CZIGLER ZOLTÁN

A COMMODORE 64 PROGRAMOZÁSÁNAK GYAKORLATA

GÉPI KÓDÚ PROGRAMOZÁS

4

SZÁMALK

A COMMODORE-64 PROGRAMOZÁSÁNAK GYAKORLATA IV.

Gépi kódú programozás

COMMODORE PROGRAMMING IN PRACTICE IV.

Assembly

ZOLTÁN CZIGLER

COMMODORE

PROGRAMMING IN PRACTICE

ASSEMBLY

Fourth volume

COMPUTING APPLICATIONS AND SERVICE CO. BUDAPEST, 1989

CZIGLER ZOLTÁN

A COMMODORE 64 PROGRAMOZÁSÁNAK GYAKORLATA

GÉPI KÓDÚ PROGRAMOZÁS

Negyedik kötet

SZÁMÍTÁSTECHNIKA-ALKALMAZÁSI VÁLLALAT BUDAPEST, 1989

COMMODORE–64 SOROZAT

Lektorálta: *Gerő Péter*
Supervised by *Péter Gerő*

Kertész Attila illusztrációival

ISBN 963 553 115 X (összkiadás)
ISBN 963 553 152 9 (IV. kötet)

Sylvester János Nyomda, Szombathely
Felelős vezető : Hanuszek Béla igazgató

Kiadja : a Számítástechnika-Alkalmazási Vállalat
Felelős kiadó : Havass Miklós
Felelős szerkesztő : Lukács Erzsébet
Műszaki szerkesztő: G. Müller Zsuzsa
A fedelet tervezte: Molnár Zoltán
Megjelent 18,50 (A/5) ív terjedelemben

TARTALOM

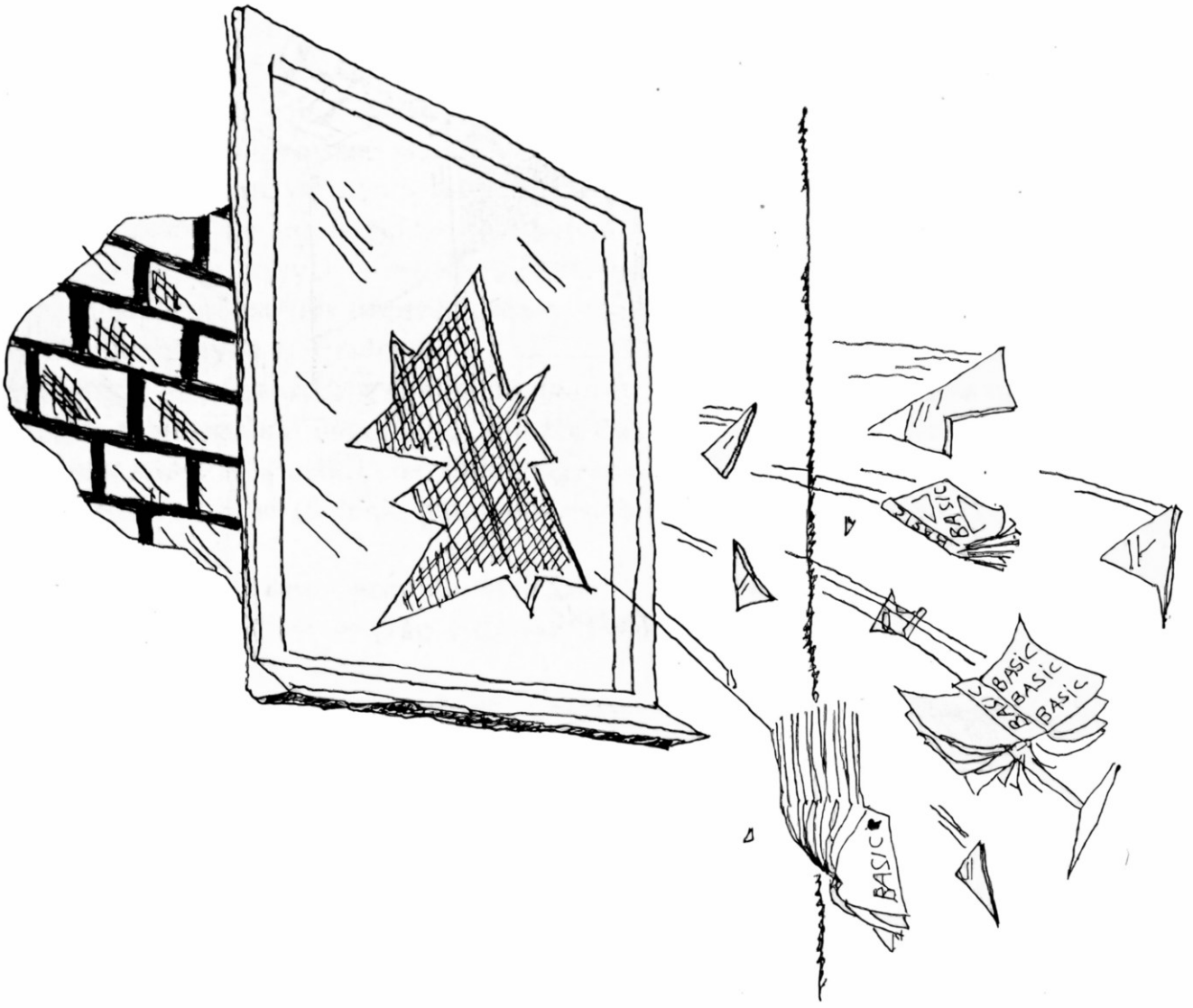
AJÁNLÁS

Mindenkivel megtörténik egyszer, hogy elkészül élete főműve, a nagy gonddal és körültekintéssel megírt, humorral telített, akciódús játékprogram, és az egész család, az egész rokonság, sőt még a család hűségben és házőrzésben megőszült öreg kutyája is ott áll a számítógép előtt, hogy láthassa azt, amit történelmünk során csak kevesen láthattak.

A család büszkesége fölényes mosollyal, szinte oda sem figyelve gépeli be, hogy

RUN, majd a hatás kedvéért tart egy kis szünetet, mielőtt lenyomná a RETURN billentyűt, és akkor:

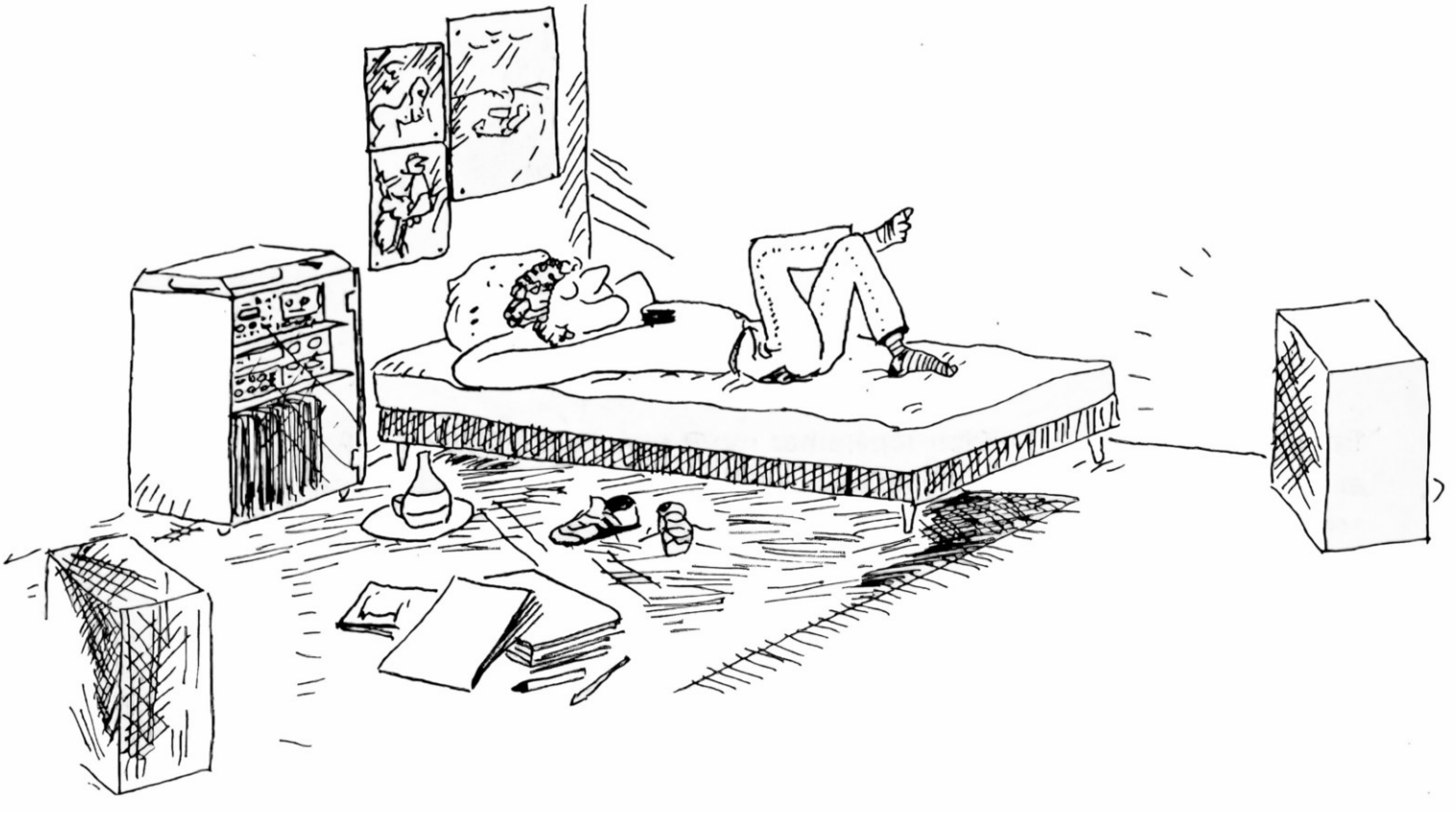
— 14 perc 30 másodpercig tart, amíg a gép kirajzolja a kezdő ábrát,
— további 8 perc 15 másodperc, mire az összes űrhajó indulásra kész.
— A játék 135. percében a kutya elkezd vonyítani, mert elmarad az esti sétáltatás.
— A 174. percben a nagypapa ülve elalszik,
— a 211. percben 3 UFO még él,
— a 243. percben a család büszkesége feláll a gép mellől, kidobálja az ablakon az összes Basic tankönyvét, és elhatározza, hogy megtanul gépi kódban programozni.



Ennek a tanulásnak a kezdeti lépéseihez nyújt segítséget könyvünk. Ajánljuk mindazoknak, akik szedtek már nyugtató pirulákat azért, mert Basic programjaik az egyszerű és gyors gépi műveleteket nyakatekerten és éjszakába nyúló lassúsággal oldották meg.

A szerző

és a rajzoló



ELŐSZÓ

Ez a könyv a C–64 gépi kódú, illetve assembly szintű programozásáról szól. Az assembly-programozás a Basicnél lényegesen gépközelibb, ezért kicsit körülményesebb, ám igen hatékony módja a C–64 programozásának. A programozás körülményessége és nehézkessége busásan megtérül, ha azt nézzük, hogy assembly nyelven megírt programunk mennyivel több lehetőséget ad a gép kedvező tulajdonságainak kihasználására, ugyanakkor mennyivel gyorsabban fut, és mennyivel kisebb memóriaterületet igényel, mint egy ugyanerre a feladatra írt Basic program. A C–64 géphez készült megannyi népszerű játék- és rendszerprogram is szinte kivétel nélkül mind assembly nyelven íródott.

A gépi kódú vagy assembly szintű programozáshoz legalább annyira nem szükséges felsőfokú matematikai végzettség, mint a Basichez, csupán logikus gondolkodás és következetesség. Könyvünk feltételez ugyan egy kevés Basic alapismeretet, de nem kell túlzottan gyakorlottnak lenni a Basicben ahhoz, hogy bevezethessen a gépi kódba.

Ennél többre ez a könyv nem is vállakozik. Elvezeti az olvasót addig a pontig, amíg megérzi ennek a sajátos és szép világnak a lényegét, és ahonnan már egyedül tanulhat tovább.

Javaslom, hogy a könyvben közölt példaprogramokat a szemléletesség kedvéért mindenki futtassa le, és azokon keresztül próbálja megérteni az adott utasítások szerepét és alkalmazását!

Ezen a helyen szeretnék köszönetet mondani a grafikusnak és egyben kedves barátomnak, Kertész Attilának, aki értékes ötleteivel és rajzaival alkotó módon járult hozzá a könyv megjelenéséhez. Ugyanitt mondok köszönetet a lektornak, Gerő Péternek, aki észrevételeivel és megjegyzéseivel nagymértékben segítette munkámat.

Budapest, 1987.

A szerző

BEKAPCSOLÁS

Ez a fejezet azt a minimumot tartalmazza, amit a gépkezelésről és a Basic nyelvről feltétlenül ismerni kell. Az itt leírt tudnivalókhoz semmiféle magyarázat nem tartozik, a magyarázatokat mindenki megtalálhatja a könyvsorozat „Alapismeretek" c. kötetében. Ugyanott olvasható sok olyan fogás is, amelyek használatával megkönnyíthetjük a munkánkat.

A központi egységet a tápegységgel és a TV-készülékkel összekapcsolni könnyű: minden vezetékvéget oda kell csatlakoztatni, ahová ellenállás nélkül sikerül. A központi egység kapcsolója a jobb oldali falán van, a tápegység csatlakozóaljzata mellett. Ha a tápegységünkön külön kapcsoló van, ne felejtsük el ezt is bekapcsolni (és a gép használata után kikapcsolni, különben akkor is fogyasztja az áramot, és melegszik, ha a gép nincs bekapcsolva).

Ha a központi egység kap áramot, és bekapcsoljuk, és ha a vele összekapcsolt TV-készülékünk be van állítva a 36-os csatornára, kb. 2 másodperc alatt megjelenik a felirat, melyben tudatja, hogy ő a Commodore 64, és várja parancsainkat.

```
    **** COMMODORE 64 BASIC V2 ****
 64K RAM SYSTEM  38911 BASIC BYTES FREE
READY.
■
```

Adatot a memóriában például **POKE 50000 , 15** paranccsal helyezhetünk el. Ha ezt begépeljük (a POKE szó után szóközzel vagy anélkül), és lenyomjuk a jobb oldalt levő széles RETURN gombot, akkor a gép az ötvenezredik memóriarekeszbe beteszi a 15-ös számot. A parancs végrehajtása után a gép ismét kiírja a READY üzenetet.

A parancs formája:
POKE memóriarekesz sorszáma (cím) vessző érték RETURN-gomb

Adatot a memóriából például a **? PEEK(50000)** paranccsal olvashatunk. Ha ezt begépeljük (a kérdőjel, illetve a PEEK szó után szóközzel vagy anélkül), és lenyomjuk a RETURN-t, akkor a gép a képernyőre írja az ötvenezredik memóriarekeszben levő adatot. A parancs végrehajtása után a gép ismét kiírja a READY üzenetet.

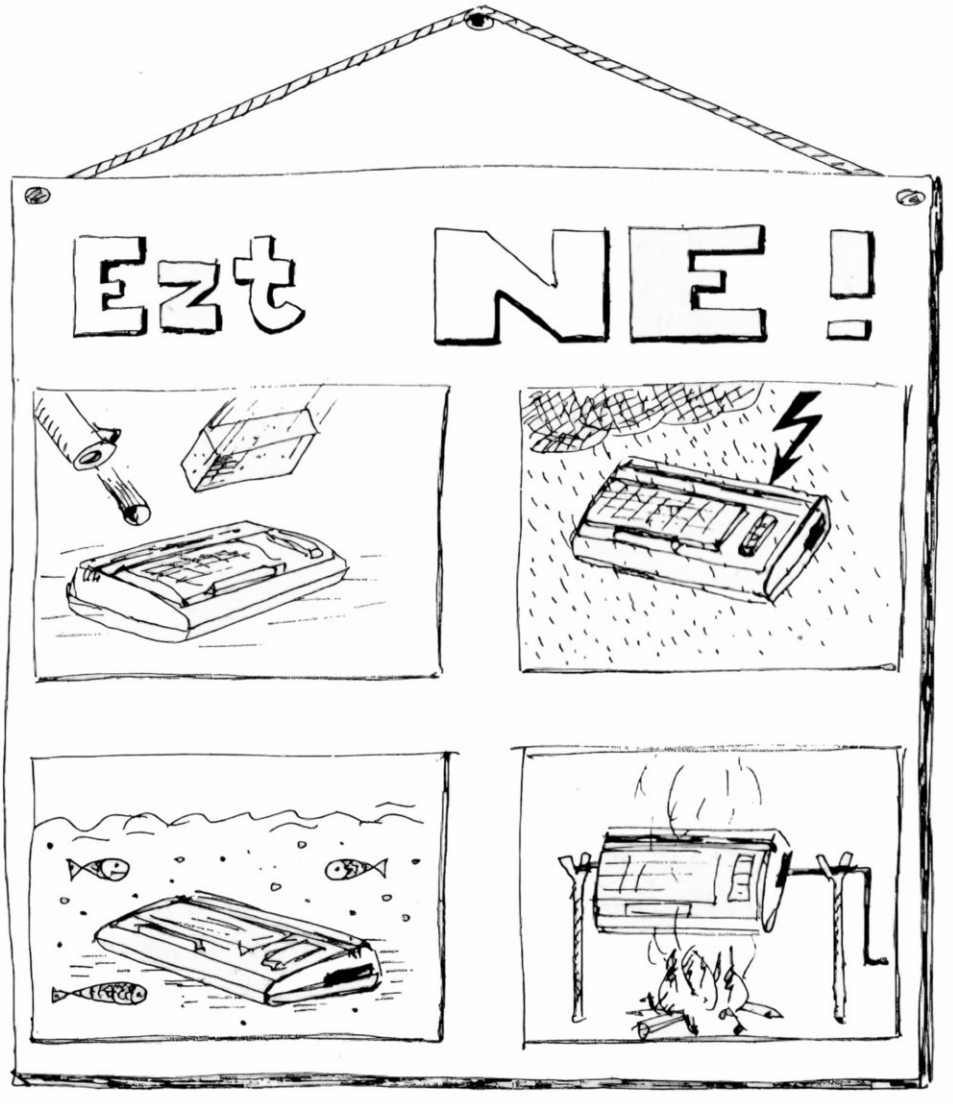
A parancs formája:
kérdőjel PEEK nyitózárójel cím csukózárójel RETURN-gomb

Gépi kódú programot elindítani például a **SYS 50000** paranccsal lehet. Ha ezt begépeljük (a SYS szó után szóközzel vagy anélkül), és lenyomjuk a RETURN-t, elindul az a gépi

kódú program, amely az ötvenezredik memóriarekeszben kezdődik. Ezt most még ne gépeljük be!

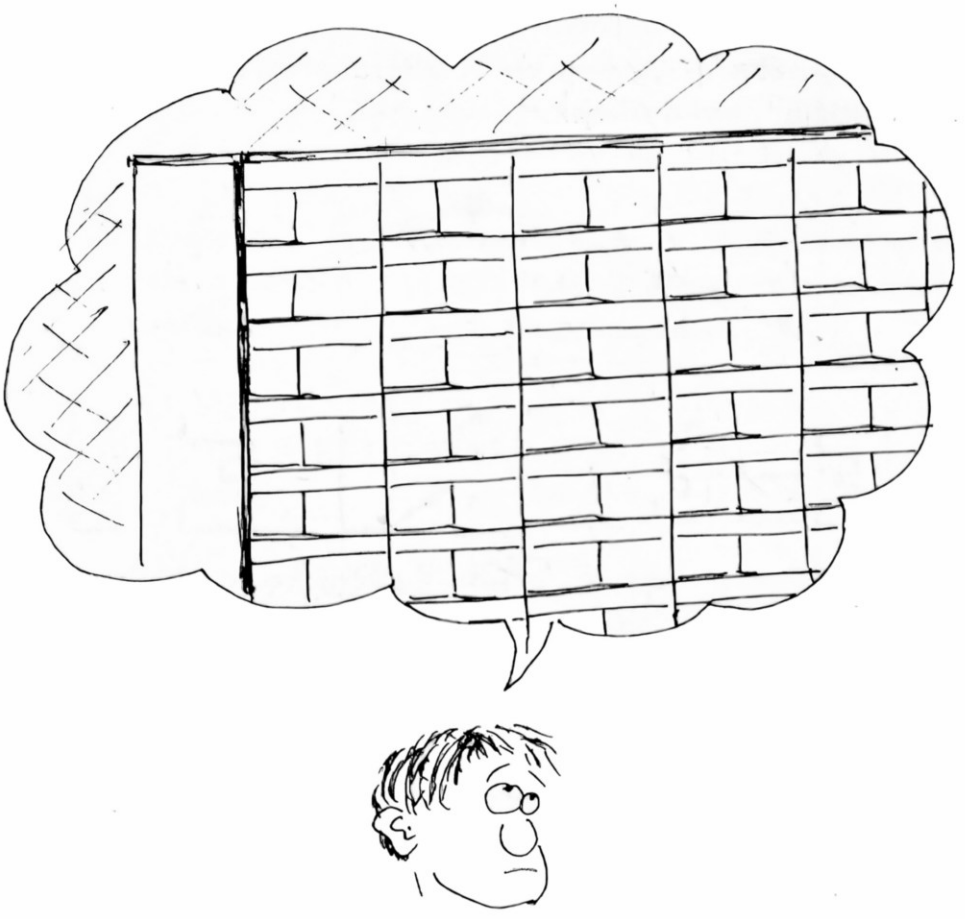
A parancs formája:
SYS cím RETURN-gomb

Hogy ezután mi történik, az attól függ, mi van az adott címmel megjelölt memóriahelyen.
Ha valamit hibásan gépeltünk, és még nem nyomtuk le a RETURN gombot, akkor a jobb felső sarokban levő INST/DEL feliratú gomb lenyomásával a legutóbb begépelt jelet (betűt, számjegyet stb.) törölhetjük. Újabb gombnyomásra eltűnik még egy jel, és így tovább. Ha a hibás részt eltüntettük, begépelhetjük a jót.
Ha bármi furcsaságot észlelünk, ha a gép a gombnyomásokra nem reagál, ha nem jelenik meg a READY, akkor kapcsoljuk ki, és néhány másodperc várakozás után kapcsoljuk vissza!
Végezetül egy bekezdés az „Alapismeretek" c. kötet első fejezetéből (25. oldal): „Egy dolog felől nyugodtak lehetünk: hibás parancsok, hibás programok, téves gombnyomogatás a Commodore gépet nem ronthatja el. Ha fizikailag és elektromosan nem nyúltunk a géphez (nem szedtük szét, nem ejtettük le, nem kötöttük 220 V-nál nagyobb feszültségű hálózatra stb.), és a központi egység ki- és visszakapcsolása után mégsem áll vissza az alapállapot, ez a gép hibája, nem a miénk."



