEQ  NEWBEF     ; Uj utasitas ?
410:                ;
420:  034E 20 79 00           JSR  CHRGOT     ; A kapcsolo visszaallitasa
430:  0351 4C E7 A7           JMP  REGIUT     ; es a regi utasitas vegrehajt.
440:                ;
450:  0354 20 73 00 NEWBEF    JSR  CHRGET     ; A kovetkezo karakter
460:  0357 C9 52              CMP  #"R"       ; Repeat-utasitas ?
470:  0359 F0 07              BEQ  REPEAT
480:  035B C9 55              CMP  #"U"       ; Until-utasitas ?
490:  035D F0 24              BEQ  UNTIL
500:  035F 4C 08 AF SYNERR    JMP  SYNTAX     ; Egyebkent SYNTAX ERROR
510:                ;
520:  0362 20 73 00 REPEAT    JSR  CHRGET     ; Mutato a kovetkezo karakterre
530:  0365 A9 03              LDA  #3
540:  0367 20 FB A3           JSR  VEREMELL   ; Van eleg hely a veremben ?
550:  036A 20 06 A9           JSR  NEXTUT     ; A kovetkezo utasitas keresese
560:  036D 18                 CLC
570:  036E 98                 TYA             ; A kovetkezo utasitas offset-je
580:  036F 65 7A              ADC  TXTPTR     ; Osszeadas es
590:  0371 48                 PHA             ; tarolas a veremben
```

```
600:   0372 A5 7B          LDA  TXTPTR+1
610:   0374 69 00          ADC  #0
620:   0376 48             PHA
630:   0377 A5 39          LDA  SORSZ       ; Sorszam
640:   0379 48             PHA              ; szinten a verembe
650:   037A A5 3A          LDA  SORSZ+1
660:   037C 48             PHA
670:   037D A9 52          LDA  #"R"        ; es a REPEAT-kod is
680:   037F 48             PHA              ; a verembe
690:   0380 4C AE A7       JMP  INTER       ; Vissza az interpreter ciklushoz
700:                ;
710:   0383 20 73 00 UNTIL JSR  CHRGET      ; Van feltetel ?
720:   0386 F0 D7          BEQ  SYNERR      ; Ha nincs, hiba
730:   0388 20 BA AD       JSR  FRMNUM      ; Kiertekeles
740:   038B AB             TAY              ; Az eredmeny tarolasa
750:   038C BA             TSX              ; A veremmutato X - be
760:   038D BD 01 01       LDA  VEREM+1,X   ; Az utolso verem bejegyzes
770:   0390 C9 52          CMP  #"R"        ; REPEAT - kod ?
780:   0392 D0 23          BNE  RPTERR      ; Ha nem, hiba
790:   0394 98             TYA
800:   0395 D0 17          BNE  RPTVEGE     ; A kifejezes hamis, ciklus vege
810:                ;
820:   0397 BD 02 01        LDA  VEREM+2,X
830:   039A 85 3A          STA  SORSZ+1     ; Sorszam
840:   039C BD 03 01        LDA  VEREM+3,X
850:   039F 85 39          STA  SORSZ
860:   03A1 BD 04 01        LDA  VEREM+4,X
870:   03A4 85 7B          STA  TXTPTR+1    ; es program mutato
880:   03A6 BD 05 01        LDA  VEREM+5,X   ; A verembol az
890:   03A9 85 7A          STA  TXTPTR
900:   03AB 4C AE A7       JMP  INTER       ; Az interpreter-ciklusba
910:                ;
920:   03AE 8A       RPTVEGE TXA              ; Veremmutato
930:   03AF 18             CLC
940:   03B0 69 05          ADC  #5          ; 5 - tel novekszik
950:   03B2 AA             TXA
960:   03B3 9A             TXS
970:   03B4 4C AE A7       JMP  INTER       ; Vissza az interpreter-ciklusba
980:                ;
990:   03B7 A9 C0   RPTERR  LDA  #< TEXT
1000:  03B9 85 22          STA  ADR         ; Mutato a hibauzenetre
1010:  03BB A9 03          LDA  #> TEXT
1020:  03BD 4C 45 A4       JMP  ERROR
1030:                ;
1040:  03C0 55 4E 54 TEXT  .ASC "UNTIL WITHOUT REPEAT"
 03CC-03D4
NO ERRORS
```

Próbáljuk ki az új BASIC bővítést!
Az egyszerűség kedvéért a REPEAT utasítást a !R, az UNTIL utasítást a !U karakterekkel jelöljük.
Fordítsuk le a programot, majd aktivizáljuk a SYS 828 utasítással. Írjunk egy kis BASIC programot az új utasítás alkalmazásával:

```
***      P41.      ***

100 I=0
110 !R
120 I=I+1 : PRINT I
130 !U I=10
```

A program kiírja a számokat 1-től 10-ig. Az új ciklusszerkezeteket egymásba is skatulyázhatjuk:

```
***      P42.      ***

100 I=0
110 !R
120 I=I+1 : PRINT "I=";I : J=0
```

```
130 !R
140 J=J+1 : PRINT "J=";J
150 !U J=3
160 !U I=3
```

A futtatás eredményeként az I változó értéke változik 1-től 3-ig, és I minden rögzített értéke mellett J szintén változik 1-től 3-ig.
Természetesen ezt a feladatot FOR–NEXT ciklussal egyszerűbben lehet megoldani. A REPEAT–UNTIL szerkezetű ciklusokat elsősorban olyan feladatok megoldásánál célszerű alkalmazni, amelyekben nem lehet, vagy nem akarjuk előre megadni, hogy hányszor fusson le a ciklus. A futások számát függővé tehetjük pl. egy billentyű leütésétől.
Különösen nagy segítséget jelent ez a szerkezet az iterációs feladatokban, amelyekben a ciklus lefutásának száma egy bizonyos pontosság elérésétől függ. Ilyen pl. a négyzetgyökvonás a Newton-féle iterációs módszerrel.

```
***      P43.      ***

100 INPUT "MIBOL VONJAK NEGYZETGYOKOT";A
110 X1=A
120 !R
130 X0=X1
140 X1=(X0+A/X0)/2
150 !U ABS(X1-X0)<1E-8
160 PRINT "A GYOK:"X1
```

A programban mindaddig folytatjuk a gyök közelítését, amíg a két egymást követő érték abszolút értékben vett eltérése nem lesz kisebb mint 1E −8. Az iterációs eljárás SQR függvénnyel azonos értéket kell hogy adjon.
Hasonlóan egyszerű a végtelen ciklus programozása REPEAT–UNTIL szerkezettel. A kilépés feltételét úgy kell megválasztani, hogy az sohase teljesüljön.

```
***      P44.      ***

100 !R
110 PRINT TI
120 !U 1=0
```

A fenti program sosem áll meg.

A REPEAT–UNTIL szerkezetű ciklusok végrehajtása sokkal gyorsabb, mintha ugyanezt IF...GOTO szerkezetű utasítással oldottuk volna meg.
Míg az UNTIL utasítás végrehajtása során az ugrási címet a veremből kell betölteni, addig az IF...GOTO szerkezetnél minden lépésben újra meg kell keresni a GOTO utasításban szereplő sor címét.
Ráadásul a REPEAT–UNTIL szerkezet a programot is sokkal áttekinthetőbbé teszi.
Nézzük meg, hogyan működik a gépi kódú program.
A korábbi programokból már jól ismert inicializálás után a P.40-es program először megvizsgálja, hogy a végrehajtandó utasítás új, vagy eredeti BASIC utasítás. Ha nem talál felkiáltójelet, akkor a vektort visszaállítja az eredeti értékre. Egyébként beolvassa a következő karaktert, és megvizsgálja, hogy az R vagy U betű-e.
Az eredménytől függően vagy elágazik a REPEAT, ill. az UNTIL programrészre, vagy kiírja a SYNTAX ERROR hibaüzenetet.

A REPEAT ágon a programmutatót beállítjuk a CHRGET rutin hívásával a következő karakterre, majd megvizsgáljuk, hogy van-e elegendő helye a veremben. Ha igen, akkor a NEXTSTAT rutinnal megkeressük a következő utasítást, amelynek relatív címe az Y regiszterben található. Ehhez hozzáadjuk a programmutató értékét (így megkapjuk az abszolút címet) és ezt elhelyezzük a veremben. Ezután az aktuális sorszámot is tároljuk a veremben, majd egy R karaktert annak jelölésére, hogy a paraméterek a REPEAT utasításhoz tartoznak. A paraméterek teljesen azonosak a GOSUB utasításnál tárolt paraméterekkel.
Az UNTIL programrészben megvizsgáljuk, hogy a kiugrási feltétel teljesül-e. A vizsgálat eredményét az Y regiszterben tároljuk. Most betöltjük az X regiszterbe a veremmutatót és összehasonlítjuk a veremben tárolt legfelső elemet az R karakterrel.
Ha az eredmény negatív, akkor kiírjuk az UNTIL WITHOUT REPEAT (REPEAT nélküli UNTIL) hibaüzenetet. A hibaüzenet kiírása előtt az utolsó karakter 7. bitjét 1-re kell állítani (a shiftelt karakterek jelölése).
Ha a legfelső karakter R volt, akkor a továbbiak a feltétel teljesülésétől függnek. Ha a feltétel teljesül, akkor betöltjük a veremből a sorszámot és a programmutatót, és ugrunk az értelmező ciklushoz.
A veremből a paramétereket nem PLA (mert ez módosítaná a veremmutató értékét), hanem LDA VEREM, X utasítással töltjük vissza, mivel előtte a veremmutatót az X regiszterbe másoltuk. Így a veremmutató értéke változatlan marad, és az UNTIL többi paraméterei a veremben hozzáférhetők lesznek.
Végül, ha a kiugrási feltétel teljesült, a veremmutató értékét megnöveljük öttel, ezzel törölve a veremből a végrehajtott ciklus paramétereit, és folytatjuk a BASIC program futtatását.

3.4 AZ ÚJ UTASÍTÁSSZAVAK BEÉPÍTÉSE

Az új utasítások beépítésének legegyszerűbb módja az lenne, ha elláthatnánk ezeket egy új névvel, amelyeket a BASIC programon belül minden megkötés nélkül lehetne használni. Az utasítások nevéhez az ún. utasításszavakat vagy kulcsszavakat az értelmező tokenek (egykarakteres szimbólumok) formájában tárolja, amelyeket másképpen értelmező kódoknak is nevezünk, és amelyek értékei a $80-tól $FF-ig terjedő tartományba esnek.
A Commodore 64-es ezen a tartományon belül csak a $80-tól $CB-ig terjedő értékeket használja, ezen kívül csak a $FF kód kötött, ami a PI-nek felel meg. A $CC (204)-től a $FE (254)-ig terjedő kódokat szabadon használhatjuk, azaz 51 új utasítást építhetünk be a kódrendszerbe. Gondoljuk végig, hogyan lehet ezt a feladatot megoldani.
Először is készítenünk kell egy olyan rutint, ami a BASIC sor tárolása közben az új utasításszavakat tokenné alakítja. Azután szükség van egy olyan rutinra, amely a program futása közben felismeri az új tokent, és meghívja azt a rutint, amely az új utasítást végrehajtja. Módosítanunk kell a LIST rutint is, hiszen eredeti formájában a program listázása közben nem tudja az eddig ismeretlen tokent szöveggé alakítani. Célszerű az új utasításszavakat a hozzájuk tartozó rutinok címével együtt egy táblázatba foglalni, ahogyan azt az értelmező is teszi.
A BASIC vektorok között van négy olyan, amely kiváló segédeszköz lehet elképzelésünk megvalósításában.
Ezek közül kettőt már ismerünk: a BASIC utasítások végrehajtására ($308) és a függvények kiértékelésére ($30A) szolgáló vektorokat.
A tokenek szöveggé és a szövegek tokenné alakításához szükségünk lesz további két vektorra, amelyek a $304-es, ill. a $306-os címen találhatók.
A rutinokat csak egyszer kell precízen elkészíteni, a további kulcsszavak beillesztése igen egyszerűen megoldható; a már meglévő táblázatot ki kell egészíteni az új utasításszóval és az új rutin kezdőcímével. Az új utasítások beillesztésének ez a módja hatékonyabb az előző módszernél. Nem lassítja a végrehajtást a megkülönböztető karakter (a felkiáltójel) azonosítása. Ugyanakkor a BASIC programban is jobban mutat a szöveges REPEAT utasítás a !R jelölésnél.
Mielőtt nekilátnánk megírni azt a gépi kódú rutint, amely egy új kulcsszót tokenné alakít, érdemes tapasztalatot meríteni az értelmező hasonló rutinjából. Feltehetően kisebb módosításokkal alkalmassá tehetjük az általunk kitűzött feladat megoldására.
A P.45 a ROM megfelelő részének disassemblált listája.