A MEMÓRIÁHOZ NYÚLUNK

A programok működés közben a tárban vannak. A tárat úgy is el lehet képzelni, mint egy hosszú polcot, amin sorban rekeszek vannak. (Természetesen itt is és a továbbiakban is a gép a Commodore–64-et, a program a C–64 gépen írt programot, az utasítás a C–64 gépi utasítását jelenti, nem törődve azzal, hogy a könyvben szereplő fogalmak és állítások közül melyek volnának ennél szélesebb körben is érvényesek.)



A rekeszek száma 65536 (a gép számára ez kerek szám). Ezekben a rekeszekben számok vannak, mégpedig mindig egészek és soha nem negatívak. A legkisebb lehetséges szám értéke 0, a legnagyobbé 255. A rekeszeknek (tárhelyeknek) nincs semmiféle neve, megkülönböztető jele, rovatcíme. Hogyan találhatjuk meg őket? A sorszámuk alapján. A rekeszek sorszámozása a nulladikkal kezdődik, és a 65535-ödikkel fejeződik be.
A ? PEEK(0) parancs hatására pl. – ha a gépet éppen most kapcsoltuk be – **47 fog meg**jelenni, ez a gép nulladik memóriarekeszének tartalma, ez a szám van **a nullás** címen.

Az első néhány cím tartalma bekapcsoláskor: 47 55 0 170 177 145 179 34 34 0.
Nem fordulhat elő, hogy egy címen semmi nincs, legalább nullának mindenképpen lennie kell. (Ha a rekeszekben levő kis golyókat kellene megszámolnunk, s valamelyik rekeszben nincsen golyó, akkor ezt úgy mondhatjuk, hogy a rekeszben nulla darab golyó van.)
Mi történik, ha a POKE paranccsal megváltoztatjuk a tár tartalmát? Vigyázzunk, nagy meglepetések érhetnek bennünket! Azt várnánk, hogy minden egyes PEEK azt találja az adott memóriacímen, amit a megelőző POKE beírt oda, ilyesformán:

```
    **** COMMODORE 64 BASIC V2 ****
64K RAM SYSTEM  38911 BASIC BYTES FREE
READY.
POKE 50000,0
READY.
? PEEK (50000)
 0
READY.
POKE 50000, 123
READY.
? PEEK (50000)
 123
READY.
■
```

Léteznek azonban címek, ahová már a beírás is furcsa mellékhatással jár. Ha az 1200-as címre egyet írunk, akkor a képernyőn megjelenik egy váratlan A betű. Más területeken teljesen hatástalan a POKE parancs.

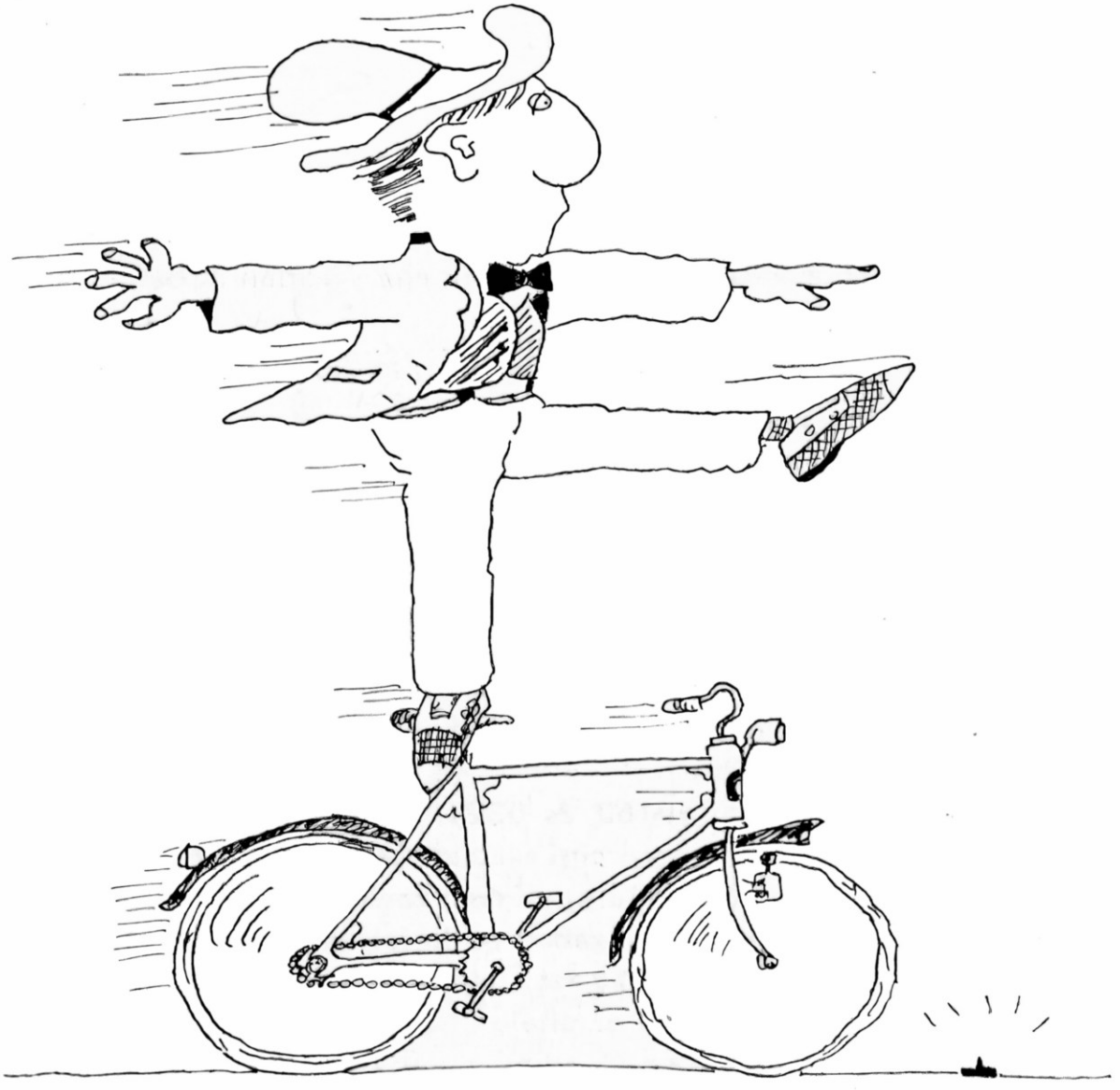
```
POKE 1200,1

READY.
■
                    A
```

```
? PEEK (60000)
 234
READY.
POKE 60000,0
READY.
? PEEK (60000)
 234
READY.
■
```

Vannak veszélyes címek is, a **POKE 1, 0** hatására például még a READY sem jelenik meg, és a gép minden gombnyomásra érzéketlenné válik. Semmi baj, kikapcsoljuk, visszakapcsoljuk, és dolgozhatunk tovább.
Jól jegyezzük meg ezt a két címet: 49152 és 53247
Ez a kettő és a köztük levő valamennyi cím szabad: ezen a 4096 db címen azt csinálhatunk, amit akarunk, ide bármit tölthetünk, ez nem zavarja meg a gép működését, és soha nem okoz váratlan eseményt. (Hasonló szabad területek találhatók másutt is a tárban, de ez a legnagyobb.) A továbbiakban mindig ezt a területet használjuk, ide fogjuk tölteni gépi kódú programjainkat, és itt tároljuk az általuk használt adatokat is. Ne feledjük, hogy bármit töltünk erre a tárterületre, az csak addig marad meg, amíg a gépet ki nem kapcsoljuk. Friss bekapcsolás után teljesen véletlenszerű, kiszámíthatatlan, hogy mit találunk ezeken a címeken. És természetesen: ha egy címre POKE utasítással vagy bárhogy másképp (pl. programmal) új adatot töltünk, a régi elvész, és semmiféle módon nem hozható vissza. Nekünk kell tehát ügyelnünk arra, hogy egy adat helyére csak akkor írjunk másikat (azaz egy adatot csak akkor írjunk felül), ha az eredetire többé már nem lesz szükségünk.

A SZÁMRENDSZEREK

A biciklitúra csodálatos élmény, de ahhoz, hogy részt vehessünk rajta, először meg kell tanulni biciklizni, össze kell csomagolni, térképet kell venni, meg kell javítani a biciklit, és szólni kell a meteorológusoknak, hogy legyen jó idő.



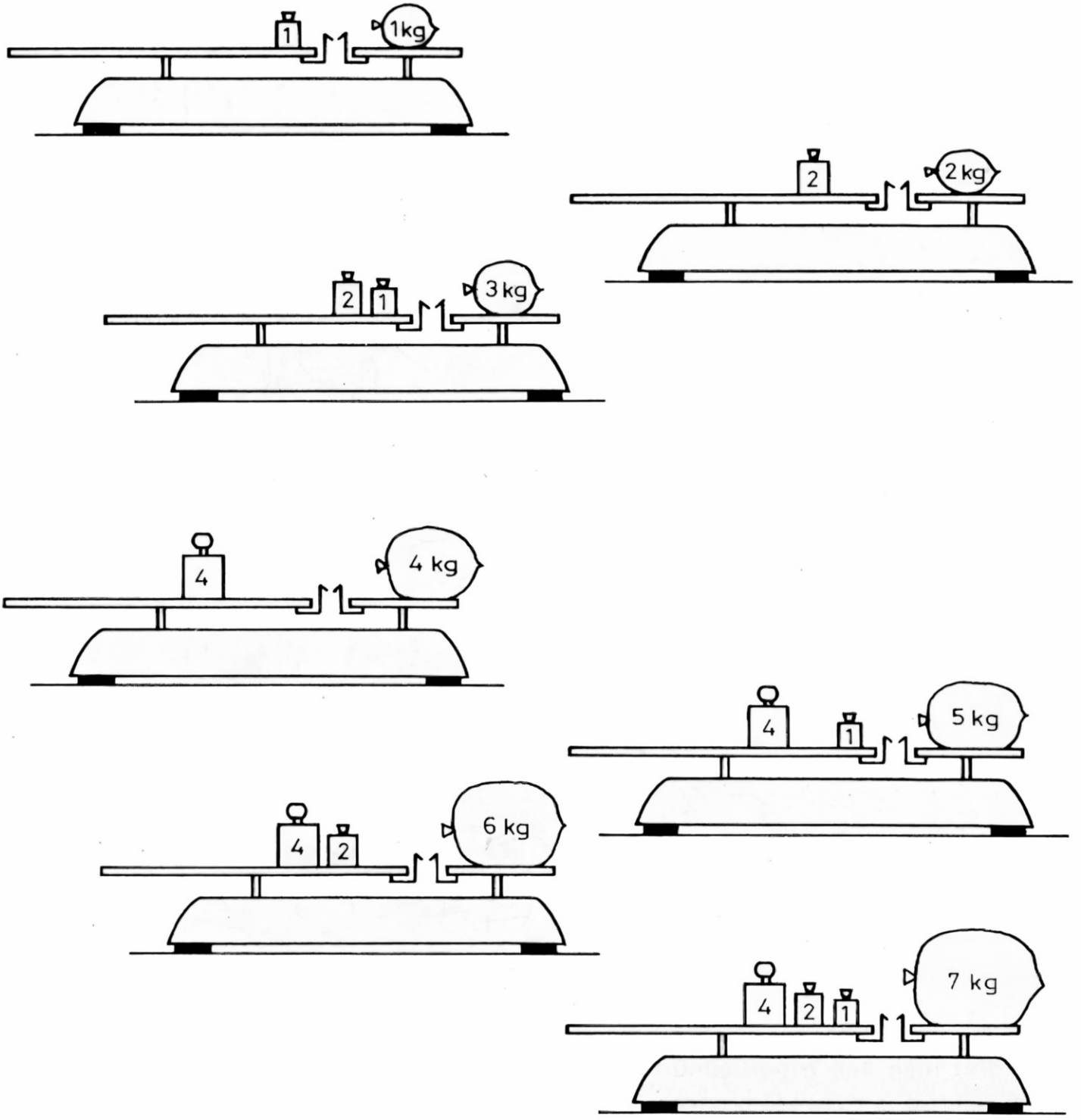
Valljuk be, hogy összecsomagolni, térképet venni, biciklit javítani, jó időt rendelni rengeteg utánjárással jár, és egyáltalán nem olyan csodálatos, a biciklizni tanulás pedig még veszélyes is lehet!

Gépi kódban programozni szintén csodálatos, de ahhoz, hogy gépi kódban programozhassunk, előbb meg kell ismerkednünk a számítógépünk által használt kettes és tizenhatos számrendszerrel (a tízes számrendszert már az általános iskolában megismertük).

A KETTES SZÁMRENDSZER

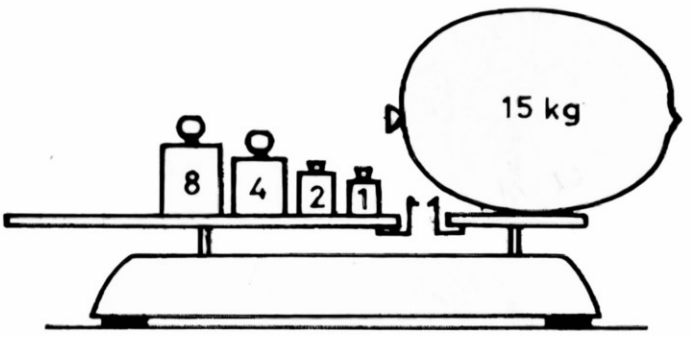
Képzeljük magunkat egy dinnyeárus helyébe! Az egyszerűség kedvéért tegyük fel, hogy országunkban csak egész kilós dinnyék teremnek! Mérésükhöz így nem kellenek egykilósnál könnyebb súlyok. Állítsunk össze egy súlykészletet, mellyel minden lehetséges dinnyét lemérhetünk!

Ha csak egy darab egykilós súlyunk van, akkor ezzel minden egykilós dinnyét megmérhetünk. Ez persze nem elég, vinnünk kell magunkkal még egy kétkilós súlyt is. A két súly segítségével már egészen három kilóig mérhetünk, hiszen az egykilós dinnyékhez elég az egykilós, a kétkilós dinnyékhez a kétkilós súly, a háromkilós dinnyéket pedig a két súly együttes használatával mérhetjük le.

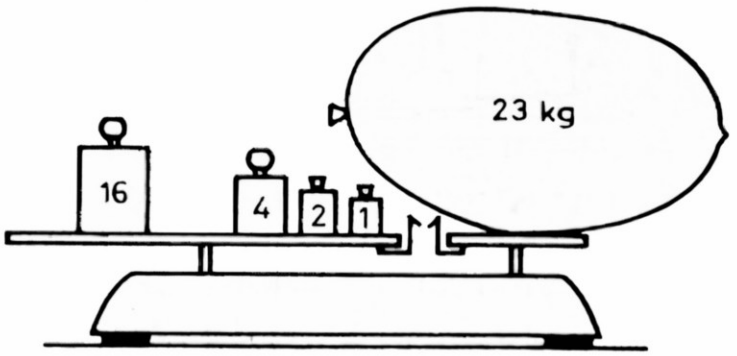


Ahhoz, hogy négykilós dinnyéket is mérhessünk, vinnünk kell magunkkal még egy négykilós súlyt is. A három súllyal már hét kilóig mérhetünk, mégpedig a következőképpen: három kilóig a négykilós súly nélkül is boldogulunk. Négykilós dinnyéhez elég a serpenyőbe tenni a négykilós súlyt. Ötkilós dinnyék méréséhez a négy- és az egykilós súly kerül a serpenyőbe. Hatkilós dinnyéket a négy- és a kétkilós súllyal mérhetünk, hétkilósakat pedig úgy, ha mindhárom súlyt egyszerre a serpenyőbe tesszük. (Figyeljük meg azt is, hogy ezzel a súlykészlettel minden kívánt súly csak egyféleképpen mérhető!)

A negyedik súly, amit magunkkal kell vinnünk, a nyolckilós lesz. Ezzel a négytagú súlykészletünkkel 15 kilóig minden dinnyét le tudunk mérni, a tizenöt kilóst pl. úgy, hogy mind a négy súlyt a serpenyőbe tesszük.

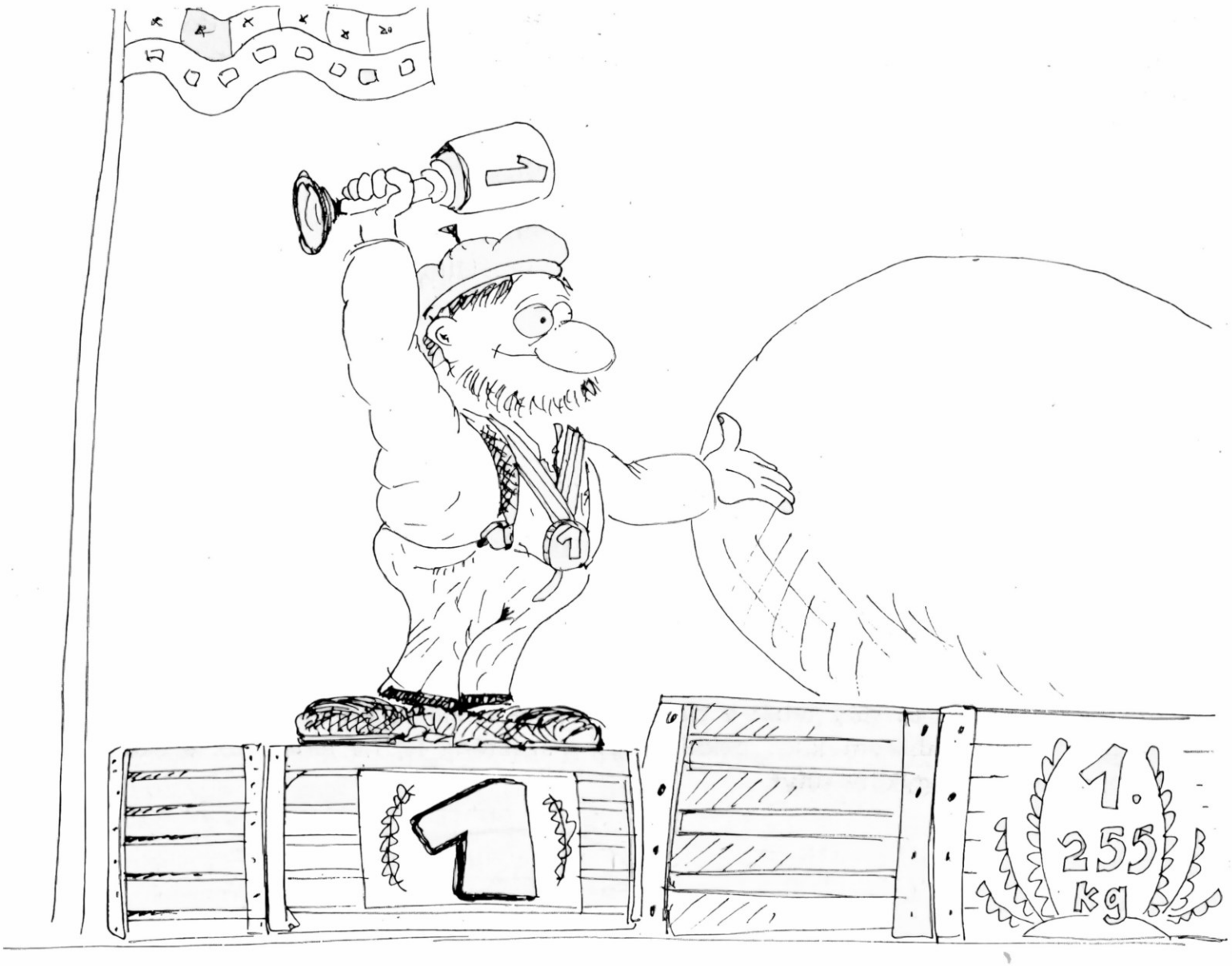


A következő szükséges súly tehát a tizenhatkilós lesz, amelynek segítségével 31 kilóig lehet mérni. Huszonhárom kilót például úgy mérhetünk le, ha feltesszük a tizenhat-, a négy-, a két- és az egykilós súlyt.



Öttagú súlykészletünkkel 31 kilóig mérhetünk. Ha hozzáveszünk egy hatodik, harminckétkilós súlyt is, akkor 63 kilóig, és ha ezek után még egy hetedik, hatvannégykilós súlyt is, akkor egészen 127 kilóig tudunk mérni. Ez már tekintélyes súly ugyan, de adódhat olyan eset, amikor még ennyi sem elég. Ahhoz tehát, hogy lemérhessük a világbajnok dinnyét is, kell még egy nyolcadik, 128 kilós súly is. Nyolctagú súlykészletünkkel lemérhetjük a 255 kilós világbajnok dinnyét, és mellesleg mi is bekerülhetünk a GUINNES rekordok könyvébe, mint a dinnyemérés világbajnokai.