```
***       P45.       ***
PROFI-ASS 64 V2.0

100:   A57C                    .OPT P
110:                    ;
120:                    ; Atalakitas token-ne, ROM rutin
130:                    ;
140:                    ; Kulonleges token-ek
```

```
150:                    ;
160:   0083            DATA    =   $83
170:   008F            REM     =   $8F
180:   0099            PRINT   =   $99
190:                    ;
200:   0008            CHAR    =   8           ; Az aktualis karakter
210:   000B            COUNT   =   11          ; Az utasitasszavak szamlaloja
220:   0071            PNT     =   $71         ; Mutato az atalakitott sorra
230:   0022            QUOTE   =   $22         ; Idezojel
240:   000F            FLAG    =   15          ; Kapcs.a DATA es REM utasitasnal
250:   007A            TXTPTR  =   $7A         ; Mutato az atalakitando sorra
260:   0200            BUFFER  =   $200        ; Beviteli puffer
270:                    ;
280:   A09E            TABLE   =   $A09E       ; Az utasitasok tablazata
290:                    ;
300:   A57C            **      =   $A57C       ; ROM - rutin
310:                    ;
320:   A57C A6 7A               LDX  TXTPTR     ; Mutato az elso karakterre
330:   A57E A0 04               LDY  #4         ; Mutato az atalakitott sorra
340:   A580 84 0F               STY  FLAG       ; A kapcsolo torlese
350:   A582 BD 00 02 NEXTCHAR   LDA  BUFFER,X   ; Egy karakter beolv. a pufferbol
360:   A585 10 07               BPL  NORMAL
370:   A587 C9 FF               CMP  #$FF       ; A "Pi" kodja ?
380:   A589 F0 3E               BEQ  TAKCHAR    ; Ha IGEN, a karaktert atvenni
390:   A58B E8                  INX             ; egyebkent atlepni
400:   A58C D0 F4               BNE  NEXTCHAR
410:                    ;
420:   A58E C9 20     NORMAL    CMP  #" "       ; Szokozkarakter ?
430:   A590 F0 37               BEQ  TAKCHAR    ; ha igen, akkor atvenni
440:   A592 85 08               STA  CHAR       ; A karakter tarolasa
450:   A594 C9 22               CMP  #QUOTE     ; Idezojel ?
460:   A596 F0 56               BEQ  GETCHAR    ; Igen
470:   A598 24 0F               BIT  FLAG       ; A kapcsolo vizsgalata
480:   A59A 70 2D               BVS  TAKCHAR    ; DATA - mod,atvenni
490:   A59C C9 3F               CMP  #"?"       ; Kerdojel ?
500:   A59E D0 04               BNE  SKIP
510:   A5A0 A9 99               LDA  #PRINT     ; PRINT - koddal helyettesiteni
520:   A5A2 D0 25               BNE  TAKCHAR
530:   A5A4 C9 30     SKIP      CMP  #"0"       ; Kisebb, mint 0 ?
540:   A5A6 90 04               BCC  SKIP1
550:   A5A8 C9 3C               CMP  #"<"       ; Kisebb, mint "<" ?
560:   A5AA 90 1D               BCC  TAKCHAR    ; Ha igen,akkor a kar.-t atvenni
570:   A5AC 84 71     SKIP1     STY  PNT        ; Sorbeli mutato tarolasa
580:   A5AE A0 00               LDY  #0
590:   A5B0 84 0B               STY  COUNT      ; A kulcsszavak szama = 0
600:   A5B2 88                  DEY
610:   A5B3 86 7A               STX  TXTPTR     ; Sorbeli mutato tarolasa
620:   A5B5 CA                  DEX
630:                    ;
640:   A5B6 C8     CMPLOOP   INY               ; A tablazatbeli es a sorbeli
640:   A5B7 E8               INX               ; mutato novelese
650:   A5B8 BD 00 02 TESTNEXT  LDA  BUFFER,X    ; Karakter a pufferbol
660:   A5BB 38                 SEC
670:   A5BC F9 9E A0           SBC  TABLE,Y     ; Osszehasonlitas a kulcsszoval
680:   A5BF F0 F5              BEQ  CMPLOOP     ; Ha egyenlo => kovetkezo kar.
690:   A5C1 C9 80              CMP  #$80        ; Utolso betu ?
700:   A5C3 D0 30              BNE  NEXTCMD     ; Egyebkent a mutato a kov.ut.-ra
710:   A5C5 05 0B              ORA  COUNT       ; Ha megvan => szam+$80 a kod
720:   A5C7 A4 71     TAKCHAR1  LDY  PNT         ; A mutato visszatoltese
730:                    ;
740:   A5C9 E8        TAKCHAR   INX
740:   A5CA C8                  INY
750:   A5CB 99 FB 01            STA  BUFFER-5,Y; A kod tarolasa
760:   A5CE B9 FB 01            LDA  BUFFER-5,Y; A kapcsolo visszaallitasa
770:   A5D1 F0 36               BEQ  ENDE       ; Sorvege ?
780:   A5D3 38                  SEC
790:   A5D4 E9 3A               SBC  #":"       ; Elvalaszto karakter ?
800:   A5D6 F0 04               BEQ  SKIP2      ; DATA - kapcsolo torlese
810:   A5D8 C9 49               CMP  #DATA-":" ; A "DATA"-kodja
820:   A5DA D0 02               BNE  SKIP3
```

```
830:  A5DC 85 0F      SKIP2    STA  FLAG       ; "DATA"-nal a 6.bit beallitasa
840:  A5DE 38         SKIP3    SEC
850:  A5DF E9 55               SBC  #REM-":"   ; A "REM"-kodja
860:  A5E1 D0 9F               BNE  NEXTCHAR   ; Nem, kov. karakter beolvasasa
870:  A5E3 85 08               STA  CHAR       ; 0 byte tarolasa "REM"-nel
880:  A5E5 BD 00 02 REMLOOP    LDA  BUFFER,X
890:  A5E8 F0 DF               BEQ  TAKCHAR    ; Ha sorveg => a kar.-t atvenni
900:  A5EA C5 08               CMP  CHAR       ; "" vagy REM vagy DATA
910:  A5EC F0 DB               BEQ  TAKCHAR    ; Igen ?
920:  A5EE C8         GETCHAR  INY
930:  A5EF 99 FB 01            STA  BUFFER-5,Y; A karakter atvetele
940:  A5F2 E8                  INX
950:  A5F3 D0 F0               BNE  REMLOOP
960:                  ;
970:  A5F5 A6 7A      NEXTCMD  LDX  TXTPTR     ; Sormutato a szo kezdetere
980:  A5F7 E6 0B               INC  COUNT      ; Szamlalo a kov. kulcsszora
990:  A5F9 C8         WEITER   INY
1000: A5FA B9 9D A0            LDA  TABLE-1,Y  ; A kovetkezo betu
1010: A5FD 10 FA               BPL  WEITER     ; A szonak nincs vege
1020: A5FF B9 9E A0            LDA  TABLE,Y
1030: A602 D0 B4               BNE  TESTNEXT   ; A kov. kulcsszo vizsgalata
1040:                 ;
1050: A604 BD 00 02            LDA  BUFFER,X
1060: A607 10 BE               BPL  TAKCHAR1   ; Kar. atv.valtozatlan formaban
1070:                 ;
1080: A609 99 FD 01 ENDE       STA  BUFFER-3,Y; A puffer lezarasa 0-val
1090:                 ;
1100: A60C C6 7B               DEC  TXTPTR+1
1110: A60E A9 FF               LDA  ##FF       ; TXTPTR $01FF-re, BUFFER-1
1120: A610 85 7A               STA  TXTPTR
1130  A612 60                   RTS
 A57C-A613
NO ERRORS
```

Az átalakítandó BASIC sor a $200-tól $258-ig terjedő beviteli pufferban van. A TXTPTR ($7A/$7B) a sorban közvetlenül a sorszám utáni első karakterre mutat. Ez a mutató bekerül az X regiszterbe. Az X regiszter mindvégig az átalakítás előtt álló, az Y regiszter pedig a már átalakított sorra mutat. A kapcsoló törlése után a rutin betölti a sor első karakterét. Ha a karakter kódja nagyobb, mint $7F, akkor megvizsgálja, hogy egyenlő-e $FF-fel. Ha igen, akkor a karaktert (PI) változatlanul tárolja, egyébként kihagyja a további vizsgálatból, azaz megnöveli eggyel a mutatót, és rátér a következő karakter vizsgálatára. Ha a kód nagyobb, mint $7F, akkor a hetedik bitje biztosan egy, ami arra utal, hogy a SHIFT billentyűvel együtt ütötték le a karaktert. Ezeket az ún. shiftelt karaktereket a rutin átlépi.

A további vizsgálatok kiszűrik a különleges karaktereket, ezeket ugyanis külön eljárással kell feldolgozni. A szóköz karaktert például a rutin változatlan formában tárolja. Az éppen vizsgált karaktert (ha nem szóköz) mindig a CHAR nevű változó tartalmazza. Ha ez idézőjel, a rutin elágazik a GETCHAR rutinba, amely a következő karaktereket egészen a bezáró idézőjelig változatlan formában tárolja.

A FLAG változó jelzi, ha az aktuális sor DATA utasítást tartalmaz. Ekkor ugyanis a DATA kulcsszót követő szöveg ismét változatlanul kerül át a programterületre. A kérdőjelet a rutin helyettesíti a PRINT tokenjével. A tényleges átalakítás akkor kezdődik, amikor a rutin számjegyet, kettőspontot vagy pontosvesszőt talál. Ezek természetesen változatlanok maradnak.

Az átalakított sorbeli mutató a PNT változóba kerül, a sorban szereplő kulcsszavak számát jelölő változó értéke nulla lesz. A kulcsszavak azonosítása a CMPLOOP címen kezdődik. A rutin a pufferban tárolt karaktert összehasonlítja

a táblázat első betűjével. Az összehasonlítás mindaddig folytatódik, amíg az eredmény nem lesz negatív. Ekkor a rutin megvizsgálja, hogy a kódok közötti különbség egyenlő-e $80-nal. Ebben az esetben ugyanis a karakterek abban különböznek egymástól, hogy az egyik shiftelt, a másik nem. A táblázatban a kulcsszó utolsó betűje mindig shiftelt, a $80 különbség tehát azt jelzi, hogy az utasítást megtaláltuk a táblázatban. Az értelmező kódot ekkor az akkumulátor tartalma és a COUNT (az utasítás sorszáma) változó közötti logikai VAGY (OR) művelet eredménye adja.

Ha a betűnkénti összehasonlítás negatív eredménnyel zárul, a NEXTCMD rutin megkeresi a táblázatban a következő utasítás kezdetét, a sorszámot megnöveli eggyel, és a hasonlítás kezdődik elölről. A táblázat végét egy nullabyte jelzi. Ha a keresés eredménytelen volt, akkor a rutin a szöveget változatlan formában tárolja.

A TAKCHAR címkétől kezdődik a tárolás – kódolva vagy eredeti formában. A különleges karakterek tárolását néhány mellékművelet is kíséri:

A kettőspont után törlődik a FLAG változó, amit az értelmező a DATA utasítás felismerésekor állít be újra. A rutin a REM utasítást követő karaktereket (a sor végéig) is változatlanul hagyja és a REMLOOP ciklusban tárolja. A REM-et egy, az aktuális karakter helyére beírt nulla jelzi.

A rutin az ENDE címkével ér véget, ahol az átalakított puffert egy nullával lezárja, és a TXTPTR mutatót ismét az adatbeviteli puffer elejére állítja.

A ROM rutint úgy kell átírnunk, hogy az, miután a saját belső táblázatában nem találta meg az utasításszót, folytassa tovább a keresést egy általunk készített táblázatban. Rögzítenünk kell azt is, hogy mi legyen az új utasítások értelmező kódja. A legjobb, ha a kódolást a $CC-től kezdjük, ott, ahol az eredeti BASIC utasítások kódjai véget érnek.