Természetesen pedáns ember nem mér dinnyéket csak úgy, mindenféle adminisztráció nélkül. Aki ad valamit a tisztességes üzletmenetre, pontosan fel is jegyzi, hogy mikor, kinek, milyen dinnyét adott el. És persze nemcsak a dinnye súlyát írja fel, hanem azt is, hogy mely súlyok kerültek a serpenyőbe. Így aztán akár évek múltán is pontosan visszaidézhető minden. Ehhez azonban dinnyeeladási adatlapokra van szükség. Egy 133-kilós dinnye eladását regisztráló dinnyeeladási adatlap valahogyan így fest:



Dinnyeeladási adatlap

Dátum: . . . . 1987. augusztus 20. . . . . .
Vásárló neve: . . . Túró Gombóc . . . . . .
Személyi száma: . 166 01 01 2323 . . . . . .
Lakhelye: .Bp. Liliom köz 893. . . . . .

A mérlegen lévő súlyok:

128	64	32	16	8	4	2	1
igen	nem	nem	nem	nem	igen	nem	igen

A dinnye súlya: . . . 133 kp. . . . . .

Ilyen adatlapokat kitölteni nem túl kellemes, de még lassú is, az eladó tehát leegyszerűsíti a dolgokat, és egy idő után az *igen* meg a *nem* helyett egyeseket és nullákat ír a lapra, attól függően, hogy az adott súly rajta volt-e a mérlegen, vagy nem. Az adatlap tehát így módosul:

A mérlegen lévő súlyok:

128	64	32	16	8	4	2	1
1	Ø	Ø	Ø	Ø	1	Ø	1

A dinnye súlya: 133 kp.

Nézzük egy 89 kilós dinnye adatlapját:

A mérlegen lévő súlyok:

128	64	32	16	8	4	2	1
Ø	1	Ø	1	1	Ø	Ø	1

A dinnye súlya: 89 kp.

Észrevehető, hogy egy adatlapot csak egyféleképpen lehet kitölteni, egy bizonyos súlyt az adott súlykészlettel csak egyféleképpen lehet kirakni. Nullától 255-ig minden súlyhoz tartozik egy és csak egy lehetőség, ahogyan az adott súlyérték kirakható.
És most felejtsük el a dinnyéket, és koncentráljunk csak a számokra! A 133 helyett azt írtuk, hogy: 10000101. A 89 helyett azt, hogy: 01011001. Ezzel el is jutottunk a kettes számrendszerhez, amelynek az a lényege, hogy minden számot egyesek és nullák sorozatával írunk le. Előnye, hogy csak kétféle számjegy van (a 0 és az 1), hátránya viszont, hogy a számok hosszabbá válnak. A tízes számrendszerben háromjegyű 255 például kettes számrendszerben nyolcjegyű lesz: **11111111**.
Kettes számrendszerben minden számjegy kétszer annyit ér, mint a tőle jobbra levő. A kettes számrendszerbeli szám értéke a következő módon számolható ki: Elindulunk jobbról balra. Minden számjegy a helye szerint kettőször annyit ér, mint az előző. Össze kell adnunk azokat az értékeket, ahol a számjegy egyes, és megkapjuk a végeredményt. A 20. oldalon két példát is mutatunk erre. Számoljunk együtt!

Nemcsak kettes számrendszerből lehet átszámolni tízesbe, hanem fordítva, tízes számrendszerből is lehet kettesbe. Ez persze már nem ilyen egyszerű, mert meg kell találni az adott számhoz tartozó változatot. (Ha már megvan, könnyen ellenőrizhetjük, hogy valóban jó-e.) A megfelelő variáció megtalálására is van egy módszer: meg kell nézni, kettőnek melyik az a legnagyobb hatványa, amely még levonható a számból. Nézzük például a

b 1 1 Ø 1 Ø Ø 1 1

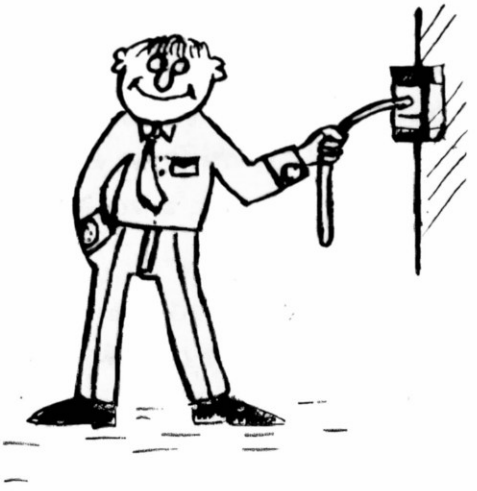
1 x 1 = 1
1 x 2 = 2
Ø x 4 = Ø
Ø x 8 = Ø
1 x 16 = 16
Ø x 32 = Ø
1 x 64 = 64
+ 1 x 128 = 128
211

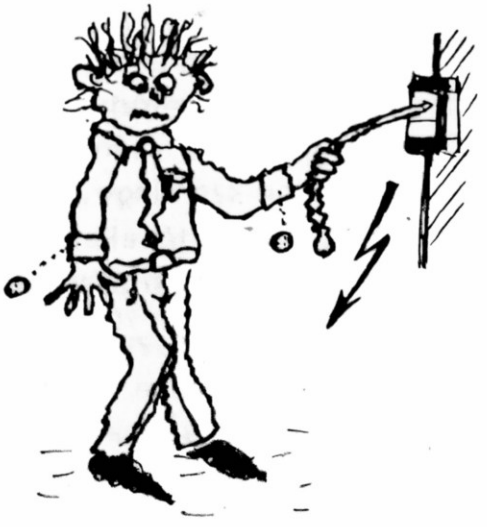
b 1 1 1 1 Ø Ø Ø Ø

Ø x 1 = Ø
Ø x 2 = Ø
Ø x 4 = Ø
Ø x 8 = Ø
1 x 16 = 16
1 x 32 = 32
1 x 64 = 64
+ 1 x 128 = 128
240

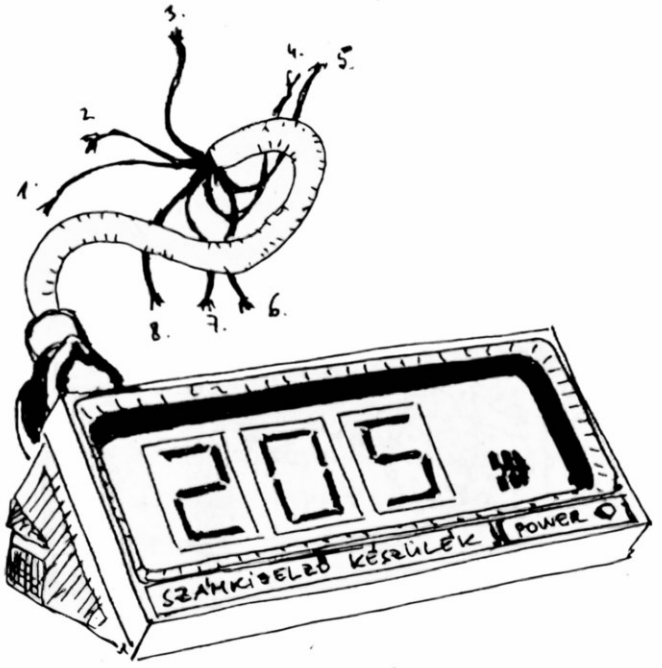
35-öt! 35-ből nem vonhatunk ki 64-et, de 32-t igen: 35—32=3. A 3-ból kivonhatunk 2-t: 3—2=1, s végül marad ez az egy. Volt tehát 32, 2 és 1. Ezt az adatlap mintájára így írhatjuk: 00100011. (Az ellenőrzést gyakorlásképpen bárki elvégezheti.)

Mire jó ez az egész? A számítógép elektronikus szerkezet. Az elektronikus szerkezetekben vezetékek vannak. Egy vezetékben a földhöz képest pedig vagy kicsi a feszültség, ez a nulla, vagy nagy a feszültség, ez az egy.



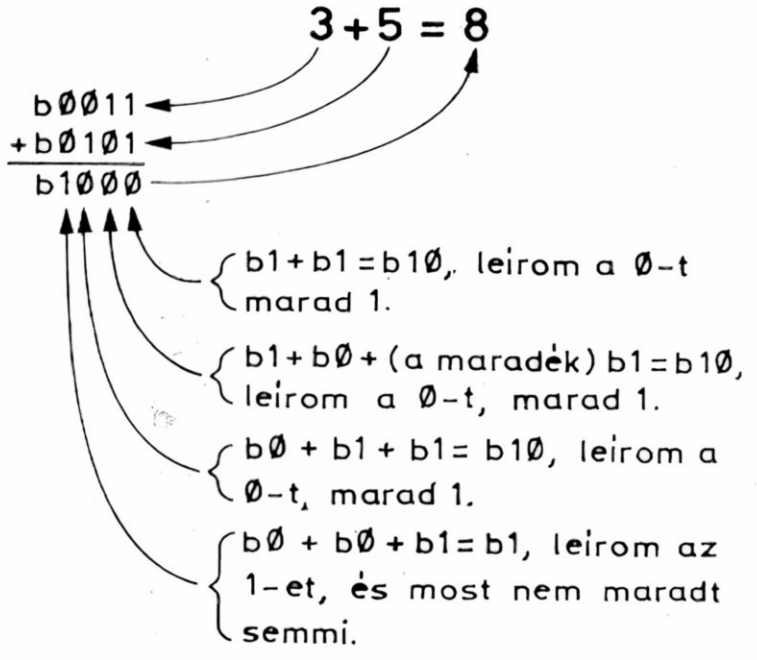


vagyis a számítógépben az jelenti az 1-et, ha valamelyik vezetéken nagy a feszültség, és a nullát az, ha kicsi. Ha sok vezeték van egymás mellett (mondjuk éppen nyolc, mint a Commodore 64-ben és sok más hasonló kategóriájú mikroszámítógépben), akkor attól függően, hogy mely vezetékeken mekkora feszültségek vannak, el tudunk számolni nullától egészen 255-ig. A számítógép tehát kettes számrendszerben számol, mégpedig úgy, hogy nyolc vezetéket együtt tekint egy számnak.



Természetesen bárkinek joga van ahhoz, hogy otthon a sufniban olyan számítóépet építsen, amely mondjuk akár 60-as számrendszerben számol, mint a régi babiloniak.

A különböző matematikai műveleteket nyilvánvalóan bármelyik számrendszerben elvégezhetjük, és az eredmények egyformán helyesek lesznek. Egy összeadás vagy szorzás, de bármi más művelet eredménye is független attól, hogy milyen számrendszerben számolunk. Az érdekesség kedvéért nézzük meg, hogyan összegezhető kettes számrendszerben két szám!



A művelet nagyon hasonlít a tízes számrendszerbeli összeadáshoz. Hasonlóan a kivonás is elvégezhető, továbbá a szorzás és az osztás is.

Ha kettes számrendszerbeli számról beszélünk, akkor megkülönböztetésül mindig írjunk elé egy „b" betűt! Azért „b" betűt, mert a kettes számrendszerbeli számokat idegen szóval bináris számoknak nevezik, és a bináris „b" betűvel kezdődik. Nem árt azonban, ha tudjuk, hogy vannak, akik a kettes számrendszerbeli számok elé egy % jelet írnak; ne lepődjünk meg, ha néhány szakkönyvben ilyen jelölést találunk, és azon se, ha a „b" betűt (vagy a „%" jelet) a szám után írják!

A b 1000 egy bináris szám (az értéke = 8) és benne az **1** egy bináris számjegy. A kettes számrendszerbeli vagy bináris számjegyet angolul binary digitnek nevezik. Ha ezt lerövidítjük, akkor azt kapjuk, hogy bit. A kettes számrendszerbeli számoknál egy számjegy egy bit. Külön jelentőségük van azoknak a kettes számrendszerbeli számoknak, amelyekben nyolc számjegy, azaz nyolc bit van. Ezeket a számokat úgy hívjuk, hogy byte (bájt). Ma a magyar nyelvű szakirodalomban már egyre inkább a „bájt" írásmódot használjuk, mint ahogyan a mikront sem úgy írjuk, hogy „micron".

A b 10100011 nyolcbites szám, vagyis bájt, mert nyolc számjegyből áll (értéke 163), a b 1111001001001 (bár ez is kettes számrendszerbeli szám) már nem nyolcbites, tehát nem bájt (értéke 15433).

Ha egy szám nyolcnál kevesebb bitből áll, attól még le lehet írni úgy, mintha nyolcbites szám lenne, csak az első valahány jegye értéktelen nulla lesz. Például: b 1101-et leírhatjuk úgy is, hogy b 00001101, és így már olyan, mintha nyolcbites szám lenne, sőt az is.
Ha viszont egy szám hosszabb, mint nyolc bit, akkor legfeljebb annyit tehetünk, hogy felosztjuk nyolcbites részekre. Például: b 11001010011 = b 110 01010011 = b 00000110 01010011
Azokat a számokat, amelyeket 16 biten le lehet írni, tizenhatbites számoknak is szokták hívni, de elterjedtebb a „kétbájtos" megnevezés.

Számítógépünk tehát nyolcbites, kettes számrendszerbeli számokkal számol, ezért nekünk is illik egy keveset tudnunk erről. Sajnos a kettes számrendszernek nemcsak előnyei vannak, hanem hátrányai is. Ezek közül már ismerjük a legnagyobbat: a kettes számrendszerbeli számok nagyon hosszúak. Például az 50000 kettes számrendszerben b 1100001101010000. Az a szám, amely tízes számrendszerben ötjegyű, kettes számrendszerben könnyen lehet akár tizenhat bites (azaz 16-jegyű) is. De még az ennél kisebb számoknál is probléma van. Írjunk le egy nyolcbites számot, mondjuk a százharmincat: 130 = b 10000010. Leírva is hosszú, hát még kimondani! Vajon kinek van kedve ennyit beszélni, ha azt akarja mondani: 130? És főleg akkor, ha aznap még tizenöt másik számot is fel kell sorolnia!! Már a hőskorban is — amikor még lifttel kellett közlekedni a számítógépeken — feltűnt a programozóknak, hogy ha túl sokáig tart a számokat leírni, akkor nem jut idő arra, hogy számoljanak is velük.

A tízes számrendszert használni nem tűnt célszerűnek, mert a tízes és a kettes számrendszerek közötti átváltás nem túl egyszerű. Olyan számrendszert kellett tehát választani, amelyikbe a bináris számok könnyen átválthatók, és a visszaváltás is egyszerű; nincs túl sokféle számjegy, de azért a számok nem is túl hosszúak. Hamar elterjedt a tizenhatos számrendszer használata, amelyben egy binárisan nyolcbites szám mindig leírható két jeggyel (legfeljebb az első jegy nulla), és amelyikbe nagyon könnyen átszámolhatók a kettes számrendszerbeli számok.

A TIZENHATOS SZÁMRENDSZER

A kettes számrendszerbeli számok feloszthatók jobbról kezdve négybites csoportokra. Például a b 1100101101011 így: 1 1001 0110 1011
Most minden csoport helyére írjunk egy jelet!

0000 = 0
0001 = 1
0010 = 2
0011 = 3
0100 = 4
0101 = 5
0110 = 6
0111 = 7
1000 = 8
1001 = 9
1010 = A
1011 = B
1100 = C
1101 = D
1110 = E
1111 = F

Azt tehát, hogy 1 1001 0110 1011,
ugyanaz, mint 0001 1001 0110 1011,
ami egyenlő lesz 1 9 6 B-vel.
Hát nem rövidebb? És milyen egyszerű! Csak arra kell vigyázni, hogy tizenhatos számrendszerben az 10 nem tíz, hanem 16 (=b10000), az 100 pedig nem száz, hanem 256 (=b100000000).
Nézzünk még néhány példát! Vegyünk egy tizenhatbites számot: b 1101000110011111
Osszuk fel jobbról négybites csoportokra: b 1101 0001 1001 1111, és végezzük el a helyettesítést: D19F! Az 53663 számot tehát tizenhatos számrendszerben úgy írjuk, hogy D19F.

A tizenhatos számrendszer lényege az, hogy negyedannyi számjeggyel írhatjuk le a számokat, mint kettes számrendszerben, továbbá az, hogy rendkívül könnyen tudunk közte és a kettes számrendszer között váltani. Hátránya viszont, hogy tizenhat féle „számjegye"

van, amelyek közül hat nem is szám, hanem betű. Idővel persze ezt is meg lehet szokni, csak kezdetben tűnik nagyon idegennek. Nézzünk még két példát az átváltásra (zárójelben a tízes számrendszerbeli megfelelőjükkel):

```
b 11100010110 =          b 1000110011 0110 =
b 111 0001 0110 =      b 1000 1100 1011 0110 =
b 0111 0001 0110 =           8    C    B    6
   7    1    6  (=1814)                    (=36022)
```

Természetesen a tizenhatosból binárisba sem nehéz átszámolni, csupán az egyes számjegyeket kell a nekik megfelelő csoportokkal helyettesíteni. Például:

```
FD = b 1111 1101        462C = b 100 0110 0010 1100 =
                             = 1000110001011 00
```

Hogy egy tizenhatos számrendszerbeli számot még véletlenül se nézhessünk tízes számrendszerbelinek (lehet, hogy éppen nincs benne betű), már most határozzuk el, hogy tizenhatos számrendszerben a szám elé a dollár jelét ($) írjuk, például $4FC2 (20268). A tizenhatos számrendszerbeli számot hexadecimálisnak is szokás nevezni, s mivel ennek kezdőbetűje H, ezért a H-betűs jelölés is előfordulhat a leírásokban és a programokban.
A továbbiakban minden alkalommal vagy nyolcbites, vagy tizenhatbites számokról fogunk beszélni.
Nyolcbites szám például: b 10101111 = $AF = 175
Tizenhatbites szám pedig: b 1100011010000111 = $C687 = 50823
A nyolcbites számok tizenhatos számrendszerben mindig leírhatók két számjeggyel, a tizenhatbites számok pedig négy számjeggyel. Számoljuk ki egy tizenhatbites szám értékét!

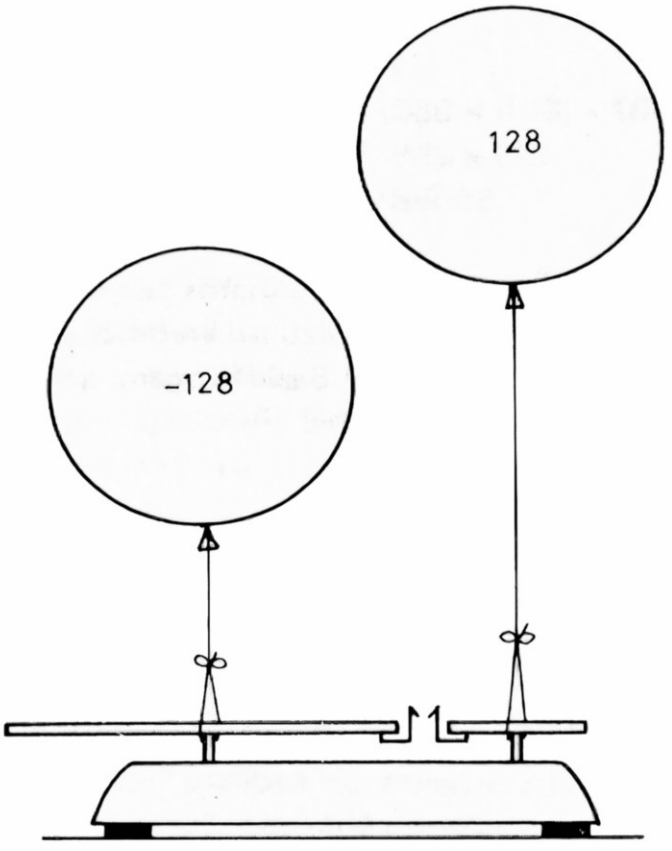
```
$C687 = ($C6 * 256) + $87 =
      198 * 256  + 135 =
         50688  + 135 = 50823
```

Aki még nem elég ismerős a kettes vagy a tizenhatos számrendszerben, gondoljon és számoljon utána! Ha helyesen számolt, ugyanezt az eredményt fogja kapni. A kényelemszeretők részére a függelékben található egy Basic progam, amely az említett három számrendszer bármelyikéből átvált a másik kettőbe.

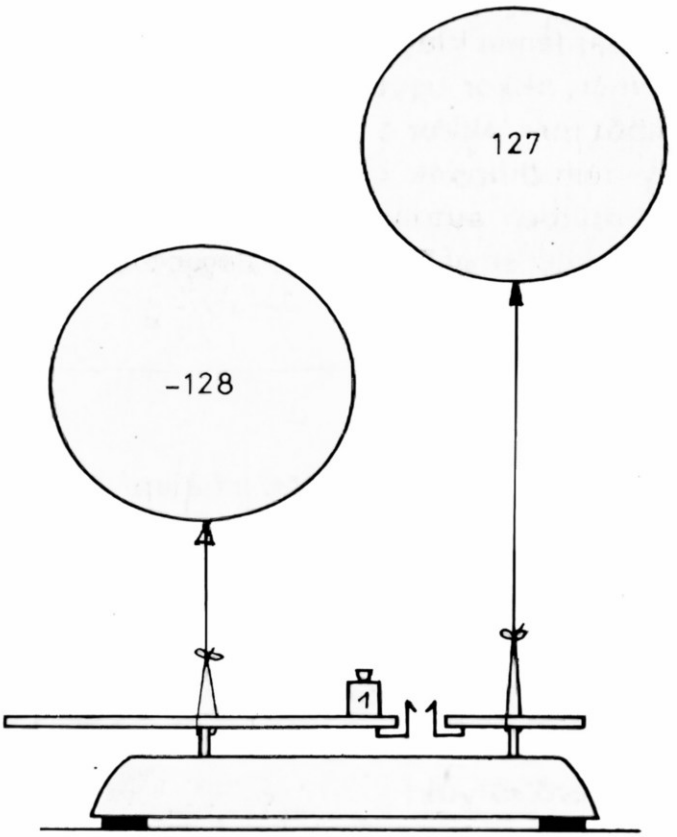
A NEGATÍV SZÁMOK

Rohanó korunkban egy üzletember csak akkor lehet sikeres, ha folyamatosan fejleszti szolgáltatásainak körét, és ha új, ötletes megoldásokat alkalmaz. Nem is csoda, hogy dinnyeárusunk időközben új szolgáltatást vezetett be: léggömbök emelőerejének mérését. Bármilyen, jelzés nélküli léggömbről szerény összegért megállapítja, hogy mekkora súlyt képes felemelni. Persze ahhoz, hogy ezt megtehesse, technikai újítást kellett eszközölnie. Le kellett cserélnie súlykészletének 128 kilós súlyát egy 128 kiló súly felemelésére képes léggömbre. Nézzük meg, ezzel az egyszerű változtatással hogyan is tudta kibővíteni mérlege képességeit!