```
***      P46.      ***
PROFI-ASS 64 V2.0

100:   C000                    .OPT P,00
110:                           ;
120:                           ; Sajat token-ek
130:                           ;
140:                           ; Kulonleges token-ek
150:                           ;
160:   0083                    DATA      =    $83
170:   008F                    REM       =    $8F
180:   0099                    PRINT     =    $99
190:                           ;
200:   0008                    CHAR      =    8
210:   000B                    COUNT     =    11
220:   0071                    PNT       =    $71
230:   0022                    QUOTE     =    $22       ; Idezojel
240:   000F                    FLAG      =    15
250:   007A                    TXTPTR    =    $7A
260:   0200                    BUFFER    =    $200      ; Beviteli puffer
270:                           ;
280:   A09E                    TABLE     =    $A09E     ; Az utasitasok tablazata
290:                           ;
300:   C000                              *=   $C000     ; Az uj rutin
310:                           ;
320:   C000 A6 7A                        LDX  TXTPTR    ; Mutato az elso karakterre
330:   C002 A0 04                        LDY  #4        ; Mutato az atalakitott sorra
340:   C004 84 0F                        STY  FLAG      ; Kulonleges kar.-ek kapcsoloja
350:   C006 BD 00 02 NEXTCHAR  LDA  BUFFER,X ; Egy karakter beolv. a pufferbol
360:   C009 10 07              BPL  NORMAL
370:   C00B C9 FF              CMP  #$FF      ; A "Pi" kodja ?
```

```
C00D F0 3E              BEQ  TAKCHAR    ; Ha IGEN, a karaktert atvenni
C00F E8                 INX             ; egyebkent atlepni
C010 D0 F4              BNE  NEXTCHAR
                        ;
C012 C9 20    NORMAL    CMP  #" "       ; Szokozkarakter ?
C014 F0 37              BEQ  TAKCHAR    ; akkor atvenni
C016 85 08              STA  CHAR       ; A karakter tarolasa
C018 C9 22              CMP  #QUOTE     ; Idezojel ?
C01A F0 55              BEQ  GETCHAR
C01C 24 0F              BIT  FLAG       ; DATA uzemmod ?
C01E 70 2D              BVS  TAKCHAR    ; Ha igen, atvenni
C020 C9 3F              CMP  #"?"       ; Kerdojel ?
C022 D0 04              BNE  SKIP
C024 A9 99              LDA  #PRINT     ; PRINT - koddal helyettesiteni
C026 D0 25              BNE  TAKCHAR
C028 C9 30    SKIP      CMP  #"0"       ; Kisebb mint 0 ?
C02A 90 04              BCC  SKIP1
C02C C9 3C              CMP  #"<"       ; Kisebb mint "<" ?
C02E 90 1D              BCC  TAKCHAR    ; Ha igen,akkor a kar.-t atvenni
C030 84 71    SKIP1     STY  PNT        ; Sorbeli mutato tarolasa
C032 A0 00              LDY  #0
C034 84 0B              STY  COUNT      ; A kulcsszavak szama = 0
C036 88                 DEY
C037 86 7A              STX  TXTPTR
C039 CA                 DEX
                        ;
C03A C8       CMPLOOP   INY
C03B E8                 INX             ; mutato novelese
C03C BD 00 02 TESTNEXT  LDA  BUFFER,X   ; Karakter a pufferbol
C03F 38                 SEC
C040 F9 9E A0           SBC  TABLE,Y    ; Osszehasonlitas a kulcsszoval
C043 F0 F5              BEQ  CMPLOOP    ; Ha egyenlo => kovetkezo kar.
C045 C9 80              CMP  #$80       ; Utolso betu ?
C047 D0 2F              BNE  NEXTCMD    ; Egyebkent a mutato a kov.ut.-ra
C049 05 0B              ORA  COUNT      ; Nr + $80 = Interpreterkod
C04B A4 71    TAKCHAR1  LDY  PNT        ; A mutato visszatoltese
                        ;
C04D E8       TAKCHAR   INX
C04E C8                 INY
C04F 99 FB 01           STA  BUFFER-5,Y ; A kod tarolasa
C052 C9 00              CMP  #0         ; A kapcsolo visszaallitasa
C054 F0 38              BEQ  ENDE       ; Sorvege ?
C056 38                 SEC
C057 E9 3A              SBC  #":"       ; Elvalaszto karakter ?
C059 F0 04              BEQ  SKIP2
C05B C9 49              CMP  #DATA-":"  ; A "DATA"-kodja ?
C05D D0 02              BNE  SKIP3
C05F 85 0F    SKIP2     STA  FLAG       ; "DATA"-nal a 6.bit beallitasa
C061 38       SKIP3     SEC
C062 E9 55              SBC  #REM-":"   ; A "REM"-kodja ?
C064 D0 A0              BNE  NEXTCHAR   ; Nem, kov. karakter beolvasasa
C066 85 08              STA  CHAR       ; A karakter tarolasa
C068 BD 00 02 REMLOOP   LDA  BUFFER,X
C06B F0 E8              BEQ  TAKCHAR    ; Ha sorveg => a kar.-t atvenni
C06D C5 08              CMP  CHAR       ; "" vagy REM vagy DATA
C06F F0 DC              BEQ  TAKCHAR    ; Igen ?
C071 C8       GETCHAR   INY
C072 99 FB 01           STA  BUFFER-5,Y ; A karakter atvetele
C075 E8                 INX
C076 D0 F0              BNE  REMLOOP
                        ;
C078 A6 7A    NEXTCMD   LDX  TXTPTR
C07A E6 0B              INC  COUNT      ; Szamlalo a kov. kulcsszora
C07C C8       WEITER    INY
C07D B9 9D A0           LDA  TABLE-1,Y  ; A kovetkezo betu
C080 10 FA              BPL  WEITER     ; A szonak nincs vege ?
C082 B9 9E A0           LDA  TABLE,Y
C085 D0 B5              BNE  TESTNEXT   ; A kov. kulcsszo vizsgalata
C087 F0 0F              BEQ  NEWTOK     ; Az uj tablazat hasznalata
                        ;
```

```
1060:  C089 BD 00 02 NOTFOUND  LDA  BUFFER,X
1070:  C08C 10 BD              BPL  TAKCHAR1  ; A karakter atvetele
1080:                          ;
1090:  C08E 99 FD 01 ENDE      STA  BUFFER-3,Y; A direktmod linkbyte = 0
1100:                          ;
1110:  C091 C6 7B              DEC  TXTPTR+1
1120:  C093 A9 FF              LDA  #$FF      ; TXTPTR #01FF-re, BUFFER-1
1130:  C095 85 7A              STA  TXTPTR
1140:  C097 60                 RTS
1150:                          ;
1160:                          ; Az uj utasitasok feldolgozasa
1170:  C098 A0 00              NEWTOK   LDY  #0        ; Mutato az uj tablazat kezdetere
1180:  C09A B9 C3 C0           LDA  NEWTAB,Y  ; Az 1. kar. beolv. a tablazatbol
1190:  C09D D0 02              BNE  NEWTEST
1200:                          ;
1210:  C09F C8                 NEWCMP   INY
1210:  C0A0 E8                 INX
1220:  C0A1 BD 00 02 NEWTEST   LDA  BUFFER,X  ; Az uj tablazat osszehasonlito
1230:  C0A4 38                 SEC            ; rutinja
1240:  C0A5 F9 C3 C0           SBC  NEWTAB,Y
1250:  C0A8 F0 F5              BEQ  NEWCMP
1260:  C0AA C9 80              CMP  #$80
1270:  C0AC D0 04              BNE  NEXTNEW   ; A kov. uj utasitas vizsgalata
1280:  C0AE 05 0B              ORA  COUNT     ; Megvan
1290:  C0B0 D0 99              BNE  TAKCHAR1  ; Feltetlen ugras
1300:                          ;
1310:  C0B2 A6 7A              NEXTNEW  LDX  TXTPTR
1320:  C0B4 E6 0B              INC  COUNT     ; A token szamanak novelese
1330:  C0B6 C8                 WEITER1  INY
1340:  C0B7 B9 C2 C0           LDA  NEWTAB-1,Y; Mutato a kov. kulcsszora
1350:  C0BA 10 FA              BPL  WEITER1
1360:  C0BC B9 C3 C0           LDA  NEWTAB,Y
1370:  C0BF D0 E0              BNE  NEWTEST   ; Osszehasonlitas
1380:  C0C1 F0 C6              BEQ  NOTFOUND  ; Az uj tablazat vege
1390:                          ;
1400:  C0C3 52 45 50 NEWTAB    .ASC "REPEAT"  ; Az uj kulcsszavak tablazata
1410:  C0C9 55 4E 54           .ASC "UNTIL"
1420:  C0CE 42 45 46           .ASC "BEFEHL"
1430:  C0D4 00                 .BYT 0         ; A tablazat vege
 C000-C0D5
NO ERRORS
```

A program utolsó része tartalmazza a kulcsszavak új táblázatát. Begépelésekor ügyeljünk arra, hogy a kulcsszavak utolsó betűjét a SHIFT billentyűvel együtt kell leütni. A programlistán ezt kiemeltük. Az új utasításokat az eredetiekhez hasonlóan lehet röviditeni, a REPEAT helyett gépelhetünk reP-et, az UNTIL helyen uN-t. (A nagybetűk a SHIFT billentyű leütésére utalnak.)
Az új utasítások száma maximum 51 lehet, a $CC-től $FE kódoknak megfeleltetve. Az utasításszavak hossza nem lehet több, mint 255 karakter (hiszen a táblázat belső mutatója nyolc bites), és a táblázatot egy nulla byte-tal kell lezárni.
Az új utasítások „felélesztéséhez" a $304/$305 vektort a rutinokra kell irányítani. Mielőtt ezt megtehetnénk, meg kell írnunk azt a rutint, amely elvégzi az előzővel ellentétes feladatot: a kódokat visszaalakítja kulcsszavakká. A listázás lehetetlen enélkül. Az értelmező a $306/$307 vektoron keresztül alakítja vissza a kódokat szöveggé. Elemezzük először ismét az értelmező saját rutinját: a LIST rutint.

```
***      P47.      ***

PROFI-ASS 64 V2.0

100:   A71A                    .OPT P
110:                           ;
120:                           ; Az interpreter LIST rutinja
```

```
130:                    ;
140:    000F            QUOTFLG     =    15          ; Az idezojelmod kapcsoloja
150:    0049            PNT         =    $49
160:    A09E            TABLE       =    $A09E       ; Az int. utasitastablazata
170:    AB47            CHAROUT     =    $AB47       ; A karakter leirasa
180:                    ;
190:    A71A                        *=   $A71A
200:    A71A 10 D7                  BPL  $A6F3       ; Ha nem kod, akkor kiirni
210:    A71C C9 FF                  CMP  #$FF
220:    A71E F0 D3                  BEQ  $A6F3       ; Ha a Pi kodja akkor kiirni
230:    A720 24 0F                  BIT  QUOTFLG     ; Idezojel-mod ?
240:    A722 30 CF                  BMI  $A6F3       ; Igen, valtozatlanul kiirni
250:    A724 38                     SEC
260:    A725 E9 7F                  SBC  #$7F        ; Az offset kivonasa
270:    A727 AA                     TAX              ; A kod tarolasa szamlalokent
280:    A728 84 49                  STY  PNT         ; A mutato tarolasa
290:    A72A A0 FF                  LDY  #-1
300:    A72C CA             NEXT    DEX
310:    A72D F0 08                  BEQ  FOUND       ; Az utasitasszo megvan ?
320:    A72F C8             LOOP    INY
330:    A730 B9 9E A0               LDA  TABLE,Y
340:    A733 10 FA                  BPL  LOOP        ; A szonak nincs vege ?
350:    A735 30 F5                  BMI  NEXT        ; A kovetkezo szo
360:                    ;
370:    A737 C8             FOUND   INY
380:    A738 B9 9E A0               LDA  TABLE,Y     ; A betu beolvasasa
390:    A73B 30 B2                  BMI  $A6EF       ; Utolso kar. ?
400:    A73D 20 47 AB               JSR  CHAROUT     ; A karakter kiirasa
410:    A740 D0 F5                  BNE  FOUND       ; Feltetlen ugras
 A71A-A742
NO ERRORS
```

A rutin megvizsgálja, hogy a tárolt jel értelmező kód-e (a hetedik bit 1 ?). A PI-t változatlanul kiírja. Az idézőjel is automatikus kiíráshoz vezet. Itt kezdődik az utasításszavak valódi keresése.
Ha a kódból levonunk $7F-et, az eredmény 1 és 76 közé esik. A táblázaton haladva minden utasításszó végén (amit az utolsó karakter 7. bitjéből ismerünk fel) eggyel csökkentjük az előbb kapott számot. Amint ez elérte a nullát, a szót megtaláltuk.
A szó karaktereit sorra kiírjuk, kivéve az utolsót. Itt elágazunk ismét a LIST rutinba, ott töröljük a karakter hetedik bitjét, majd ezt is kiírjuk.
Az új kódok listázása igen egyszerű. Meg kell vizsgálnunk, hogy a kód nagyobb-e, mint $CB.
Ha igen, a saját táblázatunkban kell keresni, egyébként az értelmezőt hagyjuk tovább dolgozni.