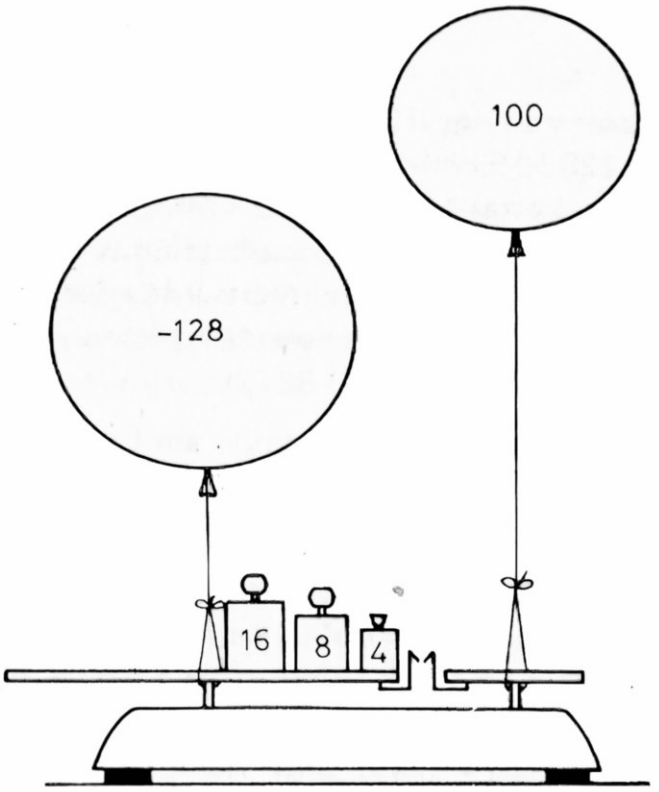
Dinnyék mérését továbbra is vállalhatja, hiszen megvan még súlykészletének hét darabja, így 127 kilóig minden súlyt le tud mérni. Valljuk be, hogy egy hétköznapi dinnyeárusnak ennél nagyobb dinnyét nem is igen kell mérnie, hiszen ilyen dinnyék is ritkán fordulnak elő. Annál gyakrabban kell viszont léggömböket mérni, de már ennek sincs akadálya. Ha például le kell mérnie egy 128 kiló emelőerejű léggömböt, akkor azt könnyedén megteheti a saját 128 kilós léggömbjével.



127 kilós léggömböt úgy mérhet, ha a serpenyőre köti a 128 kilós léggömböt, és ráteszi az egykilós súlyt. Ebben az esetben, ha a másik serpenyőre egy 127 kilós léggömb van kötve, akkor a mérleg éppen egyensúlyban van.



Így tehát minden 128 kilós vagy annál kisebb emelőerejű léggömböt lemérhet. Egyszerűen felköti a 128 kilós léggömbjét, és rátesz még annyi súlyt a mérlegre, amennyi az egyensúlyhoz kell. Ekkor a 128 kilóból levonja a serpenyőben levő súlyokat, és máris adódik a mérendő léggömb emelőereje. Nézzünk egy konkrét példát, ahol egy 100 kilós emelőerejű léggömböt mér!



Ahhoz, hogy az emelőerő 100 kiló legyen, a 128 kilós léggömb mellé fel kell helyeznie 28 kilót, amit úgy tud megtenni, ha felteszi a 16, a 8 és a 4 kilós súlyokat is. A mérleg így egyensúlyba kerül, és árusunk végre kitöltheti az adatlapot, mert újítás ide, technikai fejlődés oda, az adminisztráció elengedhetetlen. A raktáron több ezer régi dinnyeeladási adatlapja van, ezeket pazarlás lenne kidobni... Furfangos dinnyeárusunk hamar rájön a megoldásra: Ha dinnyét mér, akkor ugyanúgy tölti ki az adatlapot, mint eddig, változás nincs. Ha viszont léggömböt mér, akkor a dinnye súlya rovatba negatív számot ír, és ebből később tudni fogja, hogy nem dinnyét, hanem léggömböt mért (mivelhogy negatív súlyú dinnye nincs). Ebben az esetben automatikusan azt is tudja, hogy a 128 kp-os rubrika nem súlyt, hanem egy 128 kiló emelőerejű mérőléggömböt jelent. A 100 kilós léggömbről készült adatlap:

Dinnyeeladási adatlap

Dátum: . . 1987. október 5. . . . . .
Vásárló neve: . . Túró Gombóc . . . . . .
Személyi száma: . . 166 01 01 2323 . . . . . .
Lakhelye: . . Bp. Liliom köz 893 . . . . . .

A mérlegen lévő súlyok:

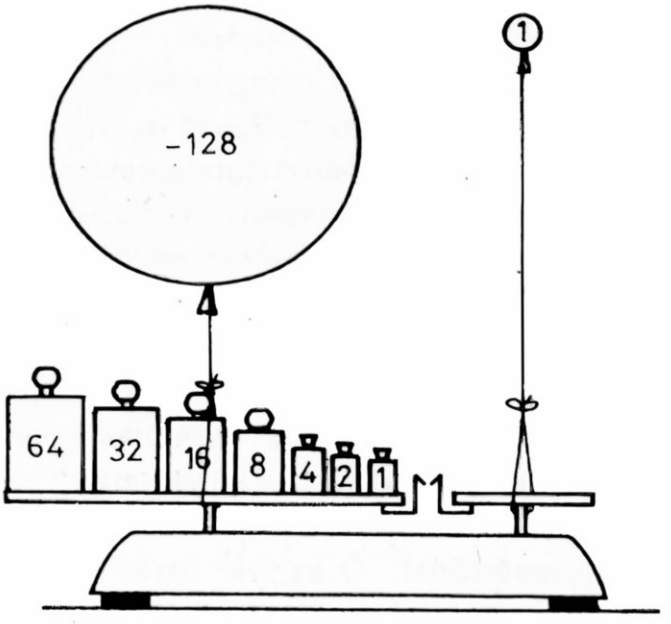
128	64	32	16	8	4	2	1
1	Ø	Ø	1	1	1	Ø	Ø

A dinnye súlya: −100 kp . . . . . .

Később könnyű az ellenőrzés: volt egy 128 kiló emelőerejű léggömb, egy 16, egy 8 és egy 4-kilós súly, ami összesen −128+16+8+4=−100 kiló.
Ezzel emberünk megalkotta a kettes komplemens kódot, amit azonban ő még nem nevezett így (az elnevezést már a számítástechnikusok találták ki). Újításával még a súlyok szállítását is megoldotta: ha mind a hét súlyt rákötí a léggömbre, az még így is a levegőben marad. A hét súly összesen még mindig csak 127 kiló, ami kevesebb, mint a léggömb 128 kilós emelőreje.

Ez persze azt is jelenti, hogy a legnehezebb dinnye, amit meg tud mérni, 127 kiló, a legnagyobb emelőerejű léggömb pedig −128 kiló. Amikor pl. egykilós léggömböt mér, akkor a léggömb mellett mind a hét súly rajta van a mérlegen, így lesz a végleges emelőerő 1 kiló.

Visszatérve a számítástechnikához, sokáig idegesítette a programozókat, hogy a számítógépek csak pozitív számokkal tudnak számolni. Kerestek olyan trükköket, amivel negatív számok látszatát tudták kelteni. Az egyik ilyen trükk a kettes komplemens kód, amelyet a C–64 is használ. Itt a programozók megegyeznek egymás között, hogy a nyolcbites számok legfelső (legmagasabb helyiértékű) bitje nem 128-at, hanem −128-at ér. Ebben



az esetben a nyolc biten nem nullától 255-ig tudnak számolni, hanem −128-tól +127-ig. Ez persze nem mindig elég. Éppen ezért a kettes komplemens kódot kétbájtos számokra is alkalmazzák. Két bájton, azaz 16 biten eddig nullától 65535-ig tudtak számolni (1+2+ +4+8+16+32+64+128+256+512+1024+2048+4096+8192+16384+32768=65535). Ha azonban a 32768 kilós súlyt kicseréljük egy ugyanekkora emelőerejű léggömbbel, akkor 32768 kilós léggömböktől kezdve egészen 32767 kilós dinnyeszállítmányokig mindent mérhetünk. Vagyis, ha megállapodunk abban, hogy a tizenhatbites számok legfelső bitje ne 32768-at, hanem mínusz 32768-at érjen, akkor tizenhat biten −32768-tól 32767-ig tudunk számolni.

AZ ELSŐ PROGRAM

Eljutottunk addig, hogy:
- tudunk a gép memóriájába számokat beírni;
- tudjuk, hol vannak olyan szabad memóriaterületek, ahol ezt váratlan következmények nélkül megtehetjük;
- tudjuk, hogy egy-egy memóriarekeszbe miért csak 0 és 255 közötti szám kerülhet;
- ezeket a számokat vissza is tudjuk olvasni;
- és a memóriában megtalálható gépi kódú programot el is tudjuk indítani.

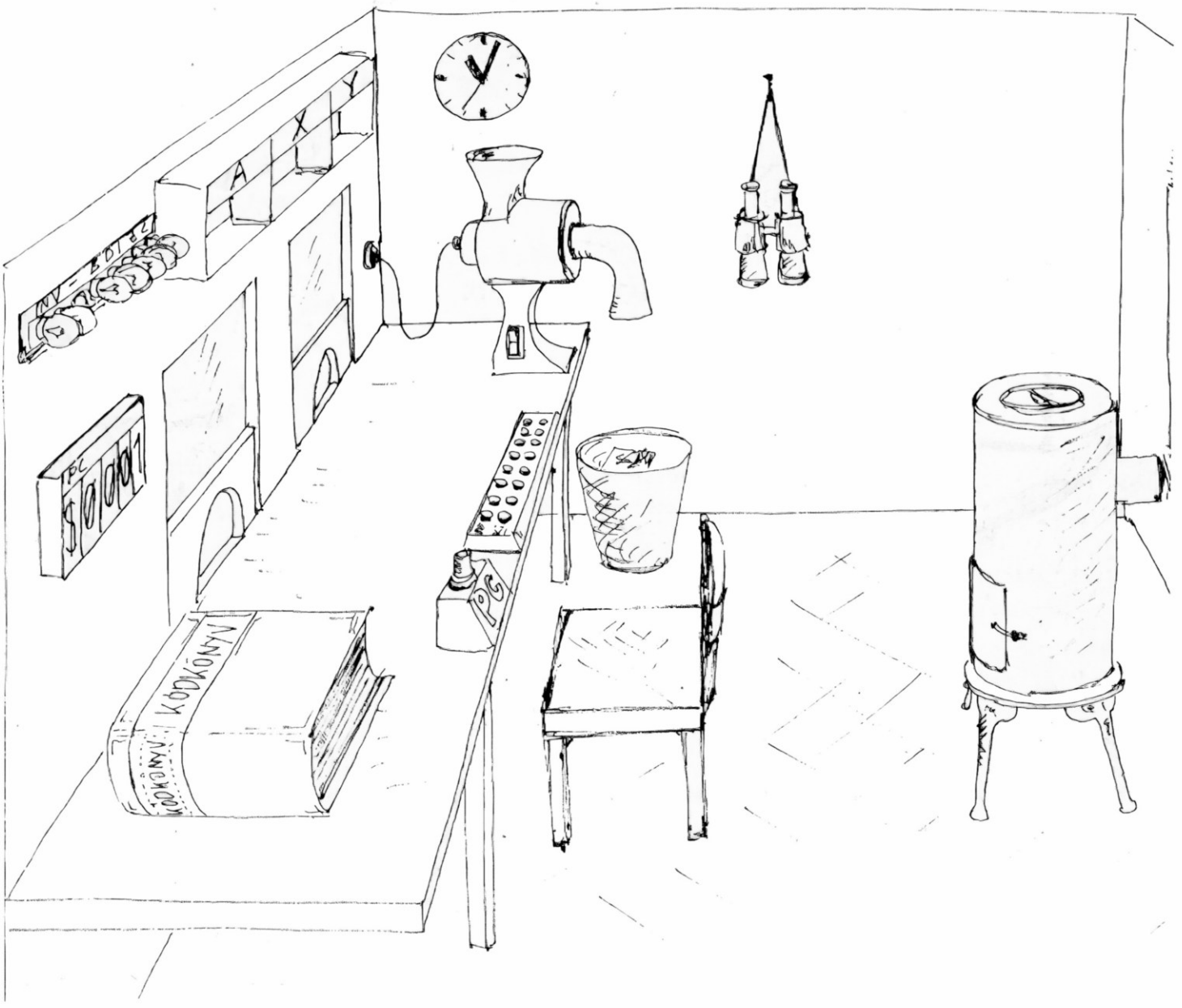
Itt az ideje, hogy magunk is írjunk egy gépi kódú programot. Cseréljük ki két memóriarekesz tartalmát! A feladat egyszerűnek tűnik, bár még a magas szintű Basic nyelvben is csak több utasítással oldhatjuk meg, először le kell bontani egyszerű műveletek egymásutánjára. Gépi kódban való programozás esetén ez még inkább így van. Az összetett feladatokat az emberek is rendszerint munkamegosztással oldják meg. A gép is ezt teszi: mintha alkatrészei egy hatalmas hivatal egyszerű, szorgos, precíz ügyintézői lennének. Ismerkedjünk meg ezzel a hivatallal!

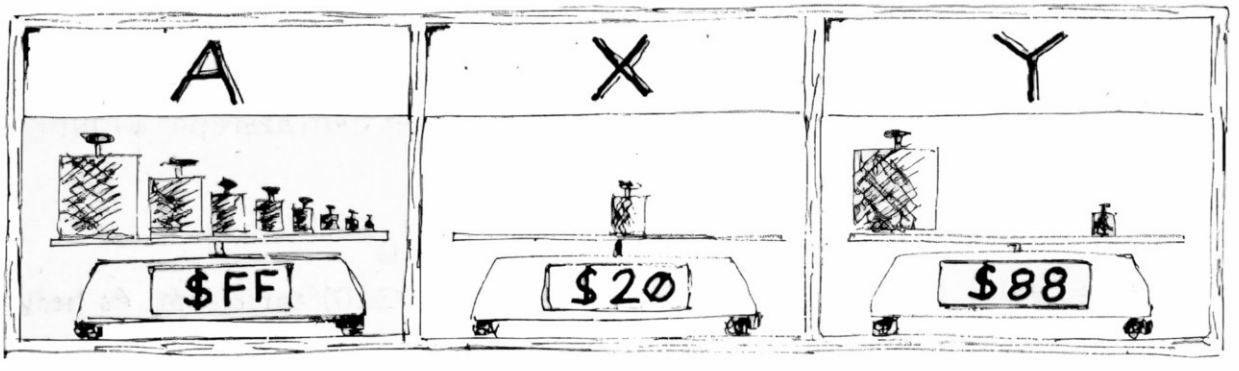
Minden hivatalban az egyik legfontosabb személy a főnök, akit nevezzünk mi egyszerűen csak vezérlőnek. Ő az, aki összehangolja és irányítja a beosztottak tevékenységét, és ő az, aki nélkül tulajdonképpen semmi érdemleges nem történhet. Hivatali szobája telve van csupa fontos holmival, és természetesen a rendetlenség legapróbb nyomát sem lehet felfedezni benne.

Logikus, hogy a vezérlőnek meglegyenek a maga speciális memóriarekeszei, ahol munka közbeni részeredményeket és más adatokat tárolhat. A C—64-es vezérlőjének három ilyen rekesze (szakmai nyelven regisztere) van. Mindegyik regiszter egy-egy nyolcbites számot tud tárolni ugyanúgy, mint az egyes memóriarekeszek. Súlypéldánkkal élve: mindegyikben van egy nyolcdarabos súlykészlet és egy mérleg, amely (hexadecimálisan) jelzi a súlyt. Így tehát vagy nincsen semmi a mérlegen, és akkor $00-t jelez, vagy attól függően, hogy mennyi a rajta levő súly, valamilyen számot mutat. Ez a szám persze maximálisan $FF lehet, hiszen ez tízes számrendszerben pontosan 255, és nyolcdarabos súlykészletünkkel ennél nagyobb súlyt nem is tudunk produkálni.

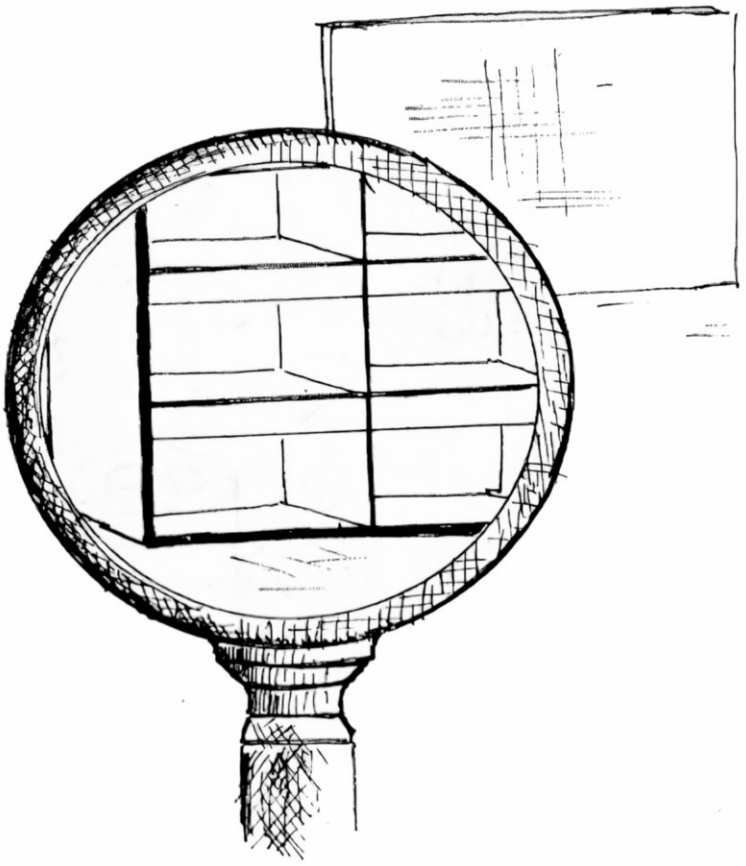
A vezérlő így a regisztereiben igen körülményes módon ugyan, de számokat tud tárolni, mégpedig 0 és 255 közötti számokat. A regisztereknek külön jelzésük is van: A, X, Y.

Ismerkedjünk meg a vezérlő legfontosabb beosztottjával, a memóriakezelővel és magával a memóriával! A memóriát úgy képzeljük el, mint egy nagy szobát, amelynek falát olyan rekeszek alkotják, mint amilyenek a vezérlő íróasztalában is vannak. Minden rekeszben van egy-egy mérleg, a hozzá tartozó nyolcdarabos súlykészlettel. Korábbról tudjuk, hogy





a C–64 memóriarekeszeinek száma 65536; képzeljük el úgy, mintha a falon 256 oszlopban, oszloponként 256 rekesz lenne. A memória a nulladik rekesszel kezdődik, és a 65535. ($FFFF) rekesszel ér véget. Ha úgy képzeljük el, hogy az oszlopok és sorok sorszámai 0-tól $FF-ig futnak, akkor egy-egy rekesz sorszámát úgy is megkapjuk, ha egymás után írjuk az oszlop sorszámát és a sorét.



A memóriakezelő ezeket a mérlegeket kezeli. Mindig mindent a vezérlő utasítására tesz, semmit sem kezdeményez. Összesen két feladata lehet: adatot helyez el a memóriában a vezérlő utasítására, vagy adatot hoz onnan a vezérlő számára.
Ha a vezérlő arra szólítja fel, hogy hozzon egy adatot a memóriából, akkor „lefényképezi" az adott rekeszt, és a fényképet átadja a vezérlőnek. (A lefényképezett rekesz tartalma megmarad, a memóriakezelő semmit nem vesz ki a rekeszből.) Ha viszont adatot kell elhelyeznie egy memóriarekeszben, akkor odamegy a meghatározott rekeszhez, és úgy rakja fel a súlyokat a mérlegre, hogy az a vezérlő által kívánt értéket mutassa. Ha éppen nincs semmi dolga, akkor türelmesen vár a következő parancsra.

A beosztottaknak a vezérlő parancsol, a vezérlőnek viszont a programozó. Mit tegyen tehát a programozó, ha azt szeretné, hogy megcserélődjön az 50000. és az 50001. memóriarekesz tartalma? Nem lehet rögtön azt kívánni, hogy a memóriakezelő cserélje ki őket, mert ő csak hozni és vinni tud adatot. A cserélgetés már nem szerepel a munkaköri leírásában. Egy lehetséges utasítássorozat lehet a következő:

MUNKATERV (Program)

1) A vezérlő hozassa be az 50000. memóriarekesz ($C350) tartalmát, és helyezze az X regiszterbe!
2) A vezérlő hozassa be az 50001. memóriarekesz ($C351) tartalmát, és helyezze az Y regiszterbe!
3) A vezérlő az X regiszter tartalmának megfelelő számot helyeztessen az 50001. ($C351) címre!
4) A vezérlő az Y regiszter tartalmának megfelelő számot helyeztessen az 50000. ($C350) címre!
5) Állj!