```
***       P4B.       ***
PROFI-ASS 64 V2.0

100:    C000                        .OPT P,00
110:                    ;
120:                    ; Az uj utasitasok LIST rutinja
130:                    ;
140:    000F            QUOTFLG     =    15          ; Az idezojelmod kapcsoloja
150:    0049            PNT         =    $49
160:    A09E            TABLE       =    $A09E       ; Az int. utasitastablazata
170:    AB47            CHAROUT     =    $AB47       ; A karakter leirasa
180:                    ;
200:    C000 10 0F                  BLP  OUT         ; Ha nem token, akkor kiirni
210:    C002 24 0F                  BIT  QUOTFLG.    ; Idezojel-mod ?
220:    C004 30 0B                  BMI  OUT         ; Ha igen, kiirni
230:    C006 C9 FF                  CMP  #$FF        ; Pi ?
240:    C008 F0 07                  BEQ  OUT         ; Ha igen, kiirni
250:    C00A C9 CC                  CMP  #$CC        ; Uj token ?
```

```
260:   C00C B0 06                BCS  NEWLIST    ; Igen
270:                  ;
280:   C00E 4C 24 A7             JMP  $A724      ; A regi token listazasa
290:   C011 4C F3 A6 OUT         JMP  $A6F3      ; A byte kiirasa
300:                  ;
310:   C014 38       NEWLIST     SEC
320:   C015 E9 CB                SBC  #$CB       ; Az offset levonasa
330:   C017 AA                   TAX             ; A kod, mint szamlalo
340:   C018 84 49                STY  PNT
350:   C01A A0 FF                LDy  #-1
360:   C01C CA       NEXT        DEX             ; A szo megvan ?
370:   C01D F0 08                BEQ  FOUND      ; Igen
380:   C01F C8       LOOP        INY
390:   C020 B9 35 C0             LDA  NEWTAB,Y
400:   C023 10 FA                BPL  LOOP       ; Varakozas a szo vegere
410:   C025 30 F5                BMI  NEXT       ; Kovetkezo szo
420:                  ;
430:   C027 C8       FOUND       INY
440:   C028 B9 35 C0             LDA  NEWTAB,Y   ; Utasitasszo
450:   C02B 30 05                BMI  OLDEND     ; Vege ?
460:   C02D 20 47 AB             JSR  CHAROUT    ; A kar. kiirasa
470:   C030 D0 F5                BNE  FOUND      ; es tovabb
480:                  ;
490:   C032 4C EF A6 OLDEND      JMP  $A6EF      ; Ugras a regi rutinra
500:                  ;
510:   C035 52 45 50 NEWTAB      .ASC "REPEAT"   ; Utasitastablazat
520:   C03B 55 4E 54             .ASC "UNTIL"
530:   C040 42 45 46             .ASC "BEFEHL"
540:   C046 00                   .BYT 0
 C000-C047
NO ERRORS
```

Állítsuk át a LIST vektort ($306/$307) a mi rutinunk kezdőcímére. A LIST parancs az általunk megadott kulcsszavakat helyesen írja ki. A NEWTAB az új utasítástáblázat kezdőcíme. A táblázatot az új szavakkal természetesen csak egyszer kell elkészíteni.A két rutint fordítsuk le, és kössük össze egy olyan főprogrammal, amelyben a vektorokat a megfelelőképpen módosítjuk.

Az eddig elkészült programok még nem készítették fel az értelmezőt az új utasítások felismerésére. Ahhoz, hogy az utasítások valóban használhatóak legyenek, meg kell írnunk a végrehajtásukhoz szükséges rutinokat, és módosítanunk kell a $308/309 (utasítások)/, ill. a $30A/$30B (függvények) vektorokat. A feldolgozás egyszerűsítése úgy kívánja, hogy az új utasításokat és függvényeket egy blokkban helyezzük el. A feldolgozás során ellenőriznünk kell, hogy a végrehajtandó utasítás tokenje benne van-e az új utasítások tartományában. A token sorszámát indexként használhatjuk az utasítások kezdőcímeit tartalmazó táblázatban.

Az alábbiakban közöljük annak a rutinnak az assemblerlistáját, amely teljesen átveszi az értelmezőtől az új utasítások feldolgozásának feladatát. Futtatása előtt el kell döntenünk, hogy a tárban hol helyezzük el az új rutinokat, és a kezdőcímeket egy táblázatba kell foglalnunk.

```
***      P50.      ***
PROFI-ASS 64 V2.0

100:   C000                      .OPT P1
110:                  ;
120:                  ; Az uj utasitasok kiepitese
130:                  ;
150:   0308          CMDVEK      =    $308       ; Utasitasvektor
160:   030A          FUNVEK      =    $30A       ; Fuggvenyvektor
170:                  ;
170:   000D          TYPFLAG     =    13         ; Kapcsolo : numerikus/string
```

```
0073            CHRGET  =  $73
0079            CHRGOT  =  CHRGET+6
007A            TXTPTR  =  CHRGOT+1
A7ED            EXECOLD =  $A7ED      ; A regi utasitas vegrehajtasa
A7AE            INTER   =  $A7AE      ; Interpreterciklus
AE8D            FUNKTOLD=  $AE8D      ; A regi fuggvenyszamitas
AEF1            GETTERM =  $AEF1      ; A zarojeles kif. beolvasasa
AD8D            CHECKNUM=  $AD8D      ; Az eredmeny num. ellenorzese
0054            JUMP    =  $54        ; Ugras a fuggvenyre
00CC            CMDSTART=  $CC        ; Az elso utasitastoken
00E0            CMDEND  =  $E0        ; Az utolso utasitastoken

00E1            FUNSTART=  $E1        ; Az elso fuggvenytoken
00FE            FUNEND  =  $FE        ; Az utolso fuggvenytoken

C000 A9 15      INIT    LDA  #< NEWCMD
C002 A0 C0              LDY  #> NEWCMD
C004 8D 08 03           STA  CMDVEK   ; Az utasitasvektor
C007 8C 09 03           STY  CMDVEK+1

C00A A9 3C              LDA  #< NEWFUN
C00C A0 C0              LDY  #> NEWFUN
C00E 8D 0A 03           STA  FUNVEK   ; Fuggvenyvektor
C011 8C 0B 03           STY  FUNVEK+1
C014 60                 RTS

C015 20 73 00 NEWCMD    JSR  CHRGET   ; A token beolvasasa
C018 20 1E C0           JSR  TESTCMD  ; Az utasitas vegrehajtasa
C01B 4C AE A7           JMP  INTER    ; Vissza az interpreter ciklushoz

C01E C9 CC      TESTCMD CMP  #CMDSTART
C020 90 04              BCC  OLDCMD   ; Regi utasitas ?
C022 C9 E1              CMP  #CMDEND+1
C024 90 06              BCC  OKNEW    ; Az uj utasitas feldolgozasa
C026 20 79 00 OLDCMD    JSR  CHRGOT   ; Kapcsolo visszaallitasa
C029 4C ED A7           JMP  EXECOLD  ; A regi utasitas vegrehajtasa

C02C 38         OKNEW   SEC           ; Uj utasitas
C02D E9 CC              SBC  #CMDSTART; Az offset levonasa
C02F 0A                 ASL           ; 2 - szer
C030 AA                 TAX
C031 BD 6F C0           LDA  CMDTAB+1,X; Felso byte
C034 48                 PHA           ; Visszateresi cim a verembe
C035 BD 6E C0           LDA  CMDTAB,X
C038 48                 PHA           ; Also byte
C039 4C 73 00           JMP  CHRGET   ; A kovetkezo karakter beolvasasa

C03C A9 00      NEWFUN  LDA  #0
C03E 85 0D              STA  TYPFLAG  ; Numerikus kapcsolo
C040 20 73 00           JSR  CHRGET   ; A token beolvasasa
C043 C9 E1              CMP  #FUNSTART
C045 90 04              BCC  OLDFUN   ; A regi fuggveny
C047 C9 FF              CMP  #FUNEND+1
C049 90 06              BCC  OK1NEW
C04B 20 79 00 OLDFUN    JSR  CHRGOT   ; A kapcsolo visszaallitasa
C04E 4C 8D AE           JMP  FUNKTOLD ; A regi fuggveny kiszamitasa

C051 38         OK1NEW  SEC           ; Az uj fuggveny
C052 E9 E1              SBC  #FUNSTART; Az offset levonasa
C054 0A                 ASL
C055 48                 PHA           ; A tablazatmutato tarolasa
C056 20 73 00           JSR  CHRGET   ; A kovetkezo kar. toltese
C059 20 F1 AE           JSR  GETTERM  ; Az argumentum beolvasasa
C05C 68                 PLA
C05D A8                 TAY           ; A mutato, mint index
C05E B9 72 C0           LDA  FUNTAB,Y ; Also cim-byte
C061 85 55              STA  JUMP+1
C063 B9 73 C0           LDA  FUNTAB+1,Y; Felso cim-byte
C066 85 56              STA  JUMP+2
C068 20 54 00           JSR  JUMP     ; A fuggveny vegrehajtasa
```

```
930:    C068 4C BD AD              JMP   CHECKNUM   ; Az numerikus ell.
940:                       ;
950:    C06E XX XX         CMDTAB  .WOR  CMD1-1     ; Az utasitascimek tabl.-1
960:    C070 XX XX                 .WOR  CMD2-1
970:                       ;.....
980:    C072 XX XX         FUNTAB  .WOR  FUN1       ; Fuggvenycimek tablazata
990:    C074 XX XX                 .WOR  FUN2
 C000-C076
NO ERRORS
```

A program 300-as és 310-es soraiban meg kell adni az első és utolsó új utasítás, a 330-as és 340-es sorokban pedig az első és utolsó új függvény tokenjét. A 950-től 960-ig terjedő sorokban el kell helyezni az új rutinok kezdőcímeit. Mivel a rutinokba az RTS utasítással ugrunk, amely általában a veremből veszi a visszaugrási címet, az új rutinok tényleges címe helyett mindig eggyel kisebbet kell beírni, mert az RTS utasítás a visszaugrási címet automatikusan eggyel megnöveli.
A függvények esetére ez nem igaz, hiszen a függvényeket a JSR utasítással hívjuk meg.

3.5 AZ OPERÁCIÓS RENDSZER VEKTORAI

A BASIC értelmezőhöz hasonlóan az operációs rendszer legfontosabb eljárásait is vektorokon keresztül lehet meghívni, ami nagymértékben megkönnyíti egyéni elképzeléseink megvalósítását. A hardveres célú vektorokon (IRQ, BRK, NMI) kívül – amelyeket már korábban megismertünk – minden elemi input/output művelet végrehajtása hasonló szervezésű. Ez vonatkozik a $FFXX Kernal rutinon keresztül hívható függvényekre is. Az alábbi táblázat tartalmazza az összes vektort és azokat a címeket, amelyekre a vektorok a gép bekapcsolása után mutatnak.

```
***      T47.      ***

 Vektor            Cim        Jelentes
-------------------------------------------------------
$0314/$0315       $EA31      IRQ-vektor
$0316/$0317       $FE66      BRK-vektor
$0318/$0319       $FE47      NMI-vektor
$031A/$031B       $F34A      OPEN-vektor
$031C/$031D       $F291      CLOSE-vektor
$031E/$031F       $F20E      CHKIN-vektor
$0320/$0321       $F250      CKOUT-vektor
$0322/$0323       $F333      CLRCH-vektor
$0324/$0325       $F157      BASIN-vektor
$0326/$0327       $F1CA      BSOUT-vektor
$0328/$0329       $F6ED      STOP-vektor
$032A/$032B       $F13E      GET-vektor
$032C/$032D       $FE66      Melegstart-vektor (kihasznalatlan)
$032E/$032F       $F4A5      LOAD-vektor
$0330/$0331       $F5ED      SAVE-vektor
```

Ebben a fejezetben részletesen ismertetjük a vektorok jelentését és a hozzájuk tartozó rutinok feladatát. Az elméleti alapok elsajátítása után az Olvasó kis befektetéssel készíthet saját input/output rutinokat.

OPEN – JSR $FFCO

Feladata tökéletesen megegyezik az azonos nevű BASIC utasítás feladataival. Hívás előtt a szükséges paramétereket elő kell készíteni. Erre szolgál a következő két rutin.

SETFLS – JSR $FFBA

A rutin rögzíti a logikai file-szám, az egységszám és a másodlagos cím értékét. Hívása előtt a paramétereket egyszerűen be kell írni a processzor regisztereibe:

```
***      P51.      ***

LDA LF      ; Logikai file-szam
LDX FA      ; Egysegszam
LDY SA      ; Masodlagos cim
JSR SETFLS  ; A parameterek beallitasa
```

SETNAM – JSR $FFBD

Ez a rutin előkészíti az OPEN rutin végrehajtásához a file nevét. A név tárbeli kezdőcímét és hosszát ismét a regiszterekbe kell beírni:

```
***       P52.       ***

        LDA #NEV1-NEV    ; A file-nev hossza
        LDX #< NEV       ; A cim also byte-ja
        LDY #> NEV       ; A cim felso byte-ja
        JSR SETNAM       ; A parameterek atadasa
        ...
NEV     .ASC "NEV"
NEV1    = * ; NEV VEGE
```

Ha nincs file-név, a hossz helyére nulla kerül.
Az utóbbi két rutin végrehajtása után meghívhatjuk az OPEN rutint.

JSR OPEN

Ha a logikai file megnyitása közben hiba lépne fel, a rutin a C (Carry) kapcsolót 1-re állítja, és a hiba számát elhelyezi az akkumulátorban. A hibaszámok jelentése:

```
***       T48.       ***
 Sorszam          Jelentes
-------------------------------------
   0          Megszakitas STOP-billentyuvel
   1          Tul sok nyitott file
   2          A file nyitva van
   3          A file nincs nyitva
   4          A file nincs meg
   5          Az egyseg nincs jelen
   6          Nem INPUT-file
   7          Nem OUTPUT-file
   8          Hianyzo file-nev
   9          Illegalis egysegszam
 240          RS 232 OPEN/CLOSE
```

A kernalrutinok hívása után mindig ellenőrizni kell a C kapcsolót, és ha értéke 1, akkor a hibát le kell kezelni:

```
***       T49.       ***

    JSR OPEN   ; File nyitas
    BCC OK     ; Minden rendben ?
    JMP ERROR
OK  ...
```

A hibaüzeneteket az Olvasó már ismeri, ezek közül eddig csak egyről nem esett szó. Az RS–232-es illesztő (2-es egységszámmal) nyitása és zárása esetén kapjuk a 240-es sorszámú hibaüzenetet, ha a velet elvégzése közben hiba lépett fel.
Ahogyan ezt az Olvasó feltehetően tudja; az RS 32-es csatorna megnyitásánál a rendszer két, egyenként 256 byte input/output pufferrel dolgozik. A két puffer a BASIC terület felső határán kezdődik.
A BASIC vége programonként változik a $A000–$9E00 tartományon belül. Elő-

fordulhat olyan eset is, hogy a füzérek teljesen elfoglalják ezt a területet, és így a pufferek számára nem marad hely. Ilyenkor az OPEN rutin a 240-es hibaszámot beírja az akkumulátorba és a C kapcsolót 1-re állítja, ezzel tájékoztatva az értelmezőt a hibajelenségről. Az értelmező válaszul végrehajt egy CLR utasítást, és törli a változókat. A CLOSE utasítás ismét felszabadítja a puffereket és szintén törli a változókat. Ezért az RS–232-es csatornával dolgozó programokban mindig a program elejére kell írni az OPEN és a végére a CLOSE utasítást. Csak így biztosíthatjuk, hogy a program változóit futás közben az OPEN és CLOSE utasítások ne semmisítsék meg.
Előfordulhat, hogy valakinek ez az eljárás nem megfelelő – pl. nem akarja megcsonkítani a BASIC területet a pufferek méretével. Az OPEN rutint át lehet úgy írni, hogy a puffereket mindig fix helyre, pl. a $C000-s címtől kezdve rakja le, így nem lesz szükség a változók törlésére, a rutinból kihagyhatjuk a CLR utasítást.
Az input/output rutinok is a C kapcsolót használják hibajelzésre, és az előzőekhez hasonlóan az akkumulátor tartalmazza a hiba számát.
A hibaüzeneteket az operációs rendszer külön rutinja írja ki. A rutin az üzenetet az alábbi formában generálja:

I/O ERROR#X

ahol X a hiba száma (1-től 9-ig). A hiba nem szakítja meg a program futását. A hibaüzenet kiírását a

SETMSG – JSR $FF90

rutin hívásával aktivizálhatjuk úgy, hogy előtte az akkumulátorba **$40-et** írunk (a hatodik bit értéket 1-re állítjuk). Az üzenetek kiírását ugyancsak a SETMSG rutinnal lehet kikapcsolni, az akkumulátor nulla értéke mellett.
A SETMSG rutin másik feladata a program- és a parancs üzemmód megkülönböztetése. A kapcsoló az akkumulátor hetedik bitje. Program üzemmódban a kapcsoló értéke 1. A rendszer üzenetei ilyenkor:
SEARCHING FOR , LOADING , SAVING .

CLOSE – JSR $FFC3

A CLOSE rutin egyetlen paramétere a file logikai száma, ezt hívás előtt az akkumulátorba kell tölteni.

```
***       T50.       ***

LDA LF
JSR CLOSE
```

A CLOSE rutin, egyetlen esettől eltekintve, nem generál hibaüzenetet. Az RS–232-es csatorna lezárása közben a rendszer felszabadítja a puffereket, és végrehajtja a CLR utasítást. A meg nem nyitott file lezárási kísérlete hibajelzéshez vezet.

CHKIN – JSR $FFC6

Alapállapotban az elsődleges beviteli egység a billentyűzet. A rutinnal ezt az állapotot módosíthatjuk, azaz kijelölhetjük elsődleges bevitelre a korábban megnyitott logikai file-t. Hívása előtt a logikai számot be kell írni az X regiszterbe.

```
***       T51.       ***
LDX LF
JSR CHKIN
```

Természetesen a logikai file-hoz rendelt egységnek bemeneti egységnek kell lennie, egyébként hibaüzenetet kapunk (NOT INPUT FILE). A hibakezelés módja a megszokott. Hibaüzenetet okozhat pl. az, hogy előzetesen nem nyitottuk meg a file-t (FILE NOT OPEN). A tulajdonképpeni adatbevitelt a BASIN rutin bonyolítja le, amiről a későbbiekben esik majd szó.

CKOUT – JSR $FFC9

A CHKIN rutinnal szemben, ami az adatbevitelt készíti elő, a CKOUT rutin az adatok kiírását készíti elő. Alapállapotban az elsődleges kiviteli egység a képernyő.
A CKOUT rutin az alaphelyzetet módosítja, és kiviteli egységként az előzetesen megnyitott logikai file-hoz rendelt egységet jelöli meg. A rutin BASIC megfelelője a CMD utasítás. A file logikai számát ismét az X regiszterben kell a rutin számára előkészíteni.

```
***       T52.       ***
LDX LF
JSR CKOUT
```

A hibakezelés az előzőekhez hasonló. Ha pl. egy olvasásra megnyitott szalagos file-t akarunk adatkiírásra kijelölni, a NOT OUTPUT FILE hibaüzenetet kapjuk. A kijelölt file-ba az adatokat a BSOUT rutin írja fel.

BASIN – JSR $FFCF

A rutin a BASIC INPUT rutin megfelelője.
Hívása előtt CHKIN rutinnal meg kell jelölnünk az adatbeviteli forrást, az input file-t.
Alapállapotban a rutin a billentyűzetről olvassa az adatokat, és a képernyőre írja.
Ha egy gépi kódú programból meghívjuk ezt a rutint, a kurzor megjelenik a képernyőn, és mindaddig folyamatosan olvassa az adatokat, amíg a RETURN billentyűt (CHR$(13)) le nem ütjük.
Az első beolvasott karaktert elhelyezi az akkumulátorban. Minden további karaktert a rutin újabb hívásával kaphatjuk vissza az akkumulátorban.
Az adatok folyamatos begépelése közben használhatjuk a képernyőszerkesztőt (karakterek törlése, beszúrása stb.).

Ha az adatokat nem a billentyűzetről, hanem pl. egy lemezes file-ból akarjuk beolvasni, ami megfelel a BASIC INPUT utasításnak, akkor a rutin hívása előtt a CHKIN rutinnal át kell irányítanunk a beolvasást az adott logikai file-ra. Ebben az esetben a BASIN rutin minden hívásakor beolvas egy karaktert a file-ból és átadja az akkumulátorba.

BSOUT– JSR $FFD2

A BSOUT rutin kiír egy karaktert a képernyőre.
A karakter ASCII kódját előzetesen be kell tölteni az akkumulátorba. Például az

```
***       T53.       ***
LDA #$41
JSR BSOUT
```

utasításpár a $41 (65) ASCII kódú karaktert kiírja a képernyőre (történetesen az "A" betűt).
A rutin a közönséges karakterek kiírásán kívül vezérlőkarakterek kiírására is alkalmas, amit BASIC-ben pl. a PRINT CHR$(X) utasítással szoktunk megtenni.
A soremelést, amit BASIC nyelven egy változókat nem tartalmazó PRINT utasítás végez el, a következőképpen kell gépi kódban programozni:

```
***       T54.       ***
LDA #13     ; Kocsivissza
JSR BSOUT   ; Kiirni
```

A kiírás vonatkozhat egy tetszőleges file-ra is, amelyhez előzetesen hozzárendeltük pl. a nyomtatót vagy a lemezegységet. Ilyenkor a rutin hívása előtt a CKOUT rutinnal át kell irányítani az adatok kiírását az adott egységre.
A kiírás közben fellépő hibák kezelése az eddig megismert módon történik. Ha pl. a soros busz adott egysége nem „válaszol", akkor a DEVICE NOT PRESENT hibaüzenetet kapjuk.

CLRCH – JSR $FFCC

A rutin feladata ellentétes a CHKIN, ill. CKOUT rutinok feladatával.
Visszaállítja az alaphelyzetet; kijelöli elsődleges beviteli egységként a billentyűzetet, kiviteli egységként a képernyőt.
A következő programrészletben a 2-es logikai file-ból (a lemezegységről) beolvasunk 10 karaktert.

```
***       P53.       ***
        LDX #2        ; Logikai file-szam
        JSR CHKIN     ; Beolvasas a 2-es file-bol
        LDY #0
LOOP    JSR BASIN     ; Egy karakter beolvasasa a lemezegysegrol
        STA STORE,Y   ; es tarolasa
        INY
        CPY #10       ; Mar beolvasott 10 karaktert ?
        BNE LOOP      ; Nem
        JSR CLRCH     ; Az alaphelyzet visszaallitasa
```

A program futtatása előtt meg kell nyitni a 2-es logikai file-t. Első lépésben átkapcsoljuk a beolvasást erre a file-ra, a BASIN rutinnal betöltünk 10 karaktert és ezeket tároljuk, majd a CLRCH rutinnal visszarendeljük a beolvasást a billentyűzethez.
A program futtatása után a file-t CLOSE utasítással le kell zárni.

GET – JSR $FFF4

A rutin a BASIC GET utasítás megfelelője: beolvas egy karaktert a billentyűzetről. Mindaddig, amíg le nem nyomunk egy billentyűt, az akkumulátor tartalma nulla marad – hasonlóan a GET utasításhoz, ahol ilyenkor üres karaktert kapunk vissza. Egy billentyű lenyomására várakozik az alábbi két utasítás:

```
***       T55.      ***
LOOP    JSR GET
        BEQ LOOP
```

A GET rutin beolvasása is vonatkozhat egy logikai file-ra. A beolvasás előkészítése hasonló a BASIN rutinnál megismert eljáráshoz. Ha a beolvasás egy file-ra vonatkozik, akkor a GET és BASIN rutinok tökéletesen egyformán működnek. A beolvasás végeztével itt is meg kell hívnunk a CLRCH rutint az alaphelyzet visszaállítására.

CLALL – JSR $FFE7

A rutin feladata azonos a CLRCH rutin feladatával, ill. annyiban több annál, hogy hívásakor az alaphelyzet visszaállításán kívül a megnyitott file-ok számát is nullára állítja. Az operációs rendszer ezután minden file-t lezártnak tekint. Valójában a rutin nem zárja le annak rendje s módja szerint a nyitott file-okat, ha előtte azt CLOSE utasítással nem tettük meg.
Pl. egy írásra megnyitott lemezfile lezárása rendellenes lesz, ha előtte CLOSE utasítással nem zártuk le.
Az értelmező a RUN parancs hatására mindig meghívja a CLALL rutint.

LOAD – JSR $FFD5

Az operációs rendszer LOAD rutinjának hívása előtt elő kell készíteni az egységszámot, a másodlagos címet és a file nevét a SETFLS és SETNAM rutinokkal.
A másodlagos cím értékétől függően a rutin a programot vagy arra a címre tölti, amelyet a lemezről beolvas, vagy arra a címre, amit paraméterként átadunk neki.
Ha a másodlagos cím nulla, akkor a tárolási címet az X (alsó byte) és az Y (felső byte) regiszterek tartalma határozza meg. Azt, hogy valódi programbetöltést vagy programellenőrzést (VERIFY) végez a rutin, az akkumulátor tartalma dönti el.