Ha a vezérlő pontosan végrehajtja a programot, akkor megcserélődik a két megjelölt rekesz tartalma. Igen ám, de hogyan lehet a vezérlőnek utasításokat adni?
Először is: nem lehet minden program minden utasítását ilyen szószátyár módon leírni. A vezérlőnek a C–64-ben úgyis csak 56 lényegileg különböző utasítást lehet adni, így az ügymenet rövidítése érdekében az utasítások neveit nyugodtan rövidíthetjük. Ennek megfelelően a program a következő lesz:

```
1) LDX   $C350
2) LDY   $C351
3) STX   $C351
4) STY   $C350
5) BRK
```

Mielőtt begépelnénk, nézzük, mit jelentenek az egyes utasítások!
Az első utasítás (LoaD to register X the contents of memory register number $C350): Töltsd az X regiszterbe a $C350 számú memóriarekesz tartalmát!
A második utasításrövidítés (LoaD to register Y the contents of memory register number $C351) jelentése: Töltsd az Y regiszterbe a $C351 számú memoriarekesz tartalmát!
A következő két utasítás ellentétes hatású. A harmadik rövidítés (STore the contents of register X to memory register number $C351) értelme: Töltsd az X regiszter tartalmát a $C351. memóriarekeszbe!
A negyedik utasítás (STore the contents of register Y to memory register number $C350) jelentése: Töltsd az Y regiszter tartalmát a $C350. memóriarekeszbe!
A BRK a BReaK szó rövidítése. Ez itt megállást, programmegszakítást jelent. Ez az utasítás tehát befejezi a gépi kódú programot, és hasonló a szerepe, mint a Basicben az END parancsnak.
A Commodore–64 gépi kódú utasításainak mindig hárombetűs rövidítése van, amely az angol nyelvű parancs rövidítése. Nem kell azonban angolul tudni ahhoz, hogy a néhány lehetséges utasítás rövidítését megjegyezzük.
Még mindig megválaszolatlan maradt a kérdés, hogyan tudjuk az utasításokat a vezérlővel közölni. Mivel számítógépről van szó, kézenfekvő, hogy az utasításokat is számok formájában kell kiadni. A programozó számokat helyez el a memóriában, majd közli a vezérlővel, hogy hol találja meg a gépi kódú programot. Nézzük meg a mi kis példaprogramunkat:

```
LDX $C350
LDY $C351
STX $C351
STY $C350
BRK
```

A programnak megfelelő számokat (azaz a kódokat) az azonnali kipróbálhatóság érdekében most felsorolom (a magyarázat a fejezet második részében található).
Tizenhatos számrendszerben: $AE, $50, $C3, $AC, $51, $C3, $8E, $51, $C3, $8C, $50, $C3, $00
Tízes számrendszerben: 174, 80, 195, 172, 81, 195, 142, 81, 195, 140, 80, 195, 0
A gépi kódú programunk tehát 13 számból áll, ezeket a számokat kell elhelyezni egymás után a memóriába POKE utasításokkal. Ezt megtehetjük egyenként:

```
POKE 49152,174 : POKE 49153,00 : POKE 49
154,195 :   POKE 49155, 172
READY.
POKE 49156,81 : POKE 49157,195 : POKE 49
158,142 : POKE 49159,81
READY.
POKE 49160,195 : POKE 49161,140 : POKE 4
9162,80 : POKE 49163,195
READY.
POKE 49164,0
READY
■
```

de megtehetjük egy rövid Basic programmal is, melynek lefuttatása után a gépi kódú program a memóriába kerül:

```
100 FOR I=0 TO 12
110 READ A
120 POKE 49152+I,A
130 NEXT I
140 DATA 174,80,195,172,81,195,142,81,195,140,80,195,0
```

A gépi kódú program most már a memóriában van a 49152. rekesztől kezdődően. A program (ha lefuttatjuk) megcseréli az 50000. és az 50001. memóriarekesz tartalmát. Ellenőrizzük! Először helyezzünk el a két címre egy-egy számot. Nézzük meg, hogy valóban bekerültek-e a számok a rekeszekbe! Futtassuk le a gépi kódú programot:

```
POKE 50000,42
READY.
POKE 50000,47
READY.
? PEEK (50000)
 42
READY.
? PEEK (50001)
  47
READY.
SYS 49152■
```

A képernyő most villant egyet, és a bal felső sarokban megjelent a READY. Lefutott a gépi kódú program. Nézzük meg, mi van a rekeszekben:

```
READY.
? PEEK (50000)
 47
READY.
? PEEK (50001)
 42
READY.
■
```

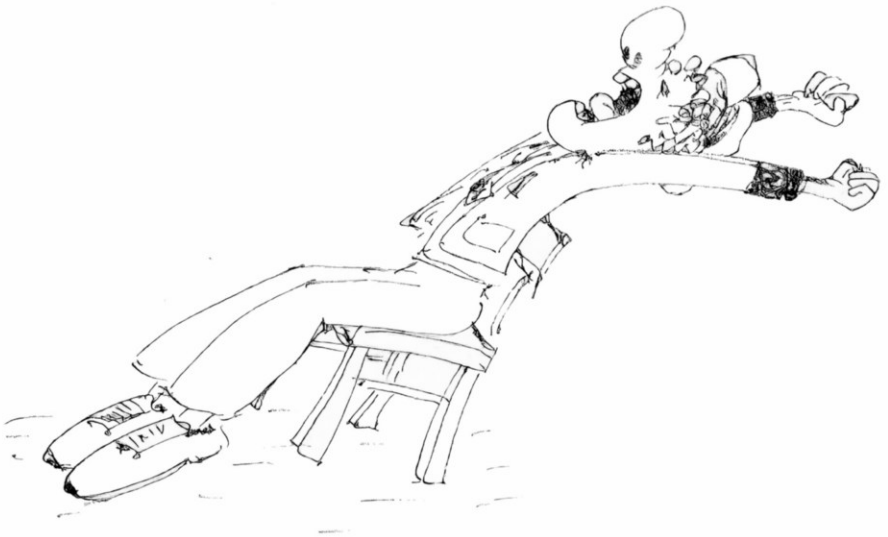
A rekeszek tartalma megcserélődött, azaz a programunk jól működött, azt csinálta, amit akartunk. Nézzük most végig részletesen!
Először elhelyeztük a memóriában a programot a 49152. rekesztől kezdődően. Ezek után az 50000. és 50001. címekre elhelyeztünk két számot, és ellenőriztük, hogy valóban beíródtak-e. Mindezek után (vagyis a program és a kezdeti adatok memóriába írása után) utasítást adtunk a vezérlőnek, hogy hajtsa végre a 49152. memóriacímen kezdődő gépi kódú programot.

A vezérlő beállítja az asztalán található PC-t (Program Counter, programszámláló) a $C000 (49152) értékre, mert ezen a címen kezdődik a gépi kódú program. Most kezdődik a program végrehajtása.
Ránéz a PC-re, ami $C000-t mutat, és utasítást ad a memóriakezelőnek, hogy hozza be neki a $C000 címen levő számot. Közben teker egyet a PC-n, így az ekkor már $C001-re mutat.
A memóriakezelő odamegy a $C000 rekeszhez, lefényképezi, és elviszi a fényképet a vezérlőnek. A fényképen természetesen egy olyan mérleg látható, mely $AE-t mutat, mivel előzőleg ezt a számot írtuk be a 49152. címre.
A vezérlő, miután megkapja a fényképet, megnézi a nagy kódkönyv $AE oldalán, hogy a $AE melyik utasítás kódja. A $AE oldalon az LDX utasítást találja.

A vezérlő ekkor a kódkönyv olvasása nélkül, halkan elkezd dünnyögni, felidézve a vezérlőiskolán tanultakat: „LDX $ . . . . , vagyis az X regiszterbe kell majd tenni egy számot valamelyik memóriarekeszből. De még nem tudom, hogy melyikből. Ahhoz, hogy megtudjam, be kell kérnem a program két következő számát. Legelőször tehát ezt teszem meg."
Ránéz a PC-re, ami $C001-et mutat, és elküldi a memóriakezelőt a $C001 memóriarekesz tartalmáért. Közvetlenül utána teker egyet a PC-n, így az már $C002-t mutat (vagyis mindig az épp sorrakövetkező, megnéznivaló címet).
A memóriakezelő odamegy a $C001. rekeszhez, lefényképezi, és a fényképet odaviszi a vezérnek. A fénykép persze egy $50 számot mutat, mert előzőleg mi, a programozók ezt írtuk a 49153. ($C001) címre.

A vezérlő tovább dünnyög: „Az első behozott számot az utolsó két pont helyére kell írni!" Le is írja a jegyzetfüzetébe: LDX $ . . 50
Hátra van még egy szám behozatala. A PC $C002-t mutat. Elküldi tehát a memóriakezelőt a $C002 rekesz tartalmáért, közben megtekeri a PC-t, ami már $C003-at mutat. A memóriakezelő lefényképezi a $C002 memóriarekesz tartalmát. A fénykép, amit odavisz a vezérlőnek, a $C3 számot mutatja a mérleg kijelzőjén. Ez persze azért van, mert előzőleg a programozó ezt a számot helyezte el a 49154. ($C002) memóriarekeszbe.
A vezérlő megkapja a fényképet, és ismét csak dünnyög: „A második számot az első és második pont helyére írom." És írja is: LDX $C350
Ránéz a cédulára: „LDX $C350 azt jelenti, hogy be kell hozatnom a $C350 memóriarekesz tartalmát, és annak megfelelően be kell állítanom az X regisztert". Ez az a pillanat, amikor anélkül, hogy akár csak egy pillantást is vetne a PC-re, utasítja a memóriakezelőt, hogy hozza be neki a $C350 memóriacím tartalmát. A memóriakezelő hűséges ku-

tya módjára elmegy a $C350 rekesz tartalmáért. A fénykép ezúttal egy olyan kijelzőt mutat, mely $2A-t jelez. Természetesen ez is azért van, mert mi, a programozók előzőleg az 50000. ($C350) memóriarekeszbe 42-t ($2A) írtunk.
A vezérlő jól megvizsgálja a fényképet, majd megpróbálja ugyanúgy beállítani az X regiszterben a mérleget. Miután végzett, megnézi, hogy egyezik-e a mérleg kijelzője a fényképen levővel, és ismét csak dünnyög: „Az X regisztert az utasításnak megfelelően beállítottam. Keletkezett összesen 4 db fénykép és egy feljegyzés a jegyzetfüzetbe. Ezekre a továbbiakban már nem lesz szükségem, hát megsemmisítem őket." Fogja a fényképeket és a feljegyzést, ledarálja a hivatali iratdarálóján, majd a csíkokat bedobja a kályhába. (Pontosan dolgozni csak jól fűtött helyiségben lehet.) Nyújtózik egy nagyot, és megelégedetten jegyzi meg: „Ismét végrehajtottam egy gépi kódú utasítást! Jöhet a következő!"



Ennyi minden történik addig, míg a számítógép végrehajtja azt az egyszerű kis utasítást, melynek a hatására az X regiszterbe is az kerül, ami az 50000. rekeszben van. A főnök persze nem tűri a lazaságot, és azonnal nekikezd a következő utasítás végrehajtásának. Ránéz a PC-re (az még mindig $C003-at mutat), elküldi a memóriakezelőt a $C003 cím tartalmáért. Teker egyet a PC-n, majd átveszi a memóriakezelőtől a fényképet. A fényképen a kijelző $AC-t jelez. A főnök fellapozza a kódkönyvben a $AC oldalt, és felírja: $AC – LDY $ . . . .
A helyzet tehát majdnem ugyanaz, mint az előbb, csak most az Y rekeszbe kell a számot tenni. Elküld először a cím utolsó két számjegyéért: LDY $ . . 51
Elküld a cím első két számjegyéért is: LDY $C351
Ezt az utasítást is végrehajtja: elküld a $C351 cím tartalmáért, ugyanolyanra állítja be az Y regisztert, majd a felesleges iratokat elégetve újból hangot ad elégedettségének és megújuló tettvágyának.
A PC $C006-ot mutat, az X, illetve Y regiszterekben pedig ugyanaz van, mint a $C350, illetve a $C351 rekeszekben: az X regiszterben $2A (42), az Y regiszterben pedig $2F (47).
A következő két utasítást az előző kettőhöz hasonlóan hajtja végre, azzal a különbséggel, hogy most nem behozatja a számokat, hanem kiküldi őket. Először behozatja az utasításkódot, megkeresi a hozzátartozó utasítást, utána behozat még két számot, amiből megtudja a címet, és ezek után végrehajtja az utasítást. Először a STX $C351 utasítás hatására a $C351 memóriarekeszbe ugyanaz kerül, mint ami az X regiszterben van, utána pedig a STY $C350 utasítás hatására a $C350 címre kerül az, ami az Y regiszterben van. Mind-

ezek után a PC $C00C-t mutat, és a program érdemi része be is fejeződött: a két memóriarekesz tartalma megcserélődött. De a főnök nem tudhatja, hogy ez nem egy nagyobb feladat része volt-e csupán. Ő mindenesetre nekikezd az újabb utasításnak. Kiküld a $C00C rekesz tartalmáért, és feltekeri a PC-t $C00D-re, majd átveszi a fényképet. A fényképen még a régebbi POKE utasítások következtében $00 látható. Ránéz a kódkönyv nulladik oldalára.

Szinte halljuk a mormogást: „BRK annyit tesz, mint BReaK, ami jelen esetben azt jelenti, hogy állj." Ezért van az, hogy a képernyő bal felső sarkában megjelenik a READY., és lehet ellenőrizni a program futását.
Foglaljuk most össze a fejezetben tanultakat! Megismertünk öt gépi kódú utasítást, amelyekkel megírtunk egy rövid gépi kódú programot. A programot ellenőriztük is. Tudjuk már, hogy ha gépi kódú programot akarunk írni, akkor először is számba kell vennünk a feladatot, majd meg kell állapítanunk, hogy milyen gépi kódú utasításokkal oldható meg. Ekkor megírhatjuk a programot az utasítások nevével, egyelőre még csak papíron:

```
LDX $C350
LDY $C351
STX $C351
STY $C350
BRK
```

Adatot tölteni mindhárom regiszterbe és mindhárom regiszterből lehet. Az A regiszterbe LDA $ . . . . utasítással tölthetünk be, onnan STA $ . . . . utasítással írhatunk ki adatot. Nyugodtan megírhattuk volna programunkat LDA $ . . . . és STA $ . . . . utasítások segítségével is.
Ha kiválasztottuk az utasításokat, akkor a megfelelő táblázatok (ld. 163. oldal) alapján minden egyes utasításhoz meg kell találnunk a neki megfelelő számsorozatot:

```
LDX $C350  — $AE, $50, $C3
LDY $C351  — $AC, $51, $C3
STX $C351  — $8E, $51, $C3
STY $C350  — $8C, $50, $C3
BRK        — $00
```

A program tehát egy számsorozat lesz:
$AE, $50, $C3, $AC, $51, $C3, $8E, $51, $C3, $8C, $50, $C3, $00
A programot úgy írjuk be a memóriába, hogy ezt a számsorozatot kihagyás nélkül, egymás után következő memóriarekeszekbe töltjük. Ezt megtehetjük POKE utasításokkal, Basic programmal vagy másképp is. A gépi kódú program ezzel futtatásra kész.
Gépi kódú programunkat egy SYS utasítással futtathatjuk. A SYS után azt a számot kell írni, amelyik memóriacímen az első gépi kódú utasítás kezdődik. A RETURN billentyű lenyomása után a gép azonnal elkezdi végrehajtani az utasításokat egészen addig, amíg olyan utasítást nem talál, amely megállítja ezt a folyamatot, és visszaadja a vezérlést a Basic rendszernek. Ilyen utasítás például a BRK, amelynek a kódja a $00. Vigyázzunk: ha a programunk végére elfelejtünk külön leállító utasítást tenni, a gép a következő memóriarekesz tartalmát akkor is megpróbálja utasításkódként értelmezni — s ha balszerencsénk van, ez sikerül is neki.
A fejezet végén ismerkedjünk meg néhány elengedhetetlenül fontos fogalommal. Vegyük például azt az utasítást, amelynek hatására az X regiszter tartalma a $C351 memóriarekeszbe kerül. Ez az utasítás: STX $C351
Az STX szócskát az utasítás mnemonikájának (emlékeztető címének) nevezik, mert ez a szócska emlékeztet az utasításra. Ez tulajdonképpen az angol nyelvű utasítás rövidítése. A Commodore–64 számítógép minden gépi kódú utasításának hárombetűs rövidítése (mnemonikája) van. A $C351 adatként szolgál, azt jelzi, hogy melyik az a memóriarekesz, ahová az X rekesz tartalmát tenni kell. Ez az utasítás operandusa. Az utasítás gépi kódja az a számsorozat, amiről a gép felismeri az adott utasítást. Ez az STX $C351 utasítás esetében három számból áll. Ezek: $8E, $51 és $C3. Van olyan utasítás is, amelynek a gépi kódja csak egyetlen szám, ilyen a BRK, amelynek gépi kódja $00. Vannak olyan utasítások is, amelyek gépi kódja két számból áll.

Ha a számsorozatot elhelyezzük a memóriában, és erről készítünk egy listát, akkor ez a programunk kódlistája:

cím	—	*tartalom*
$C000	—	$AE
$C001	—	$50
$C002	—	$C3
$C003	—	$AC
$C004	—	$51
$C005	—	$C3
$C006	—	$8E
$C007	—	$51
$C008	—	$C3
$C009	—	$8C
$C00A	—	$50
$C00B	—	$C3
$C00C	—	$00

Ez a lista a memória egy részének állapotáról informál, és teljesen korrekt módja egy gépi kódú program megadásának. Megvan viszont az a hibája, hogy nehéz belőle kihámoz-

ni az utasításokat, és nem túl szemléletes. Éppen ezért a különböző folyóiratokban és leírásokban a program ún. fordítási listáját: az egyes utasítások helyét, kódját és nevét szokás megadni — így jóval áttekinthetőbb, és minden lényeges adatot tartalmaz.

```
 cím     kód         utasítás
 ---     ---         --------
C000   AE 50 C3     LDX $C350
C003   AC 51 C3     LDY $C351
C006   8E 51 C3     STX $C351
C009   8C 50 C3     STY $C350
C00C   00           BRK
```

Gépi kódú programjainkat a továbbiakban is bevihetjük POKE utasításokkal, de egyszerűbb, ha írunk egy általánosan használható Basic betöltőprogramot erre a feladatra. (Egy ilyen program szerepel a függelékben.) Ez persze nem elegáns módja a gépi kódú programozásnak.

Elegánsabb és könnyebb is, ha monitorprogramot használunk. (Egy ilyen monitorprogram a SUPER 64—MON, melynek kezelését ld. 171. oldalon.) A monitorprogram lényegesen megkönnyíti a gépi kódú programok bevitelét, nem kell többé se kódok keresgélésével, se Basic nyelvű betöltők megírásával foglalkozni. A legelegánsabb mód pedig egy assembler program, de ehhez már valamelyest jártasnak kell lennünk a gépi kódban, így ezzel még várjunk.

Saját gépi kódú programjainkat lehetőleg a 49152—53247 területen helyezzük el! Ez az a memóriaterület, amely kifejezetten erre a célra van fenntartva, a gép tervezői is erre szánták. A könyvben szereplő további programok is ezen a területen futnak. Ha ezen a területen dolgozunk, akkor nem érhetnek váratlan, kellemetlen meglepetések.