```
***       P54.       ***

LDA #0          ; A kapcsolo erteke betolteshez
LDX #< CIM      ; Kezdocim
LDY #> CIM
JSR LOAD
STX VEGCIM      ; Vegcim also byte
STY VEGCIM+1    ; Vegcim felso byte
```

Ha a másodlagos cím nulla, a rutin a programot a CIM-től tölti be. A betöltött program végcímének alsó és felső byte-ját az X és Y regiszterekben kapjuk vissza. Ha betöltés helyett a tárbeli és a lemezen tárolt programot akarjuk összehasonlítani, akkor hívás előtt az akkumulátorba 1-et kell beírni.

```
***       T56.       ***

LDA #1      ; A VERIFY kapcsoloja
JSR LOAD
```

Ha a másodlagos cím egy, akkor a betöltési címet a rutin a lemezről olvassa be, és ilyenkor nem kell előzetesen az X és Y regiszterek tartalmát rögzíteni. Az ellenőrzés (VERIFY) során felmerülő hibát (a lemezen tárolt és a tárbeli program különbözőséget) a $90-es cím nullától különböző értéke jelzi.
A $90-es cím az ún. állapotjelző változó (státusz).
A program végét az operációs rendszer abból ismeri fel, hogy a státuszban a 6. bit 1-re vált, értéke 64 lesz.

```
***       T57.       ***

    LDA STATUS
    AND #%10111111   ; EOF-bit maszkolasa
    BEQ OK
    JMP ERROR
OK  ...
```

SAVE – JSR $FFD8

A SAVE rutin a tár tetszőleges részét kimásolja egy külső egységre. Hívása előtt a SETFLS és SETNAM rutinokkal elő kell készíteni az egységszámot, a file nevét, és meg kell adni a tárterület kezdő- és végcímét (ill. technikai okokból a végcím + 1-et).
Az utóbbi alsó és felső byte-ját az X és Y regiszterekben kell tárolni.
Az akkumulátorba be kell tölteni egy mutatót, amely a nullaslapon arra a címre mutat, ahol a tárterület kezdőcíme található (alsó és felső byte).
A $1234-től $1FFF-ig terjedő terület tárolására szolgáló programszám:

```
***       P55.       ***

LDA #< $1234
STA START
LDA #> $1234
STA START+1
LDX #< $1FFF+1
LDY #> $1FFF+1
LDA #START
JSR SAVE
```

A program elhelyezi a nullás lap START és START + 1 címén a tárterület kezdőcímét.
Az X és Y regiszterekbe betölt a program végcíménél eggyel nagyobb értéket alsó és felső byte-ra bontva. Végül az akkumulátorba betölti a START kezdőcímet közvetlen címzéssel.
Tárolás közben a DEVICE NOT PRESENT, a MISSING FILENAME, ill. az ILLEGAL DEVICE NUMBER hibaüzenetek léphetnek fel. Az utóbbi pl. akkor, ha kiviteli egységként a billentyűzet vagy az RS–232-es csatornát adjuk meg.
Mielőtt saját input/output rutint írnánk, érdemes közelebbről megvizsgálni az operációs rendszer kernal rutinjainak munkamódszerét.

OPEN

Az OPEN utasítás bejegyzi a paramétereket – a file logikai számát, az egységszámot és a másodlagos címet – egy táblázatba. A táblázat mérete 10 bejegyzésre elegendő, ha ennél több file-t akarunk megnyitni, akkor kiírjuk a TOO MANY FILES (túl sok file) hibaüzenetet és a végrehajtást befejezi.
Ha hiba nem lép fel, akkor a további lépések az egységszámtól függnek. Ha a megnyitás a képernyőre (3-as egységszám) vagy a billentyűzetre (0-s egységszám) vonatkozik, akkor nincs szükség sem a file nevére, sem egyéb teendőre, így a rutin futása véget ér. Amennyiben az input/output művelet pl. szalagegységre (1-es egységszám) vonatkozik, akkor a következő elágazás irányát a másodlagos cím határozza meg.
A 2-es másodlagos cím is írásra utal, de ebben az esetben a CLOSE utasítás végrehajtásában lesz eltérés.
Olvasásnál a rutin megkeresi a szalagon az OPEN utasításban megadott filenevet, ill. ha nem adtunk meg file-nevet, akkor a legelső file-t nyitja meg. Ha az egységszám 2 (RS–232-es csatorna), akkor előkészíti az adatátvitelt, ami az alábbi lépésekből áll.
Kijelöli a BASIC tár felső határán a két, egyenként 256 byte-os input/output puffert (erről már korábban esett szó). A másodlagos címet ebben az esetben nem veszi figyelembe.
Az RS–232-es csatornán megnyitott file neve 4 karakterből állhat, és ezeknek a karaktereknek a műveletvégzésre vonatkozóan külön jelentésük van.
A rendszer az első két karaktert tárolja a $293-as és $294-es címeken. Ezekből meghatározza az átviteli bitek számát (első karakter 5. és 6. bitje), ill. az átviteli sebességet. A bitek számát a $295/$296 címeken tárolja. Az utóbbit betölti a CIA 2 számlálóiba. X-line handshake esetén megvizsgálja, hogy DSR (Data Set Ready) jel érkezett-e.
Hibás jelet észlelve, az RS–232-es állapotjelző regiszterében ($297) a megfelelő bitet bebillenti.
Egyébként az OPEN utasítás végrehajtásakor az állapotjelző regisztert (ST-t) mindig törli.
Ha az egységszám 3-nál nagyobb, akkor az adatátvitel a soros buszra vonatkozik.
Ilyen esetben, ha másodlagos címet és file-nevet nem adtunk meg (mint pl. egy a nyomtatóra vonatkozó OPEN1,4 utasításban), akkor az OPEN utasítás végrehajtása mindössze egy file-bejegyzést jelent.

Ha a másodlagos címet nem adták meg, egy negatív értéket (pl. $FF-et) kell átadni a SETFLS rutinnak.
Egyébként ugyanis a rendszer a soros buszra küldi az OPEN utasítást, úgy, hogy először LISTEN parancsot továbbít az egységnek egy $F0 címmel együtt. A $F0 cím az egység számára OPEN utasítást jelent.
A másodlagos cím feldolgozása után a rendszer a file nevét is továbbítja (ha volt), és egy UNLISTEN paranccsal az adatközlést befejezi.

CLOSE

A CLOSE utasítás befejezi az átvitelt és törli a tárból a file bejegyzését. A rutin következő lépéseit megint az egységszám szabja meg. A billentyűzetre és a képernyőre megnyitott file-ok esetén a rutin a bejegyzés törlése után befejeződik.
Ha azonban az adatátvitel egy külső egységre, pl. szalagegységre irányult, akkor a rutin megvizsgálja a másodlagos címet. Ha a file olvasásra volt nyitva (a másodlagos cím nulla), akkor nincs más teendő, a munka itt ismét véget ér. Írásnál azonban a kazettapuffer tartalmát a tényleges lezárás előtt kiüríti – az adatokat felírja a file-ba és a 2-es másodlagos cím hatására még egy EOT (End of Tape) adatblokkot is felír a szalagra.
RS–232-es adatátvitel esetében a rutin felszabadítja a két puffert.
Soros átvitelnél abban az esetben, ha megadtunk másodlagos címet, a soros buszon át a külső egységhez továbbít egy a másodlagos címnél $E0-val nagyobb értéket, amit az egység CLOSE utasításként értelmez.

CHKIN

Ha file-ból olvasunk adatokat, akkor a gép a logikai file-szám alapján a bejegyzésből meghatározza az egységszámot és a másodlagos címet, és ezek értékétől függően különbözőképpen jár el.
Szalagegység esetén megvizsgálja, hogy a file-t olvasásra nyitottuk-e meg (másodlagos cím: 0), és ha nem, akkor egy hibaüzenetet generál (NOT INPUT FILE).
Ha a soros buszra kapcsolt egységről olvashatunk, a rendszer elküld az egységhez egy TALK utasítást és a másodlagos címet, erre válaszul a külső egység jelzi, hogy felkészült az adatok küldésére.
Ha szükséges, a rendszer tárolja az egységszámot is, hiszen az adatokat csak addig olvashatjuk erről az egységről, amíg az input/output egységek alaphelyzetét a CLRCH utasítással vissza nem állítjuk.

CKOUT

A CKOUT utasítás a CHKIN-hez hasonlóan működik.
Szalagegység esetében megvizsgálja, hogy a másodlagos cím nagyobb-e mint nulla (egyébként kiírja a NOT OUTPUT FILE hibaüzenetet). A soros buszra elküldi a LISTEN utasítást és a másodlagos címet – válaszul a külső egység jelzi, hogy készen áll az adatok fogadására.

BASIN

A rutin beolvas egy karaktert a CHKIN utasítással kiválasztott, éppen aktív egységről, vagy alaphelyzetben a billentyűzetről. A beolvasott karaktert az akkumulátorban tárolja.

BSOUT

A rutin az akkumulátor tartalmát (egy karaktert) kiírja az előzetesen CKOUT rutinnal aktivizált egységre. Az egység alapértelmezésben a képernyő.

CLRCH

A CLRCH utasítás az input/output egységek korábbi kijelölését megszünteti, és az alaphelyzetet visszaállítja.
Az input egység számát 0-ra (billentyűzet), az output egység számát pedig 3-ra (képernyő) állítja.
Ha a soros busz egységei is aktívak voltak, a rendszer a buszra egy UNTALK, ill. UNLISTEN üzenetet is küld, jelezve ezzel, hogy az adatátvitel befejeződött.

3.6 RENDSZERFÜGGETLEN NYOMTATÁS (SPOOLING)

Az operációs rendszer input/output vektorainak alkalmazását egy nyomtatási feladaton keresztül mutatjuk be. A nyomtatót egy Centronics illesztővel csatlakoztathatjuk az alapgép user portjára.
Rendszerfüggetlen nyomtatásról (spooling) abban az esetben beszélünk, amikor a karakterek kiírása az éppen futó program munkájától független, ún. háttértevékenység.
A meghatározásból kiderül, hogy ezt a feladatot csak a megszakítási rutin igénybevételével oldhatjuk meg. Az egyszerű PRINT utasítást a rendszer úgy hajtja végre, hogy minden karakter kiírása előtt addig várakozik, amíg a nyomtató készen nincs az adat fogadására. A rendszerfüggetlen nyomtatásánál a várakozást úgy kerüljük ki, hogy a kiírandó karaktereket egy pufferben tároljuk, ahelyett, hogy PRINT utasítással kinyomtatnánk.
Az általunk készített megszakítási rutin – amit összekapcsolunk a rendszermegszakításokkal – megvizsgálja, hogy van-e a pufferban kiírandó karakter. Ha igen, akkor a rutin folyamatosan nyomtatja a pufferben talált karaktereket mindaddig, amíg a puffer ki nem ürült, vagy míg a nyomtató nem jelzi, hogy nem tud több adatot fogadni.

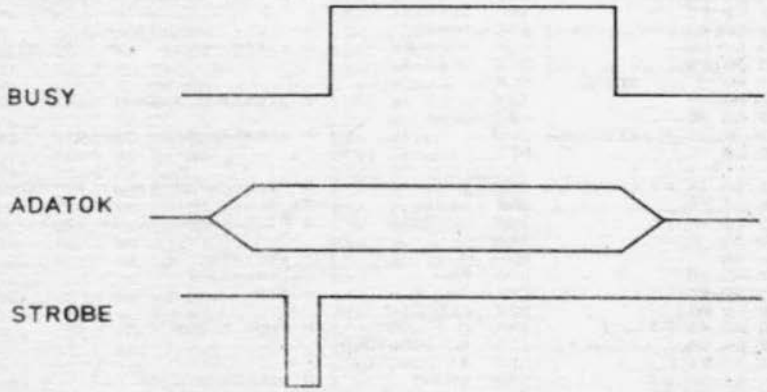
```
***      P56.      ***
PROFI-ASS 64 V2.0