A KÉPERNYŐ

Előző programunk legkényesebb, legnehézkesebb, legveszélyesebb és leglassúbb része az volt, amikor ellenőriztük, hogy jól működött-e. Sokkal több időt töltöttünk el a memóriarekeszek tartalmának töltésével és vizsgálatával, mint amennyi idő a program futásához kellett. Lényegesen egyszerűbb lenne, ha maga a program ki is jelezné a rekeszek tartalmát. Így nem kell PEEK utasítások kiadásával tölteni az időt, elég csak ránézni a képernyőre.
Már mindenki tudja, hogy a POKE paranccsal a memóriarekeszekbe lehet számokat helyezni. Csak annyit kell megadni, hogy melyik rekeszbe akarunk számot tenni, és természetesen azt is, hogy milyen számot. Az is érthető, hogy a szám azért nem lehet 255-nél nagyobb, mert ennél nagyobb szám egy rekeszbe nem fér be. Említettük viszont, hogy néhány POKE utasításnak furcsa hatása is lehet. Ez főleg attól függ, hogy mely rekeszbe helyezünk vele számot. Láttuk, hogy ha az 1. rekeszbe helyezünk nullát (POKE 1,0), akkor olyan galibát okozunk, amit csak újbóli ki- és bekapcsolással tudunk helyrehozni. Bizonyos rekeszekbe való írással akár a képernyőn is érhetünk el furcsa hatásokat. Egy ilyen utasítás például a POKE 1484,48. Ennek hatására a képernyő közepén megjelenik egy „0" (próbáljuk ki). Íme néhány POKE utasítás, amelyek szintén a képernyővel kapcsolatosak, próbáljuk ki ezeket is!

```
POKE 2000,24
POKE 1800,24
POKE 1801,25
POKE 1803,7
```

Ha az 1024–2023 rekeszek valamelyikébe írunk egy számot, akkor annak a képernyőn minden esetben lesz valamilyen hatása. Nyugodtan kísérletezzünk.
Természetesen a dolog akkor is működik, ha a számot nem egy Basic nyelvű POKE utasítással helyezzük a memóriába, hanem egy gépi kódú utasítással. Eddig három ilyen utasítást ismerünk: STA, STX, STY. Ha tehát az előző programunkat kibővítjük olyan utasításokkal, amelyek ennek a memóriaterületnek két rekeszébe is kiírják az értékeket, akkor nem kell PEEK-elgetni. Rögtön láthatóvá is válik a csere. Nézzük a kibővített programot!

```
 cím     kód         utasítás
 ---     ---         --------
C000   AE 50 C3     LDX $C350
C003   AC 51 C3     LDY $C351
C006   8E 51 C3     STX $C351
C009   8C 50 C3     STY $C350
C00C   8E 00 05     STX $0500
C00F   8C 01 05     STY $0501
C012   60           RTS
```

A programot az alábbi Basic programmal tudjuk a memóriába tölteni. Mindenekelőtt ezt futtassuk le!

```
100 FOR I=0 TO  18
110 READ A
120 POKE 49152+I,A
130 NEXT I
140 DATA 174,80,195,172,81,195,142,81,195,140,80,195
150 DATA 142,0,5,140,1,5,96
```

Nézzük át a gépi kódú programot! Az első utasítás hatására az X regiszterbe kerül az 50000. ($C350) memóriarekesz tartalma. A második utasítás hatására az Y regiszterbe kerül az 50001. ($C351) memóriarekesz tartalma. A következő két utasítás hatására a regiszterek tartalma fordítva kerül a rekeszekbe. (A harmadik utasítás kiírja az X regiszter tartalmát az 50001. memóriarekeszbe, vagyis most már az van az 50001. rekeszben, ami előzőleg az 50000. rekeszben volt. A negyedik utasítás az Y regiszter tartalmát írja ki az 50000. memóriarekeszbe, így ott most már az van, ami előzőleg az 50001. rekeszben volt.) Az első négy utasítás tehát megcserélte a két rekesz tartalmát. Az ötödik utasítás még egyszer kiírja az X regiszter tartalmát, de most az 1280. ($0500) memóriarekeszbe. Erről jelenleg annyit tudunk, hogy a képernyőn lesz valamilyen hatása. Az előzőhöz hasonlóan a hatodik utasítás is újból kiírja az Y regiszter tartalmát, de most az 1281. ($0501) memóriarekeszbe. Ennek is a képernyőn lesz hatása. Az utolsó utasítás egy új utasítás („ReTurn from Subroutine"): visszatérés szubrutinból. Jelenleg elég annyit tudni róla, hogy ezzel az utasítással is befejezhetjük gépi kódú programjainkat, nemcsak a BRK-val. A különbség az, hogy míg a BRK után a képernyő is törlődik, addig az RTS után nem. Mielőtt lefuttatnánk a programot, helyezzünk el két számot az 50000. és az 50001. memóriarekeszbe!

```
POKE 50000, 42
READY.
POKE 50000, 47
READY.
■
```

És most futtassuk le gépi kódú programunkat, amely megcseréli az 50000. és az 50001. memóriarekesz tartalmát, majd a képernyőre is ír valamit!

```
POKE 50000, 42
READY.
POKE 50001, 47
READY.
SYS 49152
READY.
■
```

Ha eddig mindent pontosan csináltunk, akkor a képernyőn most egymás mellett megjelent egy csillag és egy osztásjel. Futtassuk le még egyszer a programot!

```
POKE 50000, 42
READY.
POKE 50001, 47
READY.
SYS 49152
READY.
SYS 49152
READY.
■
```

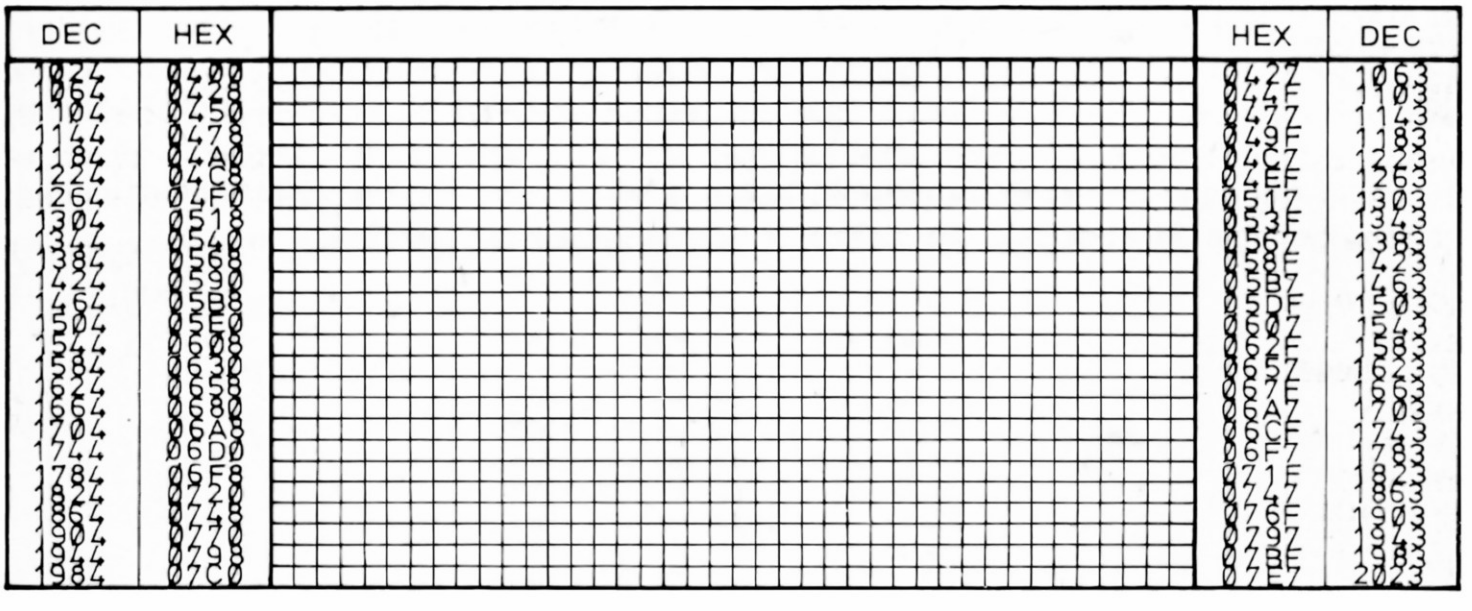
A két jel megcserélődött. Figyeljük meg, hogy akárhányszor futtatjuk is le a programot, a két jel mindannyiszor helyet cserél! Szépen látszik tehát, hogy a programunk valóban csereberél, ezért van a képernyőn változás.
S most nézzük, hogyan is működik a képernyő! Mit tudunk eddig a képernyőről? Először is azt, hogy a képernyő tartalma attól függ, milyen számokat helyezünk el az egyes memóriarekeszekben. Másodszor pedig azt, hogy ezek a sajátos rekeszek az 1024 és 2023 közöttiek. Ez pedig összesen ezer darab rekesz. Kísérletezzünk most ismét POKE utasításokkal!

```
POKE 1030,0
POKE 1400,0
POKE 1823,0
POKE 2007,0
```

A képernyőn négy különböző helyen jelent meg egy @. Próbálkozzunk meg most más címekkel is (természetesen csak olyanokkal, amelyek 1024 és 2023 között vannak). Bárhová is írunk nullát, a képernyőn valahol megjelenik egy @. Próbáljuk végig újból a memóriacímeket, de most írjunk mindenhova 1-et! Azt tapasztaljuk, hogy bárhová is írtunk 1-et, mindig egy A betű jelent meg a képernyőn valahol. Levonhatjuk az első tanulságot: hogy milyen betű vagy jel jelenik meg, attól függ, hogy milyen számot írunk a memóriarekeszbe. Ezek után sejthető, hogy az adott karakter (jel, szám vagy betű) megjelenési helye attól függ, hová írjuk a neki megfelelő számot. Nézzünk most egy rövid Basic programot, amely ezt a kérdést is megvilágítja!

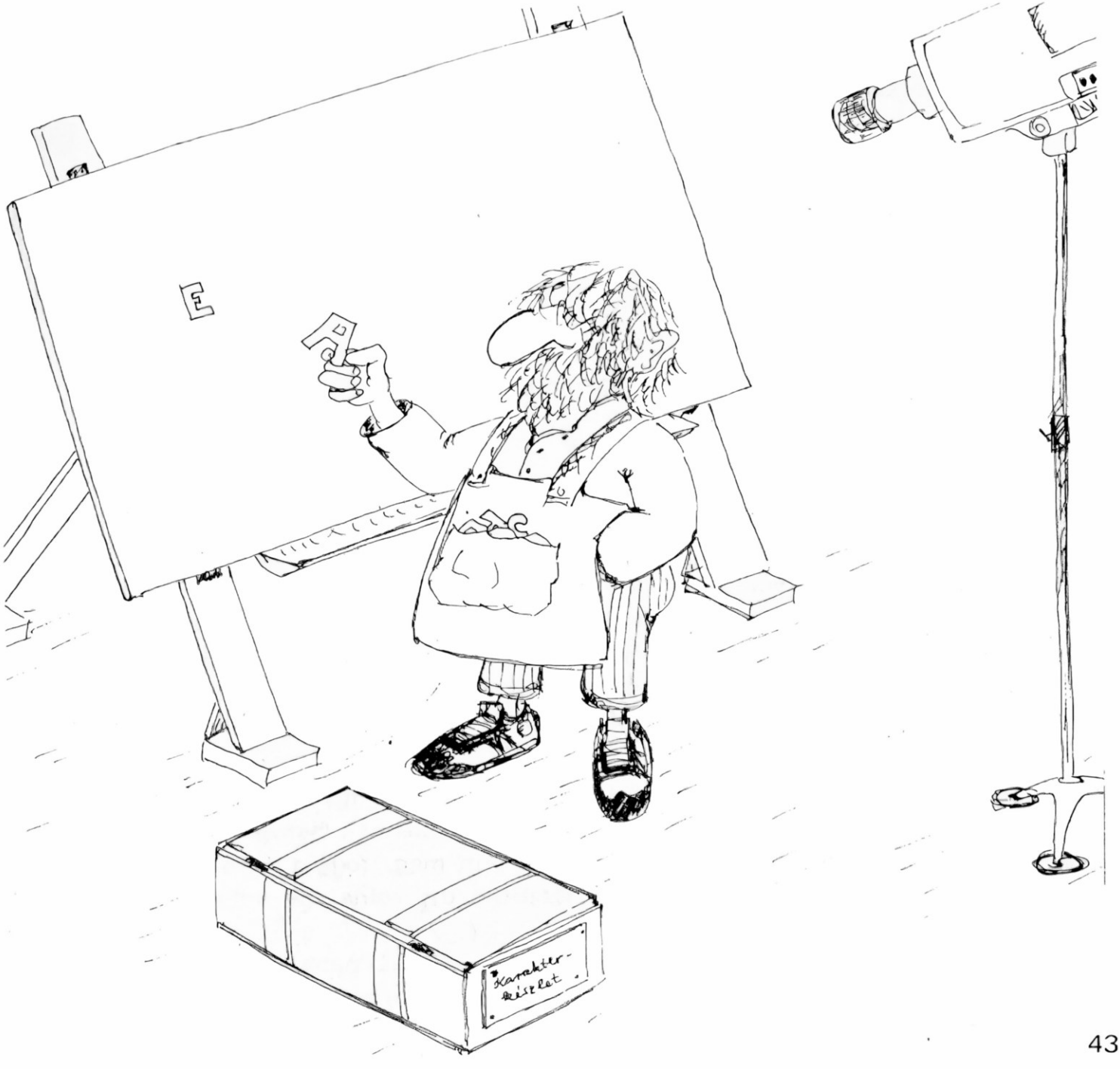
```
100 FOR I=1024 TO 2023
110 POKE I,42
120 NEXT I
```

Futtassuk le ezt a programot, és összegezzük, amit tudunk! A legelső csillag a képernyő bal felső sarkába került, majd minden sort balról jobbra haladva töltöttünk meg csillagokkal. Az utolsó éppen a jobb alsó sarokba került. Ezek szerint a képernyőn minden egyes betűpozíciónak megfelel egy memóriarekesz. Ha az illető memóriarekeszbe beírunk egy számot, akkor a neki megfelelő képernyőpozíción megjelenik egy jel. Ez a jel attól függ, milyen számot írtunk a memóriarekeszbe. A képernyőn 25 sorban és 40 oszlopban helyezhetünk el betűket, és ez pontosan 1000 karakterpozíció. Ennek az 1000 karakterpozíciónak felel meg a memória 1024—2023 ($0400—$07E7) közötti ezer rekesze.



A függelékből megtudhatjuk, hogy az egyes beírt számoknak milyen karakterek felelnek meg (184. old.). Nyugodtan kísérletezhetünk a képernyővel, a számítógépnek nem árt. Azt viszont nagyon fontos megjegyezni, hogy a különböző karakterek képernyőkódjai nem azonosak a Basicből már ismert ASCII-kódjaikkal.

Vajon miért pont a képernyőn van ennek az 1000 memóriahelynek hatása? Ha egy memóriarekeszben elhelyezett érték a képernyőn jelenik meg mint valamilyen karakter, akkor valakinek a keze van a dologban. Ezt a valakit úgy hívják, hogy VIC (Video Interface Controller), ő a vezérlő egyik beosztottja. Helyesebb azonban, ha inkább külső beosztottnak nevezzük, mert a vezérlő soha nem találkozik vele. Parancsolni annál inkább tud neki. Ebből is látszik, hogy milyen jól szervezett ez a hivatal. A parancsoláshoz itt már nem kell személyes kapcsolat, elég egy kis memória. Arról ugyanis még nem beszéltünk, hogy mi van a memóriaterem nagy memóriafalának túloldalán. Az egyes memóriarekeszek tartalmát a „másik oldalról" is meg lehet nézni. Vagyis bárki odamehet és megnézheti, hogy például az 1976. memóriarekeszben mennyit mutat a mérleg. Ez egyben felfogható parancsnak is. A VIC munkaköri leírásában az szerepel, hogy állandóan figyelnie kell egyes rekeszeket, és a rekeszek tartalmától függően egy táblára betűket kell kiraknia.



A táblára csak 25 sorban és 40 oszlopban lehet karaktereket tenni, és összesen 256 különböző karakter közül válogathatunk. Profi programozók a karakterkészletet is meg tudják változtatni, de ez már ólomöntés nehézségű probléma, így könyvünk nem részletezi.

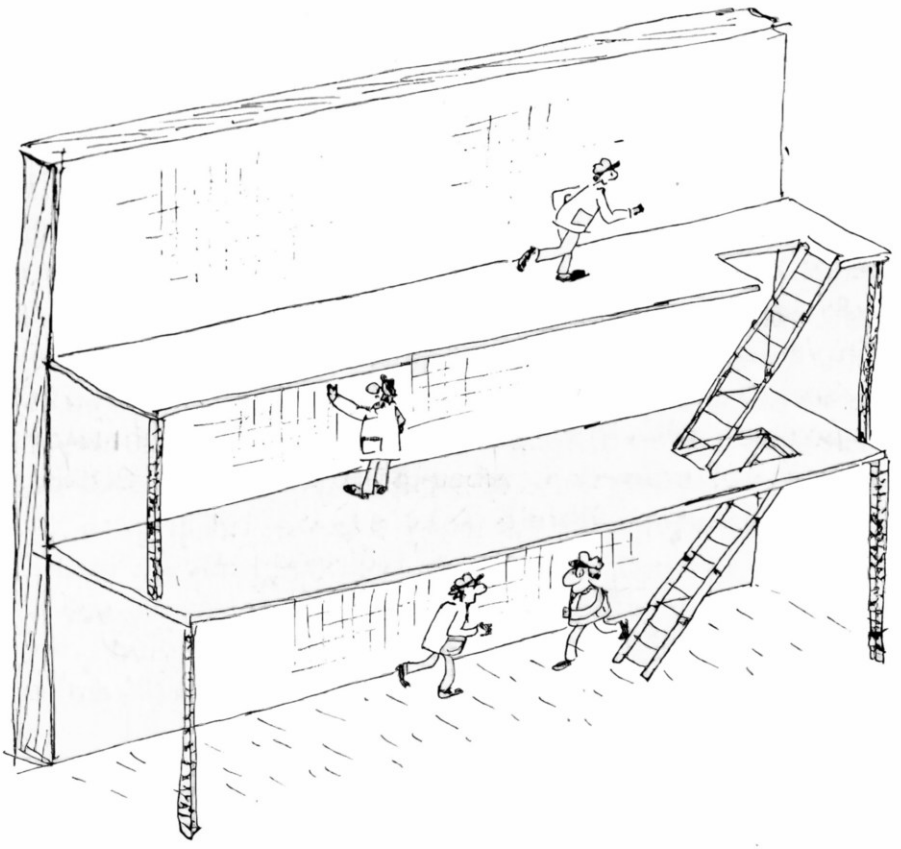
Mint a hivatal alkalmazottainak többsége, a VIC is dünnyögve végzi a feladatát: „A tábla színe a $D021 címen van meghatározva. Odamegyek a $D021 rekeszhez, és megnézem, hogy mennyit mutat a mérleg. Aha, $F2-t. Ebből csak a második számjegy lényeges. A második számjegy 2. Most elmegyek a színkódtáblázathoz, és megnézem, hogy a 2 milyen színt jelöl. A kettő a piros szín kódja, tehát a piros színű táblát fogom használni. A keret színét a $D020 címen találom meg. Odamegyek a $D020 rekeszhez. A mérleg $F0-t mutat. Ebből nekem csak a második számjegy lényeges, az pedig nulla. A színkódtáblázat szerint a nulla a fekete színt jelenti, tehát a keret fekete színű lesz. Az első sor első karakterének színe a $D800 címen van meghatározva. Ott $F4 van, amiből a négyes a bíbor kódja, így bíbor színű karakter kell majd. Az első sor első karaktere a $0400 címen van meghatározva. Ott azt találom, hogy nulla. A karakterkód-táblázat szerint ez egy @ jel. Az első sor első karakterpozíciójára tehát egy bíbor színű @ jelet teszek. Az első sor második karakterének színét a $D801 címen tudom meg. Ott $F1 van, amiből az egy a fehér kódja, így a karakter fehér lesz. Az első sor második karaktere a $0401 címen van meghatározva. Az ott levő $01 a táblázat szerint az A betű kódja. Az első sor második karaktere tehát egy fehér színű A betű lesz.'' És így tovább. Mire elérkezik az utolsó sor utolsó karakteréhez, már csak tőmondatokat dünnyög:

„Sor: 25
Oszlop: 40
Szín: $DBE7
$DBE7 – $F9
9=barna
Karakter: $07E7
$07E7 – $20
$20=semmi (szóköz) – vagyis a szín nem számít.
25. sor, 40. oszlop – semmi.''

Ezzel készen is van a kép. Elveszi az előző képet a videokamera elől, és helyébe teszi az újat. Nézi egy darabig a művét, aztán eszébe jut, hogy lazsálni nincs idő. Nekilát tehát a következő kép elkészítésének. Akár hisszük, akár nem, másodpercenként kb. ötven képet készít. Döbbenetes! Nem unja meg, nem lázadozik rabszolgasorsa ellen, nem kér fizetésemelést, csak dolgozik bekapcsolástól kikapcsolásig. De a hivatal alkalmazottai már csak ilyenek... A videokamerából aztán egy vezeték vezet a Commodore-64 számítógép hátlapjához, ahonnan egy újabb vezetékkel televíziós készülékhez vagy akár videomonitorhoz is lehet csatlakozni.

A hivatal sok-sok alkalmazottja dolgozik hasonló módon. Figyelik a memóriát (a másik oldalról), és munkaköri leírásuknak megfelelően intézkednek hatáskörükben. Még soha egyetlen Commodore-64 számítógépben sem történt meg, hogy a memóriafal túloldalán szaladgáló alkalmazottak közül valaki is összeütközött volna egy munkatársával. Pedig itt aztán van sebesség!

Az alkalmazottak munkáját a memória irányítja, a memóriát a memóriakezelő kezeli a főnök parancsai alapján, a főnök pedig a programozó programjainak engedelmeskedik – az



összes alkalmazott munkáját a programozó irányítja programjával. Könyvünk már csak a billentyűzet- és joystick-kezelő munkáját ismerteti, de hangot is hasonlóan lehet csiholni. A hangszerek kezelése egy SID (Sound Interface Device) nevű alkalmazott feladata. Az ügyes programozó még arra is rá tudja venni a SID-et, hogy kimondja:

Most, hogy már ismerjük a képkészítés csínját-bínját, nézzünk egy öncélú feladatot: hogyan lehet pontosan a képernyő középső sorába, annak is a közepébe kiírni négy csillagot? A csillag kódja 42 (=$2A). Megkeressük, hogy a képernyő középső sorában a középső négy karakternek mely memóriacímek felelnek meg: 1522, 1523, 1524 és 1525 (tizenhatos számrendszerben: $05F2, $05F3, $05F4 és $05F5). Tegyük fel, hogy piros csillagokat szeretnénk kiírni. A piros szín kódja olyan hexadecimális szám, amelynek a második jegye 2. Lehet tehát $02, $12 és így tovább $F2-ig bármi. A színek megállapításakor a VIC csak a második jegyet veszi figyelembe. Mivel ez a második jegy 0-tól F-ig terjedhet, éppen 16-féle szín valósítható meg a gépünkön. A lehetséges számok közül használjuk mindig azt, amelyiknek 0 az első jegye; a piros szín kódjaként tehát a $02-t!
A kívánt karakterpozíciók színeit az alábbi címeken kell meghatározni: 55794, 55795, 55796, 55797 (tizenhatos számrendszerben $D9F2, $D9F3, $D9F4, $D9F5). Programunk tehát a következőképpen épül fel:

```
 cím     kód        utasítás
 ---     ---        --------
C000   A9 02        LDA #$02
C002   8D F2 D9     STA $D9F2
C005   8D F3 D9     STA $D9F3
C008   8D F4 D9     STA $D9F4
C00B   8D F5 D9     STA $D9F5
C00E   A9 2A        LDA #$2A
C010   8D F2 05     STA $05F2
C013   8D F3 05     STA $05F3
C016   8D F4 05     STA $05F4
C019   8D F5 05     STA $05F5
C01C   60           RTS
```

A program betöltését az alábbi Basic program futtatásával végezhetjük el:

```
100 FOR I=0 TO 28
110 READ A
120 POKE 49152+I,A
130 NEXT I
140 DATA 169,2,141,242,217,141,243,217,141,244,217,141,245,217
150 DATA 169,42,141,242,5,141,243,5,141,244,5,141,245,5,96
```

Az LDA #$02 utasítás (ilyen még nem szerepelt) hatására az A regiszterbe $02 kerül. Idáig is ismertünk egy LDA utasítást, amelynek a kódja $AD volt. Ha a vezérlő egy $AD kódot talált, akkor tudta, hogy ez az LDA $ . . . . utasítás kódja. Behozta a következő két számot, és ezekből megtudhatta, hogy melyik memóriacímről kell az A-ba tölteni. Ez az LDA viszont másmilyen. Már abból is látszik, hogy csak kétbájtos, nem hárombájtos. De a neve ennek is LDA, tehát ez is az A regiszterbe tölt. A kódja $A9. Nézzük, mi a hatása. Ha a vezérlő egy ilyen kódot talál, akkor leírja, hogy: LDA #$ . .
Abból, hogy az utasítás kódja $A9 volt, tudja, hogy már csak egy számot kell behozatnia, és kész is az utasítás. Ha tehát behozatta a következő számot, akkor leírja: LDA #$02
Ez pedig most az jelenti, hogy be kell állítani az A regisztert úgy, hogy a mérleg $02-t mutasson. Most tehát nem valamelyik memóriarekeszből kell másolni, hanem közvetlenül a vezérlőnek kell beállítania egy meghatorzott számot. Mindkét utasítás LDA utasítás, azaz LoaD A (töltés az A regiszterbe), de a működésük szempontjából egyáltalán nem azonosak. Azt szokás erre mondani, hogy más a címzési módjuk. Azt az utasítást például, hogy: LDA $C000 abszolút címzésű LDA-nak nevezzük. Az abszolút címzésű utasítások

hárombájtos utasítások. Az első bájt az utasítás kódja, a másik kettő pedig azt az abszolút címet adja meg, amelyik memóriarekesz tartalmával valamit tenni kell.