100:   CC00                    .OPT P,00
110:                    ;
120:                    ; Printer spooling
130:                    ;
140:                    ; I/O vektorok
150:   031A             OPEN    =    $31A       ; OPEN vektor
160:   031C             CLOSE   =    $31C       ; CLOSE vektor
190:   0326             BSOUT   =    $326       ; BSOUT vektor
200:                    ;
210:   00F7             WPNT    =    $F7        ; Irasmutato a pufferben
220:   00F9             RPNT    =    $F9        ; Olvasasi mutato a pufferben
230:                    ;
240:   0098             NRFLS   =    $98        ; A nyitott file-ok szama
250:   00B8             LF      =    $B8        ; Logikai file-szam
260:   00BA             FA      =    $BA        ; Egysegszam
270:   00B9             SA      =    $B9        ; Masodlagos cim
280:   0259             LFTAB   =    $259       ; A log.file-szamok tablazata
290:   0263             FATAB   =    LFTAB+10   ; Az egysegszamok tablazata
300:   026D             SATAB   =    FATAB+10   ; A masodlagos cimek tablazata
310:   009E             CHAR    =    $9E        ; A kiirando karakter
320:   0001             KONFIG  =    1          ; Tarfelosztas
330:   009A             OUTDEV  =    $9A        ; Egysegszam kiirashoz
340:   0314             IRQVEK  =    $314       ; IRQ - vektor
350:   EA31             IRQALT  =    $EA31      ; A regi IRQ rutin
360:                    ;
370:   F34A             OPENOLD =    $F34A
380:   F1CA             BSOUTOLD =   $F1CA
390:   F31F             SETPARA =    $F31F
400:   F314             SUCHLF  =    $F314
410:   F30F             SUCHLFX =    $F30F
420:   F2A1             OLDCLOSE =   $F2A1
430:   F2F1             CONTCLS =    $F2F1
440:   F6FE             FILEOPEN =   $F6FE
450:   F64B             TOOMANY =    $F64B
```

```
460:  F291           CLOSEOLD =   $F291
470:  DD00           CIA      =   $DD00      ; CIA2
480:  DD00           PORTA    =   CIA        ; PA2 STROBE
490:  DD01           PORTB    =   CIA+1      ; B port az adatok szamara
500:  DD03           RICHTUNG =   CIA+3      ; Adatiranyregiszter
510:  DD0D           ICR      =   CIA+13     ; Interrupt control regiszter
520:                 ;
530:  E000           PUFFER   =   $E000      ; Nyomtato-puffer a kernal ala
540:                 ;
550:  CC00                    *=   $CC00
560:  CC00 A9 0B     INIT     LDA  #< OPENNEW
570:  CC02 A0 CC              LDY  #> OPENNEW
580:  CC04 8D 1A 03           STA  OPEN       ; Az OPEN vektor atallitasa
590:  CC07 8C 1B 03           STY  OPEN+1
600:  CC0A 60                 RTS
610:                 ;
620:  CC0B A6 B8     OPENNEW  LDX  LF         ; Logikai FILE-szam
630:  CC0D F0 05              BEQ  ERROR      ; 0 nem megengedett
640:  CC0F 20 0F F3           JSR  SUCHLFX    ; A FILE-adatok keresese
650:  CC12 D0 03              BNE  OK2        ; Ha nincsenek meg => OK
660:  CC14 4C FE F6 ERROR     JMP  FILEOPEN   ; Egyebkent "FILE OPEN ERROR"
670:  CC17 A6 98     OK2      LDX  NRFLS      ; A nyitott FILE-ok szama
680:  CC19 E0 0A              CPX  #10
690:  CC1B 90 03              BCC  OK         ; Ha kevesebb, mint 10 => OK
700:  CC1D 4C 4B F6           JMP  TOOMANY    ; "TOO MANY FILES"
710:  CC20 A5 BA     OK       LDA  FA         ; Egysegszam
720:  CC22 C9 04              CMP  #4         ; = 4 ?
730:  CC24 F0 03              BEQ  SPOOL      ; Igen, SPOOLING
740:  CC26 4C 4A F3           JMP  OPENOLD
750:  CC29 E6 98     SPOOL    INC  NRFLS      ; A szam novelese
760:  CC2B 9D 63 02           STA  FATAB,X    ; Egysegszam a tablazatban
770:  CC2E A5 B8              LDA  LF
780:  CC30 9D 59 02           STA  LFTAB,X    ; Logikai FILE-szam
790:  CC33 A9 FF              LDA  #-1
800:  CC35 9D 6D 02           STA  SATAB,X    ; Nincs egysegszam
810:  CC38 A9 E0              LDA  #> PUFFER
820:  CC3A 85 FB              STA  WPNT+1     ; Irasmutato
830:  CC3C 85 FA              STA  RPNT+1     ; es olvasasi mutato
840:  CC3E A9 00              LDA  #0         ; a puffer elejere
850:  CC40 85 F7              STA  WPNT
860:  CC42 85 F9              STA  RPNT
870:  CC44 A9 FF              LDA  #$FF
870:  CC46 8D 03 DD           STA  RICHTUNG   ; USER-port kiirashoz
880:  CC49 AD 00 DD           LDA  PORTA
890:  CC4C 09 04              ORA  #%100      ; STROBE felso byte
900:  CC4E 8D 00 DD           STA  PORTA
910:  CC51 A9 B5              LDA  #< BSOUTNEW
920:  CC53 A0 CC              LDY  #> BSOUTNEW
930:  CC55 8D 26 03           STA  BSOUT      ; BSOUT vektor az uj rutinra
940:  CC58 8C 27 03           STY  BSOUT+1
950:  CC5B A9 DD              LDA  #< CLOSENEW
960:  CC5D A0 CC              LDY  #> CLOSENEW
970:  CC5F 8D 1C 03           STA  CLOSE      ; CLOSE vektor az uj rutinra
980:  CC62 8C 1D 03           STY  CLOSE+1
990:  CC65 A9 73              LDA  #< SPOOLING
1000: CC67 A0 CC              LDY  #> SPOOLING
1010: CC69 78                 SEI
1020: CC6A 8D 14 03           STA  IROVEK     ; IRQ-vektor SPOOL-rutinbol
1030: CC6D 8C 15 03           STY  IROVEK+1
1040: CC70 58                 CLI
1040: CC71 18                 CLC             ; Hibakapcsolo torlese
1050: CC72 60                 RTS
1060:                ;
1070: CC73 A5 01     SPOOLING LDA  KONFIG
1080: CC75 48                 PHA
1090: CC76 A9 35              LDA  #$35       ; A RAM kivalasztasa
1100: CC78 85 01              STA  KONFIG
1110: CC7A A5 F9     TESTNEXT LDA  RPNT       ; Iras-, olvasasmutato
1120: CC7C C5 F7              CMP  WPNT       ; Osszehasonlitas
1130: CC7E D0 06              BNE  SENDCHAR   ; Ha <>, => a karakter kiirasa
```

```
1140:  CC80 A5 FA          LDA  RPNT+1
1150:  CC82 C5 FB          CMP  WPNT+1
1160:  CC84 F0 29          BEQ  EXIT
1170:  CC86 A9 10    SENDCHAR LDA  #%10000     ; Bitmaszk a FLAG-bemenethez
1180:  CC88 2C 0D DD       BIT  ICR           ; A nyomtato kesz ?
1190:  CC8B F0 22          BEQ  EXIT          ; Nem
1200:  CC8D A0 00          LDY  #0
1210:  CC8F B1 F9          LDA  (RPNT),Y      ; A kiirando karakter
1220:  CC91 8D 01 DD       STA  PORTB         ; A port-ra
1230:  CC94 AD 00 DD       LDA  PORTA
1240:  CC97 29 FB          AND  #%11111011 ; STROBE also byte
1250:  CC99 8D 00 DD       STA  PORTA
1260:  CC9C 09 04          ORA  #%00000100 ; es felso byte
1270:  CC9E 8D 00 DD       STA  PORTA
1280:  CCA1 E6 F9          INC  RPNT
1280:  CCA3 D0 D5          BNE  TESTNEXT      ; Az olvasasi mutato
1290:  CCA5 E6 FA          INC  RPNT+1
1290:  CCA7 D0 D1          BNE  TESTNEXT
1300:  CCA9 A9 E0          LDA  #> PUFFER
1310:  CCAB 85 FA          STA  RPNT+1
1320:  CCAD D0 CB          BNE  TESTNEXT      ; A kovetkezo karakter kuldese
1330:                ;
1340:  CCAF 68       EXIT  PLA
1350:  CCB0 85 01          STA  KONFIG        ; A regi tarfelosztas
1360:  CCB2 4C 31 EA       JMP  IRQALT        ; Ugras a regi IRQ-ra
1370:                ;
1380:  CCB5 48       BSOUTNEW PHA             ; A karakter tarolasa
1390:  CCB6 A5 9A          LDA  OUTDEV        ; Egysegszam
1400:  CCB8 C9 04          CMP  #4            ; = 4 ?
1410:  CCBA F0 04          BEQ  OK1           ; Igen
1420:  CCBC 68             PLA
1420:  CCBD 4C CA F1       JMP  BSOUTOLD      ; Ugras a regi kiirasra
1430:  CCC0 68       OK1   PLA                ; A karakter visszatoltese
1440:  CCC1 85 9E          STA  CHAR          ; es tarolasa
1450:  CCC3 98             TYA
1450:  CCC4 48             PHA                ; Y mentese
1460:  CCC5 A5 9E          LDA  CHAR          ; A karakter
1470:  CCC7 A0 00          LDY  #0
1480:  CCC9 91 F7          STA  (WPNT),Y      ; kiirasa a pufferba
1490:  CCCB E6 F7          INC  WPNT
1500:  CCCD D0 08          BNE  NOINC         ; A puffer-mutato novelese
1510:  CCCF E6 F8          INC  WPNT+1
1520:  CCD1 D0 04          BNE  NOINC
1530:  CCD3 A9 E0          LDA  #> PUFFER     ; A puffer-mut. a puffer elejere
1540:  CCD5 85 F8          STA  WPNT+1
1550:  CCD7 68       NOINC PLA
1560:  CCD8 A8             TAY                ; Y visszaolvasasa
1570:  CCD9 A5 9E          LDA  CHAR
1580:  CCDB 18       KESZ  CLC                ; A hibakapcsolo torlese
1580:  CCDC 60             RTS
1590:                ;
1600:  CCDD 20 14 F3 CLOSENEW JSR  SUCHLF      ; A FILE-parameterek keresese
1610:  CCE0 D0 F9          BNE  KESZ          ; Ha nincs FILE nyitva, => kesz
1620:  CCE2 20 1F F3       JSR  SETPARA       ; A FILE-parameterek beolvasasa
1630:  CCE5 8A             TXA
1630:  CCE6 48             PHA                ; X mentese
1640:  CCE7 A5 BA          LDA  FA            ; Egysegszam
1650:  CCE9 C9 04          CMP  #4            ; = 4 ?
1660:  CCEB F0 03          BEQ  CLOSE1
1670:  CCED 4C A1 F2       JMP  OLDCLOSE      ; A regi CLOSE rutin
1680:  CCF0 A9 CA    CLOSE1 LDA  #< BSOUTOLD
1690:  CCF2 A2 F1          LDX  #> BSOUTOLD
1700:  CCF4 8D 26 03       STA  BSOUT         ; A vektor a regi rutinra
1710:  CCF7 8E 27 03       STX  BSOUT+1
1720:  CCFA A9 91          LDA  #< CLOSEOLD
1730:  CCFC A2 F2          LDX  #> CLOSEOLD
1740:  CCFE 8E 1A 03       STA  CLOSE         ; A vektor a regi CLOSE rutinra
1750:  CD01 8E 1B 03       STX  CLOSE+1
1760:  CD04 A9 31          LDA  #< IRQALT
1770:  CD06 A2 EA          LDX  #> IRQALT
```