Az LDA #$02 utasítást közvetlen címzésű LDA-nak nevezzük. Ez kétbájtos utasítás, ahol az első bájt megintcsak az utasítás kódja, a második bájt viszont közvetlenül az a szám, amelyet A-ba kell tenni. (A Commodore-64 gépi nyelvében összesen 13 címzési mód lehetséges. Van még mit tanulnunk! Az utasítások többségénél nem lehet mindegyik címzési módot használni, csak néhányat.)

Bekerül tehát A-ba a $02 — a piros szín kódja. A következő négy utasítás kirakja ezt négy memóriacímre, azaz mind a négy memóriacímre $02 kerül. Ezek a címek a középső sor középső négy karakterének színét határozzák meg. Ezzel előírtuk, hogy ami ott megjelenik, mind piros legyen. A következő utasítás, amely ismét közvetlen címzésű LDA, módosítja az A tartalmát $2A-ra, ez a csillag kódja. A következő négy abszolút címzésű STA utasítás ezt az értéket helyezi el a képernyőmemóriában, a kívánt helyre. Már azt is meghatároztuk, hogy milyen karaktereket kell megjeleníteni. Az utolsó utasítás a burkolt címzésű (ez is egy címzési mód) RTS, amely befejezi a programot. Az egybájtos utasítások majdnem mind burkolt címzésűek, ami azt jelenti, hogy nincs hozzájuk külön cím: az utasítás kódja önmagában is tartalmazza a végrehajtáshoz szükséges valamennyi információt.

Ez a program tehát négy darab piros csillagot helyez el a képernyő közepén. Itt az ideje, hogy lefuttassuk, írjuk be: SYS 49152.

Már az sem okozhat gondot, ha az egész képernyőt piros csillagokkal akarnánk megtölteni. Nézzük meg, hány utasítás és mekkora memória szükséges ehhez:

	1	db	LDA #$02	utasítás =	2 bájt
	1000	db	STA $....	utasítás =	3000 bájt
	1	db	LDA #$2A	utasítás =	2 bájt
	1000	db	STA $....	utasítás =	3000 bájt
+	1	db	RTS	utasítás =	1 bájt
	2003	db	gépi kódú utasítás	=	6005 bájt

Nincs valamilyen egyszerűbb és rövidebb megoldás?

CIKLUSSZERVEZÉS

A Basic nyelv feltétlen vezérlésátadásra szolgáló GOTO utasításának gépi kódú megfelelője JMP (az angol „Jump" szó rövidítése, magyarul azt jelenti: „ugorj"). Nézzünk az utasítás használatára egy rövid programot:

```
 cím     kód        utasítás
 ---     ---        --------
C000   A2 30        LDX #$30
C002   A0 31        LDY #$31
C004   4C 0D C0     JMP $C00D
C007   AE 00 04     LDX $0400
C00A   AC 01 04     LDY $0401
C00D   8E 01 04     STX $0401
C010   8C 00 04     STY $0400
C013   4C 07 C0     JMP $C007
```

Az első két utasítással feltöltjük az X és Y regisztereket, majd egy JMP $C00D utasítás hatására a $C00D címtől folytatódik a program. Itt a két tároló utasítás hatására a képernyő bal felső sarkába kerül egy 0 és egy 1. Majd ismét egy JMP utasítás következik, a program a $C007 címen folytatódik. A $C007 címtől kezdődően a négy utasítás megcseréli a képernyő bal felső sarkában levő két jelet, majd ismét elérkezünk a JMP $C007 utasításhoz. És a folyamat végtelen sokszor megismétlődik. A következő Basic programmal betölthetjük, majd utána kipróbálhatjuk a programot (és kaphatunk még egy benyomást a gépi kódú programok sebességéről). A program csak a RUN/STOP és RESTORE billentyűk együttes megnyomásával, vagy a gép kikapcsolásával állítható le.

```
100 FOR I=0 TO 21
110 READ A
120 POKE 49152+I,A
130 NEXT I
140 DATA 162,48,160,49,76,13,192,174,0,4
150 DATA 172,1,4,142,1,4,140,0,4,76,7,192
```

Ha a vezérlő egy $4C kódot kap, akkor behozatja a következő két számot, azokat abszolút címként kezeli, és erre az értékre állítja be a PC-t. Amikor hozzákezd a következő utasításhoz, és ránéz a PC-re, az már a módosított értéket mutatja, tehát onnan hozatja be a következő utasításkódot. Így aztán írhatunk olyan programokat, amelyek végtelen sokszor futnak le, egészen addig, míg a RUN/STOP és RESTORE billentyűk együttes leütésével, vagy a gép kikapcsolásával le nem állítjuk őket. A vezérlő nem veszi észre, hogy állandóan ugyanazt kell csinálnia. Még az az egyszerű program is lefut, hogy:

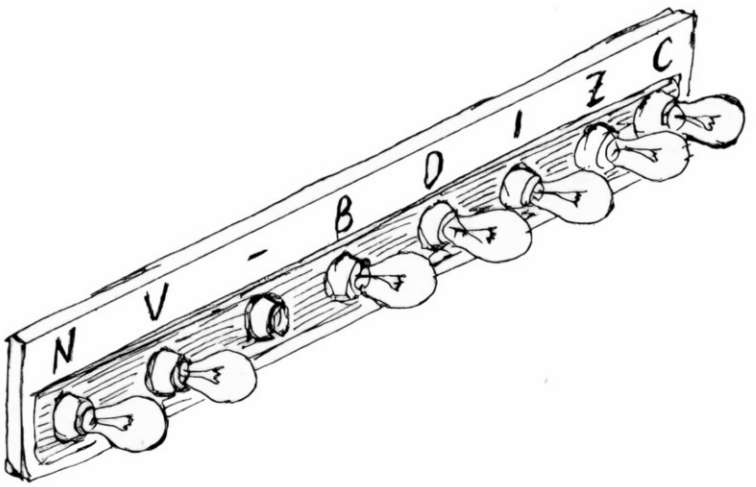
```
 cím     kód        utasítás
 ---     ---        --------
C000   4C 00 C0     JMP $C000
```

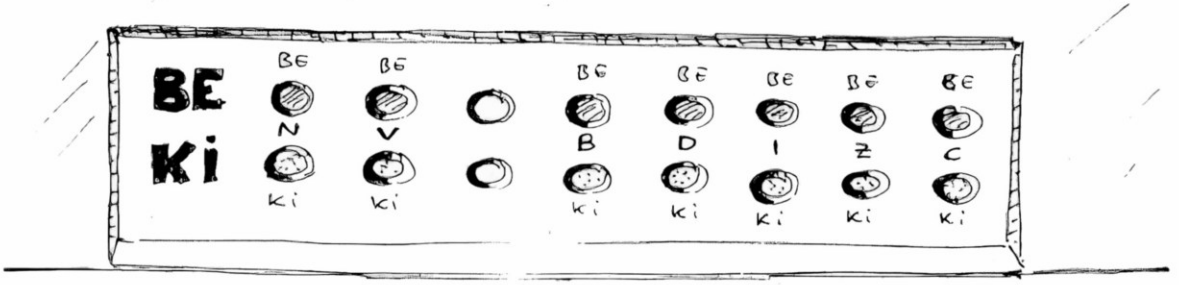
A betöltő program:

```
100 POKE 49152,76
110 POKE 49153,0
120 POKE 49154,192
```

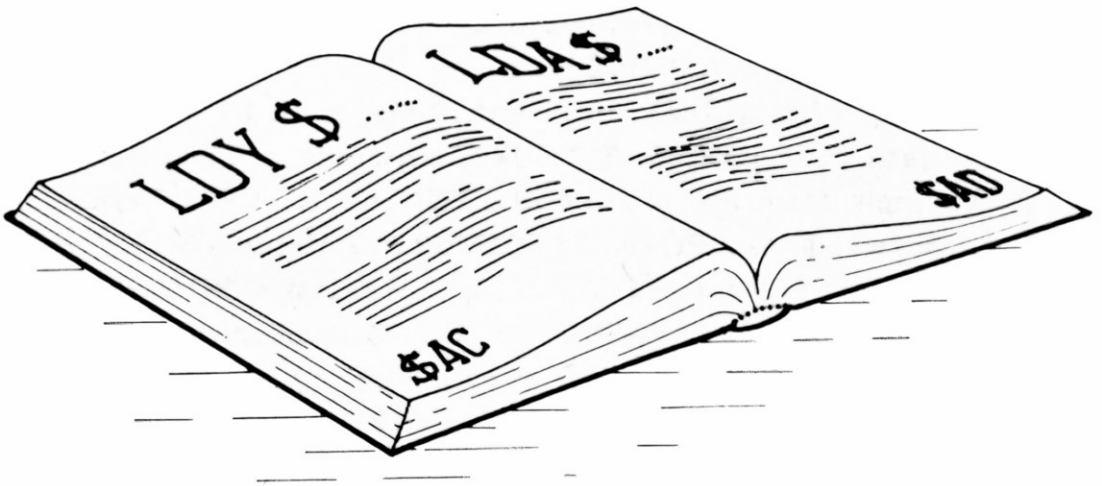
A vezér behozatja a $C000 címen levő $4C értéket, közben teker egyet a PC-n, így az $C001-et mutat. Értelmezi a kódot, és felírja, hogy: JMP $ . . ..
Ránéz a PC-re, az $C001-et mutat, behozatja a $C001 címen levő számot, teker egyet a PC-n, és írja: JMP $ . . 00
A PC most $C002-t mutat, így onnan hozatja be a következő számot, természetesen megint tekerve egyet a PC-n. Újból ír: JMP $C000
Most dünnyögni kezd: „JMP $C000 azt jelenti, hogy a PC-t be kell állítani $C000-ra." Így a $C003-át mutató PC-t beállítja $C000-ra. Ledarálja, majd elégeti az okmányokat, és szól: „Ismét végrehajtottam egy gépi kódú utasítást!" Ezek után nekikezd a következő utasítás végrehajtásának. Ránéz a PC-re. Az $C000-t mutat, így a $C000 címről hozatja be a kódot, és újra meg újra végrehajtja a JMP $C000 utasítást. Akárhányszor hajtja is végre, nem veszi észre, hogy egy teljesen értelmetlen, öncélú tevékenységet folytat. De hát neki nem az a dolga, hogy véleményt mondjon, netán bíráljon, hanem az, hogy végrehajtsa a parancsokat. Ha elindítjuk a programot egy SYS 49152 utasítással, akkor akár üdülni is elmehetünk, a számítógépben még hazaérkezésünkkor is fut majd a program. Egészen addig, míg ki nem kapcsoljuk vagy a RUN/STOP és a RESTORE segítségével meg nem állítjuk a gépet. Az ilyen, végtelenszer végrehajtott programokat vagy programrészeket végtelen ciklusoknak nevezzük.

Most már sok mindent tudunk a vezérlőről, de van néhány, eddig még ismeretlen tevékenysége is. Ezek közül most a jelzőbitek kezeléséről beszélünk. A jelzőbit egyszerű lám-





pának is tekinthető, amely vagy ég, vagy nem ég. A vezérlő összesen 7 jelzőbitet kezel, mégpedig be- és kikapcsoló nyomógombok segítségével. Ha megnyomja egy jelzőbit bekapcsológombját, akkor akár égett előzőleg, akár nem, ezt követően égni fog. A kikapcsológomb megnyomását követően pedig akár égett előzőleg, akár nem, azután nem fog égni. A nyomógombokat a vezérlő kezeli minden utasítás végrehajtását követően. A nagy, 256 ($FF) oldalas kódkönyvben a kódhoz tartozó utasítás leírása után az is szerepel, hogy az utasítás végrehajtását követően mely jelzőbiteket és hogyan kell az eredménytől függően beállítania. Nézzük meg például a nagy kódkönyv $AD oldalát.



Leírás

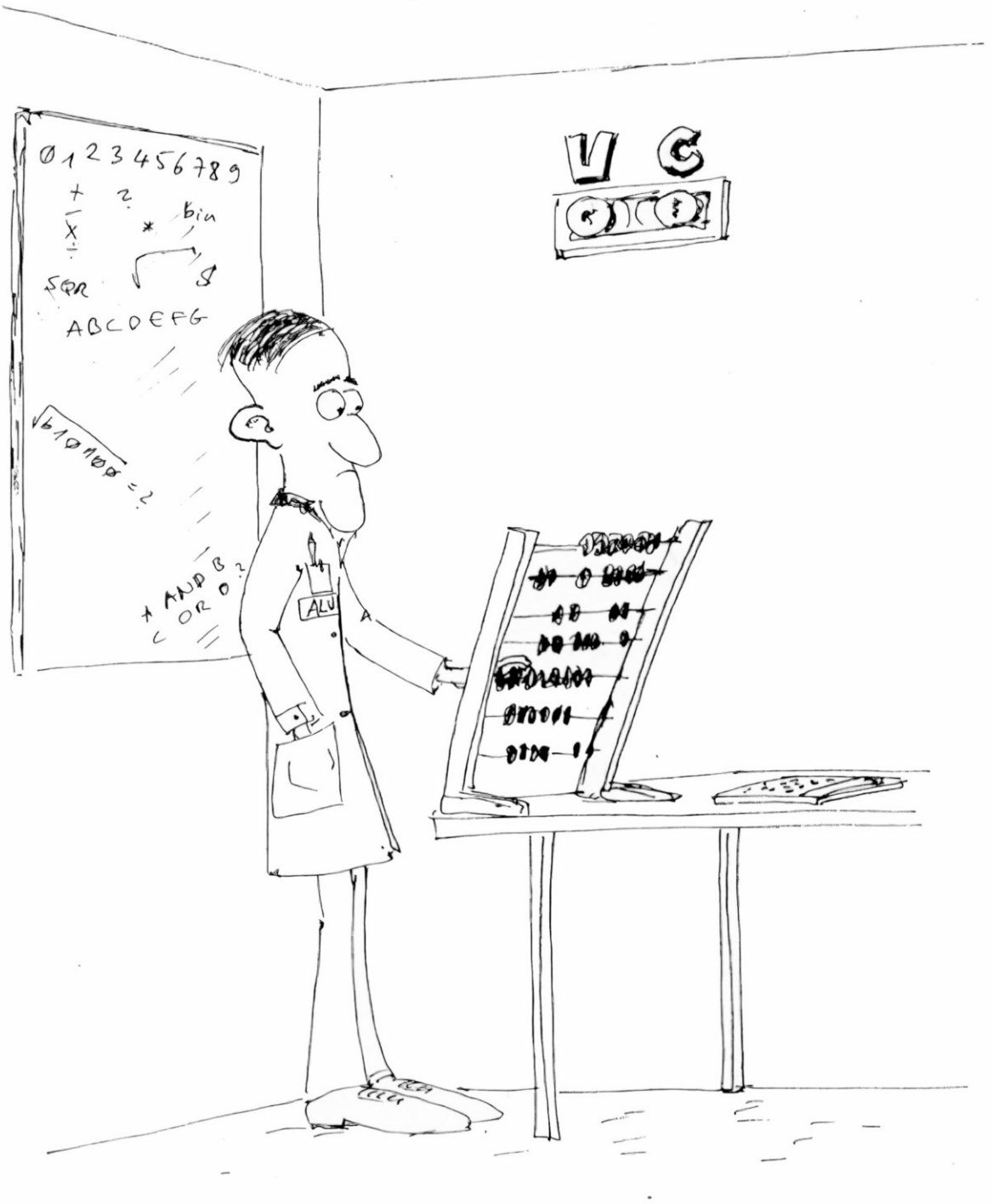
Hozassuk be a következő két számot a következő két memóriacimről! Az első behozott számot írjuk a harmadik és a negyedik pont helyére, a másodikat pedig az első és a második pont helyére! Az így megadott memóriacimről hozassuk be az ott lévő értéket, és állitsuk be annak megfelelően az A regisztert!

Jelzőbitek kezelése

1) Ha az A regiszterbe $ØØ került, akkor nyomjuk le a Z-be nyomógombot! Ellenkező esetben nyomjuk le a Z-ki nyomógombot!
2) Ha az A regiszterbe olyan érték került, mely kettes komplemens kódban negativ (tehát a legfelső bitje 1), akkor nyomjuk le az N-be nyomógombot! Ellenkező esetben (ha tehát a legfelső bit Ø) nyomjuk le az N-ki nyomógombot!

A többi jelzőbittel ennél az utasitásnál nem kell és nem is szabad foglalkozni! Azok maradjanak továbbra is olyan állapotban, amilyenben voltak!

Ha tehát egy utasítást követően a Z jelzőbit ég, akkor ez azt jelenti, hogy az utasítás eredményeként valahol $00 jelent meg, az eredmény zérus lett. Ha a Z nem ég, akkor ez értelem szerint azt jelenti, hogy az utasítás végrehajtásának eredménye nem lett zérus. Az N jelzőbit arról ad tájékoztatást, hogy az utasítást követően kettes komplemens kódban negatívnak értelmezhető szám keletkezett-e.

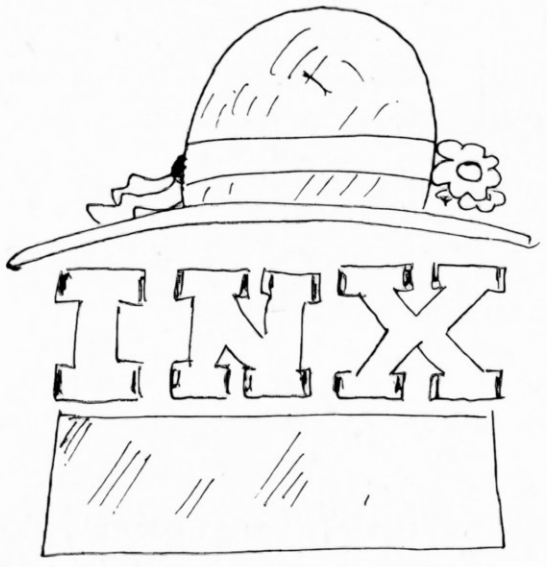


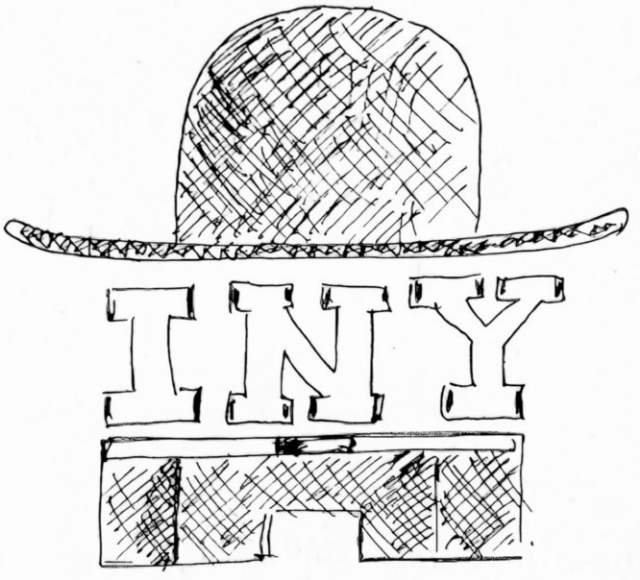
Minden hivatalban kell lennie legalább egy okos embernek. Lehet bármilyen rend, lehet bármilyen szigorú a főnök, ha nincs okos ember, akkor a hivatal fabatkát sem ér. A Commodore—64 és általában minden számítógép központi hivatalának okos emberét ALU-nak hívják (Arithmetical Logical Unit = aritmetikai és logikai egység). A számítógépben egyedül az ALU tud számolni. Egy ilyen fontos ember persze csakis a főnök közvetlen alárendeltje lehet. Ő is szolgalelkű a végsőkig, mindent kiszámol, amit a főnök kér, de semmi többet. Persze az, hogy számolni tud, kicsit túlzás, hiszen nem tud például szorozni, osztani vagy gyököt vonni, de azért a legalapvetőbb műveletet, az összeadást ismeri. Tud továbbá kivonni, és ismer néhány matematikai logikai műveletet. Kérésre például bármikor megnöveli az X rekesz tartalmát eggyel.