```
1780:  CD08 78          SEI
1790:  CD09 8D 14 03    STA  IRQVEK      ; A regi IRQ helyreallitasa
1800:  CD0C 8E 15 03    STX  IRQVEK+1
1810:  CD0F 58          CLI
1820:  CD10 4C F1 F2    JMP  CONTCLS     ; CLOSE befejezese
 CC00-CD13
NO ERRORS
```

A rutin működését könnyebben megértjük, ha van némi alapismeretünk a Centronics illesztőről.
A Centronics egy párhuzamos illesztő, egyszerre 8 bit, azaz 1 byte továbbítására alkalmas.
A számítógép és a nyomtató közötti kapcsolatot két, ún. handshake vonal teremti meg.
Az egyiket, a STROBE vonalat a gép szolgálja ki.
A vonal nyugalmi állapotban magas jelszinten áll.
A gép nyomtatáskor egy byte-ot kihelyez az adatvonalra, és a STROBE alacsonyra állításával közli, hogy az adatok nyomtatásra készen állnak.
A nyomtató átveszi az adatokat, a másik handshake vonalat (BUSY) magas jelszintre állítja, és mindaddig ezen a szinten tartja, míg fel nem dolgozta az adatokat, azaz amíg nem áll készen a következő adat fogadására. A gép minden újabb adat elküldése előtt megvizsgálja, hogy a BUSY vonal alacsony szintű-e.
Az adatok továbbítására a C64-es a CIA 2-t használja. Az adatforgalom a B porton (user port) át zajlik le. A STROBE jelet a PA 2-es kimeneten (az A port 2. bitje) állítja elő. A nyomtató BUSY vonala a user port FLAG kimenetére csatlakozik. Magas jelszintről alacsonyra váltás közben a CIA megszakításvezérlő regiszterének (ICR) 4. bitje automatikusan 1 lesz. Ebből ismerjük fel, hogy a nyomtató készen áll az adatfogadásra. Az alábbi idődiagram a leírtakat szemlélteti.



A fenti rövid áttekintés után elemezzük a programot:
A címek rögzítése után az OPEN rutin vektorát átirányítjuk a saját rutinunkra.
A rutin a rendszerrutinokhoz hasonlóan a logikai file-szám ellenőrzésével

kezdődik. Ha a file-szám nulla, hibajelzést küld, egyébként megkeresi a hozzá tartozó file-t. Ha nincs ilyen számmal nyitott file, akkor ellenőrzi, hogy a nyitott file-ok száma elérte-e a tizet. Ha igen, akkor megtelt a táblázat, így kiírja a TOO MANY FILE hibaüzenetet, egyébként folytatja a feldolgozást.
Megvizsgálja az egységszámot, és ha az nem 4, akkor visszatér a rendszer OPEN rutinjára.
Egyébként megnöveli eggyel a nyitott file-ok számát, bejegyzi a file logikai számát, az egységszámot és a másodlagos címet a táblázatba, és a puffermutatót a puffer elejére állítja. Pufferként az operációs rendszer alatti, $E000-tól $FFFF-ig terjedő 8 kbyte-os területet használjuk.
Ezután az user portot kimenetre kapcsoljuk és a STROBE jelet magas szintre állítjuk.
A BSOUT és CLOSE rutinok vektorait úgy módosítjuk, hogy az a saját rutinjainkra mutasson. A nyomtatás a megszakítási pillanatokban történik, ezért megváltoztatjuk a megszakítási vektort, az új rutin a SPOOLING címnél kezdődik.
A C kapcsoló törlése után RTS utasítással befejezzük a nyomtatást.
A SPOOLING rutin, amely össze van kapcsolva a rendszermegszakításokkal, átkapcsolja a tárfelosztást a RAM-ra, majd megvizsgálja, hogy van-e a pufferben nyomtatásra váró adat.
Az igen választ onnan észleli, hogy az írásmutató – amit a BSOUT rutin minden írási művelet után megnövel – nem egyezik meg a kiviteli mutatóval. Ha eközben a nyomtató kész az adatok fogadására, a rutin beolvas egy byte-ot a pufferből, és kihelyezi azt a user portra. Átkapcsolja a STROBE jelet alacsony, majd ismét magas jelszintre, ezzel jelezve a nyomtatónak, hogy egy karakter nyomtatásra vár.
Végül megnöveli az olvasási mutatót, hogy a következő lépésben a puffer következő adatára kerüljön sor.
Az aktuális karakter feldolgozása után visszatérünk a rutin kezdetére, és ez a ciklus mindaddig ismétlődik, míg el nem fogynak az adatok, vagy amíg a nyomtató már nem képes több adatot fogadni.
Az EXIT címkével visszaállítjuk az eredeti tárfelosztást és a rendszer megszakítási rutinját.
A BSOUTNEW rutin vizsgálja meg, hogy a kiírás a 4-es egységre vonatkozik-e. Ha igen, akkor az adatot beírja a pufferba és a puffermutatót eggyel megnöveli. A rutin nem változtatja meg a regiszterek tartalmát, és visszatérés előtt törli a C kapcsolót, ezzel jelezve, hogy a művelet hibamentes volt.
A CLOSENEW rutin szintén csak akkor dolgozik, ha az egységszám 4. Ekkor visszaállítja a BSOUT és CLOSE rutinok vektorainak és a megszakítási vektor eredeti értékét. Ezzel a pufferban tárolt adatok kiírása megszakad. Ha nem akarjuk, hogy ilyenkor adatok elvesszenek, akkor be kell iktatnunk egy várakozó ciklust, amely addig vár, amíg a puffer írás- és olvasásmutatója egyenlő nem lesz.
Végül megjegyezzük, hogy a nyomtatáshoz szükség van egy kábelra, amely a C64-es user portját pl. egy EPSON nyomtató Centronics illesztőjével összeköti.
A kábel felépítése:

```
***        T57/1.        ***
USER PORT          -          CENTRONICS
   A              GND            16
   B         FLAG - BUSY         11
   C              D0              2
   D              D1              3
   E              D2              4
   F              D3              5
   H              D4              6
   J              D5              7
   K              D6              8
   L              D7              9
   M        PA2   - STROBE        1
```

A legtöbb Centronics illesztővel ellátott nyomtató ASCII karakterekkel dolgozik. Ha a karakterek között van olyan, amely a Commodore 64-es jelkészletből hiányzik, akkor kiírás előtt még egy átkódolást is be kell iktatni.
Az üzemeltetésnél a következőkre kell ügyelni:
csatlakoztassuk a gépet és a nyomtatót, majd kapcsoljuk be őket ebben a sorrenben: először a gépet, aztán a nyomtatót. Ez a sorrend garancia arra, hogy a nyomtató a READY állapotba való átmenetkor a CIA FLAG bitjét beállítja. Most töltsük be a gépi kódú rutint, és SYS 52224 utasítással indítsuk el.
Ha most OPEN 1,4 és PRINT # utasításokat írunk a BASIC programba, a gép az adatokat a pufferba írja, és a puffer tartalmát a megszakítások közben kinyomtatja. A puffer feltöltése nagyon gyors, így lehet, hogy a BASIC program már rég lefutott, miközben a megszakítási rutin még a puffer tartalmának kinyomtatásával foglalatoskodik.

4.1 Kulcsszavak es tokenjeik tablazata

Token	Utasitas	Cim	Token	Utasitas	Cim
$80 128	END	$A831	$9F 159	OPEN	$E1BE
$81 129	FOR	$A742	$A0 160	CLOSE	$E1C7
$82 130	NEXT	$AD1E	$A1 161	GET	$AB7B
$83 131	DATA	$A8F8	$A2 162	NEW	$A642
$84 132	INPUT#	$ABA5	$A3 163	TAB(	-
$85 133	INPUT	$ABBF	$A4 164	TO	-
$86 134	DIM	$B081	$A5 165	FN	-
$87 135	READ	$AC06	$A6 166	SPC(	-
$88 136	LET	$A905	$A7 167	THEN	-
$89 137	GOTO	$A8A0	$A8 168	NOT	-
$8A 138	RUN	$A871	$A9 169	STEP	-
$8B 139	IF	$A928	$AA 170	+	$B864
$8C 140	RESTORE	$A81D	$AB 171	-	$B853
$8D 141	GOSUB	$A883	$AC 172	*	$BA2B
$8E 142	RETURN	$A8D2	$AD 173	/	$BB12
$8F 143	REM	$A93B	$AE 174	^	$BF7B
$90 144	STOP	$A82F	$AF 175	AND	$AFE9
$91 145	ON	$A94B	$B0 176	OR	$AFE6
$92 146	WAIT	$B82D	$B1 177	>	-
$93 147	LOAD	$E168	$B2 178	=	-
$94 148	SAVE	$E156	$B3 179	<	-
$95 149	VERIFY	$E165	$B4 180	SGN	$BC39
$96 150	DEF	$B3B3	$B5 181	INT	$BCCC
$97 151	POKE	$B824	$B6 182	ABS	$BC58
$98 152	PRINT#	$AA80	$B7 183	USR	$0310
$99 153	PRINT	$AAA0	$B8 184	FRE	$B37D
$9A 154	CONT	$A69C	$B9 185	POS	$B39E
$9B 155	LIST	$A69C	$BA 186	SQR	$BF71
$9C 156	CLR	$A65E	$BB 187	RND	$E097
$9D 157	CMD	$AA86	$BC 188	LOG	$B9EA
$9E 158	SYS	$E12A	$BD 189	EXP	$BFED
$BE 190	COS	$E264	$BF 191	SIN	$E26B
$C0 192	TAN	$E2B4	$C1 193	ATN	$E30E
$C2 194	PEEK	$B80D	$C3 195	LEN	$B77C
$C4 196	STR$	$B465	$C5 197	VAL	$B7AD
$C6 198	ASC	$B78B	$C7 199	CHR$	$B6EC
$C8 200	LEFT$	$B700	$C9 201	RIGHT$	$B72C
$CA 202	MID$	$B737	$CB 203	GO	-

A táblázatban először az önálló utasításszavakat soroltuk fel ($80–$A2), majd azokat, amelyek mindig egy másik utasítással együtt fordulnak elő ($A3–$B0). Ezt követik az algebrai és logikai ($AA–$B0), majd az összehasonlító műveletek ($B1–$B3), végül a BASIC függvények ($B4–$CA).
A táblázat utolsó sora a GO kódja, ugyanis a GOTO utasítás GO TO formája is megengedett.
Az utasításszavak mögött, ahol szükséges, feltüntettük a megfelelő ROM rutin kezdőcimét.

4.2 Nullaslap osszehasonlito tablazat

MEGNEVEZES	C64	C128
LOAD/VERIFY-KAPCS.	$0A	$0C
Tipuskapcsolo	$0D	$0F
INTEGER kapcsolo	$0E	$10
BASIC prg. kezdet	$2B/$2C	$2D/$2E
BASIC valt. kezdet	$2D/$2E	$2F/$30
BASIC tomb kezdet	$2F/$30	$31/$32
BASIC tomb vege	$31/$32	$33/$34
BASIC szov. kezdet	$33/$34	$35/$36
BASIC RAM vege	$37/$38	$39/$3A
Aktualis sorsz.	$39/$3A	$3B/$3C
FAC#1	$61-$66	$63-$68
FAC#2	$69-$6E	$6A-$6F
CHRGET	$70	$380
CHRGOT	$76	$386
TXTPTR	$77/$78	$3D/$3E
SR SYS-nel	$30F	$05
AC SYS-nel	$30C	$06
XR SYS-nel	$30D	$07
YR SYS-nel	$30E	$08
USR-vektor	$311/$312	$1219/$121A

Amint az a táblázatból látható, a 64-esre vonatkozó címekből úgy kapjuk meg a megfelelő 128-as címet, hogy ahhoz kettőt hozzáadunk.
Ez érvényes a BASIC értelmező mutatóira is a CHRGET rutinig.

– JEGYZET –

Megnyílt a NOVOTRADE Rt. COMMODORE User's klubja!

SZOLGÁLTATÁSAINK:

- klubtagoknak szabad gépidőt és szakirodalmat biztosítunk.
- közületeknek és magánszemélyek részére C 16-os, C 64-es programozói, valamint speciális tanfolyamokat szervezünk.

RÉSZLETES FELVILÁGOSÍTÁS:

2c

Számítástechnikai szaküzlet

1136 Budapest, Balzac u. 35.
Telefon: 402–954
Gábriel László klubvezetőnél

Ára: 319.– Ft

SZÁMÍTÁSTECHNIKA A KÖNYVESBOLTOKBAN

NOVOTRADE – 2C ÁRUHÁZ

1136 Bp., Balzac u. 35. Tel.: 402-954

Az alább felsorolt üzletekben már az Önök rendelkezésére állunk:

ÁLLAMI KÖNYVTERJESZTŐ V. – NOVOTRADE 2C

BUDAPEST

Táncsics Könyvesbolt
1073 Lenin krt. 17.
Telefon: 422-178

Műszaki Könyváruház
1061 Liszt Ferenc tér 9.
Telefon: 420-353

MŰVELT NÉP KÖNYVTERJESZTŐ V. – NOVOTRADE 2C

PÉCS

Zrínyi Miklós Könyvesbolt
7621 Jókai u. 25.
Telefon: 72-12835

SZEGED

Tömörkény Könyvesbolt
6720 Lenin krt. 48.
Telefon: 62-21453

SZOMBATHELY

Savaria Könyvesbolt
9700 Mártírok tere 1.
Telefon: 94-12341

VESZPRÉM

Kölcsey Ferenc Könyvesbolt
8200 Kossuth L. u. 8.

DEBRECEN

Szak- és Ismeretterjesztő Könyváruház
4024 Hunyadi u. 8.
Telefon: 52-23237

SZOLNOK

Szigligeti Könyvesbolt
5000 Ságvári krt. 35.
Telefon: 56-11133

Minden érdeklődőt szeretettel vár
az ÁKV, a Művelt Nép és a NOVOTRADE RT.!