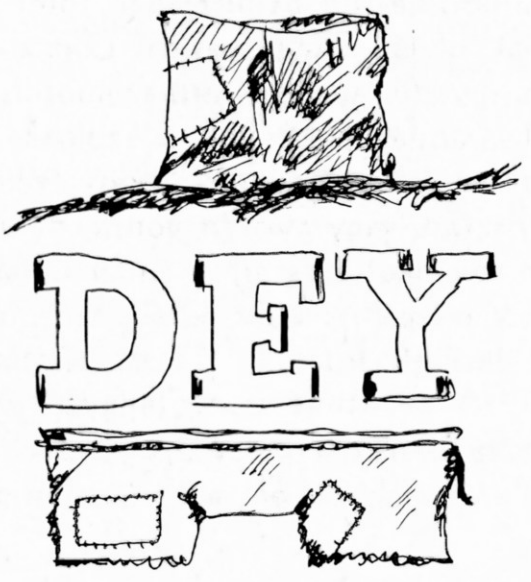
Ha azt akarjuk, hogy az X regiszter tartalma eggyel növekedjen, azt egy INX („INcrease register X!" = növeld meg (eggyel) az X regiszter tartalmát) paranccsal idézhetjük elő. Az utasítás burkolt címzésű, egybájtos, a kódja pedig $E8 (232). Ha a vezér kap egy ilyen kódot, akkor felírja egy cédulára, hogy INX. Ezek után előveszi saját polaroidját, és le-

fényképezi az X regisztert, majd a felvételt a cetlivel együtt átadja az ALU-nak. Az ALU ekkor magára zárja az ablakot, s egy darabig csak a szobájából kiszűrődő halk morgás jelzi, hogy erősen koncentrál, dolgoztatja az agyát. Aztán egyszercsak nyílik az ablak, és a beavatottak szelíd, mosolygó, árnyalatnyit büszke, de nem lenéző arckifejezésével átnyújt egy újabb felvételt, amit már ő csinált egy saját mérlegéről, ezen azonban már az eggyel megnövelt érték látható. A vezér ekkor mindig megjegyzi: „Nos hát hiába. Ehhez az én fejem már túl öreg." Ennél több bizalmaskodásra már nincs idő. Fogja a fényképet, és ennek mintájára állítja be az X rekeszben a mérleget. Most is ledarálja és elégeti az iratokat, elégedetten nyújtózkodik, majd tettrekész megjegyzést tesz. Az előírt gombokat sem felejti el megnyomni (sokszor nem is tudja, milyen komoly fejtörést okoz az ALU-nak). Ha az ALU kap egy $AA értéket, akkor amit visszaad, az $AB lesz, vagyis eggyel több. Ha $47-et kap, akkor $48-at. De néha az is megtörténik, hogy $FF-et kap, mégpedig olyan célból, hogy növelje meg eggyel. $FF+1=$100. Ezt azonban csak kilenc biten tudná tárolni. Az ALU ilyenkor az alsó nyolc bitet adja vissza, vagyis $00-t, és nem törődik a kilencedikkel. Ezért van az, hogy ha INX előtt az X regiszter tartalma $FF volt, akkor utána $00 lesz. És természetesen ilyenkor a vezér mindig lenyomja az asztalán levő „Z–BE" jelű nyomógombot, ezzel is jelezve, hogy most valami olyan történt, aminek nulla lett az eredménye. Egyébként pedig a „Z–KI" gombot nyomná meg. Az ALU szobájában csak két jelzőbit kapcsolói vannak meg, a V és a C jelű lámpák, ezeket az ALU is tudja kapcsolgatni.

Az ALU segítségével tehát bármikor megnövelhető az X rekesz értéke eggyel. Csupán







azt kell megjegyezni, hogy míg minden számból eggyel nagyobb lesz az INX hatására, addig a $FF-ből nulla lesz, és ekkor a .Z jelű lámpa is kigyullad.
Természetesen az ALU nemcsak növelni, hanem csökkenteni is tud eggyel. Ehhez az kell, hogy a főnök egy DEX („DEcrease register X!" = csökkentsd (eggyel) az X regisztert) utasítást kapjon, aminek a kódja $CA (202). Ez az utasítás ugyanúgy hajtódik végre, mint az INX, csak most eggyel kisebb érték kerül az X-be. Itt is adódhat probléma, mégpedig akkor, ha nullát kell csökkenteni (0—1= -1). Ebben az esetben jól jön a kettes komplemens kód. Kettes komplemens kódban a −1-et $FF jelöli, ilyenkor tehát az ALU $FF-et ad vissza.

Ne feledjük: INX-nél $FF után $00 következik, DEX-nél $00 után $FF.
Természetesen mindezt az Y regiszterrel is meg lehet tenni. Az INX párja az INY, kódja $C8. A DEX párja a DEY, kódja $88.

Az A regiszterhez nincs burkolt címzésű növelő vagy csökkentő utasítás. Ebben például különbözik az eddig teljesen egyformának ismert három regiszter.
Most pedig tanuljunk meg egy új címzési módot, melynek jelentősége csak a feltételhez kötött vezérlésátadó utasításokkal együtt érthető meg. Lássuk tehát, hogyan tudjuk a képernyő hatodik sorát teleírni csillagokkal anélkül, hogy több mint 40 utasítást kellene kiadnunk!

```
cím     kód        utasítás
---     ---        --------
C000  A9 2A        LDA #$2A
C002  A2 28        LDX #$28
C004  9D C7 04     STA $04C7,X
C007  CA           DEX
C008  D0 FA        BNE $C004
C00A  60           RTS
```

A Basic betöltő:

```
100 FOR I=0 TO 10
110 READ A
120 POKE 49152+I,A
130 NEXT I
140 DATA 169,42,162,40,157,199,4,202,208,250,96
```

A Basic betöltő futtatása után írjuk be: SYS 49152

A programból először is megállapíthatjuk, hogy az LDX utasításnak is van közvetlen címzésű változata, sőt az LDY-nak is van. A $C004 címen viszont furcsa utasítást találunk. Az utasítás az A regiszter tartalmát írja ki egy memóriarekeszbe. De nem a $04C7 rekeszbe, mert a $04C7-hez először még hozzáadódik az X rekesz pillanatnyi értéke is, és ez lesz a cím. Az utasítás hatására tehát a gép megnézi az STA utáni címet, megnézi az X tartalmát, a kettőt összeadja, és ez az összeg lesz az STA utasításhoz tartozó cím. Kezdetben, amikor az X értéke $28, az STA $04C7,X-nél össze kell adni $04C7-et és $28-at.

$$\begin{array}{rcr} \$04C7 & = & 1223 \\ +\$\ \ 28 & = & \underline{\ \ 40} \\ \$04EF & = & 1263 \end{array}$$

Így a $04EF (1263) címre kerül az A regiszter tartalma, $2A. Ezt a címzési módot abszolút indexelt címzésnek nevezzük.
Most az X regiszterrel indexeltünk, az X regiszterben levő érték adódott hozzá a megadott abszolút címhez, és így keletkezett a valóságos cím. Ennél a címzési módnál nemcsak a programban megadott értéktől függ a cím, hanem az X regiszter pillanatnyi értékétől is. Indexelni persze az Y regiszterrel is lehet, de az A-val nem. A regiszterek ebben is különböznek.
A következő utasítás az X regisztert csökkentő DEX. Hatására az X regiszter tartalma eggyel kevesebb, azaz $27 lesz.
Ezzel elérkeztünk egy feltételes vezérlésátadó utasításhoz, ami olyan, mint a Basicben az IF . . . THEN . . . utasításpár. A BNE $C004 azt jelenti, hogy: „Térj vissza $C004-re, ha a Z jelzőbit nem ég!" Amikor a főnök megkapja ezt az utasítást, előveszi a látcsövét, és ránéz a Z jelzőbitre. Ha a Z jelzőbit nem ég, akkor visszaállítja a PC-t $C004-re. Ha viszont a lámpa ég, nem csinál semmit. Mivel előzőleg a DEX utasítás hatására az X értéke $27 lett, ezért a Z jelzőbit most nem ég, így ennél az utasításnál a PC-t átállítja. Ezzel az utasítás végrehajtása véget ért. Amikor azonban a vezér hozzálátna a következő utasítás megkereséséhez, és ezért a PC-re pillant, ott $C004-et talál, így onnan veszi a következő utasítást, megint az STA-t. Megint az A-ból kell valamit a memóriába írnia, és a címet megint úgy kapja meg, ha a $C04C7-hez hozzáadja az X tartalmát. De most, hogy az X értéke eggyel csökkent, ez a kiszámolt cím $04EE lesz, így most a csillag kódja a szomszédos tárhelyre került. Mindez ugyanígy ismétlődik egészen addig, míg az X értéke egyre nem csökken. Addigra már 39 helyen áll a csillag kódja. Ekkor az A tartalma az indexelés következtében a $04C8 címre kerül, tehát megjelenik a csillag a 40. helyen is, majd a DEX utasítás után az X értéke nulla lesz, és kigyullad a Z jelzőbit. Így tehát a BNE utasítás hatására most semmi nem történik, az utasításszámláló (PC) nem áll vissza $C004-re, hanem marad $C00A, azaz a tárban soron következő cím. A következő végrehajtandó utasítás az RTS lesz, és a programfutás befejeződik. (A programrész tehát negyvenszer ismétlődik, majd amikor X értéke eléri a nullát, befejeződik a program.) Pusztán a tisztánlátás kedvéért álljon itt az a Basic program, ami ugyanezt a feladatot végzi el.

```
100 X=40
110 A=42
120 POK 1223+X,A
130 X=X-1
140 IF X<>0 THEN 120
150 END
```

Az STA $04C7,X tehát ugyanolyan hatású, mint a POKE 1223+X,A.
Vizsgáljuk meg még egyszer a BNE utasítást! A BNE utasítás akkor adja át a vezérlést a kívánt címre, ha a Z jelzőbit nem ég (egyébként folytatódik tovább a program). Van olyan utasítás is, amelyik akkor adja át a vezérlést, ha a Z jelzőbit ég. Ez az utasítás a BEQ. (Van még néhány, azokkal később fogunk találkozni.)

A feltételes vezérlésátadó utasítások nem abszolút címzésűek, csak kétbájtosak. Az utasítás kódja után csak egy szám van, és ezt úgy kell értelmezni, mint egy kettes komplemens kódban adott számot.

BNE utasításunk gépi kódú alakjában a $D0, $FA számsorozatból a $D0 a BNE kódja. A $FA kettes komplemens kódban -6. Magyarul a két szám, mint utasításkód, azt mond-

ja: „Ha a Z jelzőbit nem ég, akkor tekerd vissza a PC-t 6 fordulattal!" Ez a mi esetünkben azt jelentette, hogy a $C00A-hoz (a következő utasítás első bájtjához) képest kell 6-szor visszatekerni, vagyis pont $C004-re. Ebből pedig az is látszik, hogy feltételes vezérlésátadással a következő utasítás első bájtjához képest maximum 127 bájttal lehet előre, illetve maximum 128 bájttal lehet vissza lépni. Ezt a címzést relatív címzésnek nevezzük, mert a következő utasítás első bájtjához viszonyítva (tehát relatív módon) kell megadni, hogy mennyivel előbbre vagy mennyive hátrébb akarunk elágaztatni.
Most végre teleírhatjuk a képernyőt piros csillagokkal, gépi kódban. Az összehasonlítás kedvéért először futtassuk le a Basic programot:

```
100 FOR I=0 TO 999
110 POKE 55296+I,2
120 POKE 1024+I,42
130 NEXT I
```

A gépi kódú program:

```
 cím     kód         utasítás
 ---     ---         --------
C000   A2 00         LDX #$00
C002   A9 02         LDA #$02
C004   9D 00 D8      STA $D800,X
C007   9D 00 D9      STA $D900,X
C00A   9D 00 DA      STA $DA00,X
C00D   9D 00 DB      STA $DB00,X
C010   A9 2A         LDA #$2A
C012   9D 00 04      STA $0400,X
C015   9D 00 05      STA $0500,X
C018   9D 00 06      STA $0600,X
C01B   9D 00 07      STA $0700,X
C01E   E8            INX
C01F   D0 E1         BNE $C002
C021   60            RTS
```

Ez a program a kiszámolt 2003 helyett csak 14 utasításból áll, és 6005 memóriarekesz helyett csak 34 rekeszt foglal el! A Basic betöltő:

```
100 FOR I=0 TO 33
110 READ A
120 POKE 49152+I,A
130 NEXT I
140 DATA 162,0,169,2,157,0,216,157,0,217,157,0,218
150 DATA 157,0,219,169,42,157,0,4,157,0,5,157,0,6
160 DATA 157,0,7,232,208,225,96
```

Ha a betöltőt lefuttattuk, akkor már csak a gépi kódú programot kell elindítani. Gépeljük be: SYS 49152. Futtatáskor észrevehető a Basic és a gépi kódú program sebessége közötti különbség.
Gyakorló feladat a ciklusszervezésre: Töltsünk meg két különböző karakterrel felváltva előbb egy sort, majd az egész képernyőt!

ÖSSZEFOGLALÁS ÉS KIEGÉSZÍTÉS

Az alábbiakban összefoglaljuk, rendszerezzük és néhol kiegészítjük az eddig tanult utasításokat, címzésmódokat, jelzőbiteket.

LDA — Az A regiszterbe tölt be egy számot közvetlenül vagy valamelyik memóriarekeszből.
LDX — Az X regiszterbe tölt be egy számot közvetlenül vagy valamelyik memóriarekeszből.
LDY — Az Y regiszterbe tölt be egy számot közvetlenül vagy valamelyik memóriarekeszből.

STA — Az A regiszter tartalmát menti ki egy memóriacímre.
STX — Az X regiszter tartalmát menti ki egy memóriacímre.
STY — Az Y regiszter tartalmát menti ki egy memóriacímre.

Ezek mind valamely memóriacím és regiszter között valósítanak meg adatátvitelt. Vannak olyan utasítások is, amelyek két regiszter között adnak át adatot. Ezek:

TAX — Az A regiszter tartalmát másolja át az X regiszterbe.
TAY — Az A regiszter tartalmát másolja át az Y regiszterbe.
TSX — A veremmutató tartalmát (erről még később lesz szó, ez is egy rekesz) másolja át az X regiszterbe.
TXA — Az X regiszter tartalmát másolja át az A regiszterbe.
TXS — Az X regiszter tartalmát másolja át a veremmutatóba.
TYA — Az Y regiszter tartalmát másolja át az A regiszterbe.

Megismertük, milyen módon változtathatjuk az X és Y regiszterek tartalmát lépésenként:

INX — Megnöveli eggyel az X regiszter tartalmát. Ha előzőleg $FF volt, akkor $00 lesz.
DEX — Csökkenti eggyel az X regiszter tartalmát. Ha előzőleg $00 volt, akkor $FF lesz.
INY — Megnöveli eggyel az Y regiszter tartalmát. Ha előzőleg $FF volt, akkor $00 lesz.
DEY — Csökkenti eggyel az Y regiszter tartalmát. Ha előzőleg $00 volt, akkor $FF lesz.

Nemcsak az X és Y tartalmát lehet így változtatni, hanem bármely memóriarekeszét is.

Erre szolgálnak az alábbi utasítások:

INC — A címzéstől függően eggyel megnöveli valamely memóriacím értékét. Ha előzőleg $FF volt, akkor $00 lesz.
DEC — A címzéstől függően eggyel csökkenti valamely memóriacím értékét. Ha előzőleg $00 volt, akkor $FF lesz.

Megismerkedtünk a jelzőbitekkel, vagyis azokkal a lámpákkal, amelyek a különböző gombok nyomogatása következtében vagy kigyulladnak, vagy elalszanak. Ezek közül kettőnek a szerepét is tudjuk, a többiről még csak annyit tudunk, hogy vannak.

Tudjuk továbbá, hogy gépi kódban milyen a feltétlen vezérlésátadásra szolgáló utasítás.

JMP — Feltétlen vezérlésátadás a címzésben megadott memóriacímre.

Azt is tudjuk, hogy feltételesen elágaztatni csak a jelzőbitek állapotától függően lehet. Vigyáznunk kell arra, hogy feltételesen elágazni a következő utasítás első bájtjához képest csak max. 127 címmel előrébb, és max. 128 címmel hátrébb (vissza) lehet. Négy példát láttunk ilyen utasításra, két jelzőbittel kapcsolatban, aszerint, hogy akkor kell-e elágazást megvalósítani, ha égnek a lámpák, vagy éppen akkor, ha nem. Ugyanilyen utasítások segítségével még kettő, azaz összesen négy jelzőbit állapotától függően lehet elágaztatni,

A feltételes vezérlésátadásra szolgáló utasítások:

BCS – Elágazás, ha a C jelzőbit ég (egyébként folytatódik a program a következő utasítással).
BCC – Elágazás, ha a C jelzőbit nem ég.
BEQ – Elágazás, ha a Z jelzőbit ég.
BNE – Elágazás, ha a Z jelzőbit nem ég.
BMI – Elágazás, ha az N jelzőbit ég.
BPL – Elágazás, ha az N jelzőbit nem ég.
BVS – Elágazás, ha a V jelzőbit ég.
BVC – Elágazás, ha a V jelzőbit nem ég.

A jelzőbitek neve angolul flag, azaz zászló (gondoljunk a jelzőzászlókra). Sok szakkönyvben találkozhatunk a FLAG elnevezéssel.

Van néhány utasítás, amelyekkel mi is befolyásolhatjuk a jelzőbitek állapotát. Parancsot adhatunk egyes jelzőbitek kigyújtására és eloltására is:

SEC – Kigyújtja a C jelzőbitet.
SED – Kigyújtja a D jelzőbitet.
SEI – Kigyújtja az I jelzőbitet.
CLC – Eloltja a C jelzőbitet.
CLD – Eloltja a D jelzőbitet.
CLI – Eloltja az I jelzőbitet.
CLV – Eloltja a V jelzőbitet.

Végül be tudjuk fejezni gépi kódú programunkat, és erre két utasítást is ismerünk:

BRK – Program megszakítása. Hatására minden eltűnik a képernyőről, és megjelenik a READY felirat.
RTS – Visszatérés gépi kódú programból. Ez az utasítás nem törli a képernyőt.

Megtanultuk azt is, hogy mit jelent a „címzési mód" kifejezés, és megismerkedtünk néhány címzési móddal.
Burkolt címzés: Nincs hivatkozás memóriarekeszre. Az összes burkolt címzésű utasítás egybájtos, csak az utasítás kódjából áll.
Közvetlen címzés: Az utasításkód utáni bájt maga tartalmazza azt a számot, amelyet valamely regiszterbe be kell tölteni. Ennél a címzési módnál nem hivatkozunk memóriacímre.
Abszolút címzés: Az utasításkód utáni két bájt határozza meg a kívánt memóriarekesz címét.
Abszolút indexelt címzés: Két változata létezik. Az egyik esetben az X indexregiszterrel indexelünk, a másik esetben az Y-nal. Az utasításkódot követő két bájt megad egy abszolút címet, de ehhez még hozzá kell adni az indexregiszter tartalmát is, és így kapjuk meg a kívánt memóriarekesz címét.
Indirekt címzés: Csak elágaztatásnál használt. Az utasításkód után következő bájt kettes komplemens kódban jelzi, hogy a következő utasítás első bájtjához képest hány címmel előrébb vagy hátrább kell keresni a legközelebb végrehajtandó utasítást.

Tudjuk már azt is, hogy hogyan kell a képernyőn valamit megjeleníteni, és azt is, hogy hogyan kell egyszerű ciklusokat szervezni.
Az eddig megismert utasítások adatai az eddig megismert címzési módokban:

Utasítás	Címzési mód					
	burkolt	közvetlen	abszolút	abszolút, X	abszolút, Y	relatív
BCC						9Ø
BCS						BØ
BEQ						FØ
BMI						3Ø
BNE						DØ
BPL						1Ø
BRK	ØØ					
BVC						5Ø
BVS						7Ø
CLC	18					
CLD	D8					
CLI	58					
CLV	B8					
DEC			CE	DE		
DEX	CA					
DEY	88					
INC			EE	FE		
INX	E8					

Utasítás	Címzési mód					
	burkolt	közvetlen	abszolút	abszolút, X	abszolut, Y	relatív
INY	C8					
JMP			4C			
LDA		A9	AD	BD	B9	
LDX		A2	AE		BE	
LDY		AØ	AC	BC		
RTS	6Ø					
SEC	38					
SED	F8					
SEI	78					
STA			8D	9D	99	
STX			8E			
STY			8C			
TAX	AA					
TAY	A8					
TSX	BA					
TXA	8A					
TXS	9A					
TYA	98					

Ez a táblázat csak egy része annak, amely az összes utasítás adatait tartalmazza az összes lehetséges címzési módban. A függelékben levő teljes táblázat alapján (ld. 163. old.) észrevehetjük, hogy még közelről sem ismerünk minden utasítást és címzési módot.

ÖSSZEADÁS ÉS KIVONÁS

A számítógépek nemcsak adatok rakosgatására képesek, hanem számolásra is. Ez természetesen nem jelent fergeteges matematikai képességeket. Az ALU csupán összeadni és kivonni tud, valamint ismer néhány matematikai logikai műveletet, amelyekkel a szorzás is elvégezhető. Kezdjük a legegyszerűbbel, az összeadással. Mindenek előtt nézzük meg, hogy két szám összeadásakor az összeg hogyan viselkedik. Először adjunk össze egybájtos (azaz tizenhatos számrendszerben éppen kétjegyű) számokat!

$32	$0A	$88	$5D	$FF
+$22	+$0F	+$55	+$E3	+$FF
$54	$19	$DD	$140	$1FE

Látható, hogy egybájtos számok összeadásakor van olyan eset, amikor az összeg is egybájtos lesz, és az is könnyen előfordulhat, hogy nagyobb lesz, mint $FF. Azt viszont bátran kijelenthetjük, hogy hárombájtos soha nem lesz, mert az eredmény még akkor is csak kétbájtos, ha $FF-hez $FF-et adunk hozzá. Sőt, nemcsak hogy kétbájtos lesz az eredmény, de a felső bájt értéke csak nulla ($00) vagy egy ($01) lehet. Vagyis levonhatjuk a következtetést, hogy két nyolcbites szám összeadásának eredménye maximum kilenc bites lehet (annál több soha).

Az ALU mindig csak nyolcbites számokat adhat össze, így az eredmény maximum kilenc bit lehet. Az összeadásnak azonban van egy jellegzetes tulajdonsága. Csak az A regiszterhez lehet hozzáadni, és az eredmény is mindig az A regiszterben keletkezik. Összeadás után tehát az A regiszter eredeti tartalma elvész, mert átadja a helyét az összegnek. Az összeadásra szolgáló utasítás az ADC, amelynek kódja abszolút címzésben $6D. Nézzünk egy példaprogramot:

```
cím     kód         utasítás
---     ---         --------
C000  AD 50 C3      LDA $C350
C003  6D 51 C3      ADC $C351
C006  8D 52 C3      STA $C352
C009  60            RTS
```

A program betölti A-ba a $C350 memóriacím tartalmát, hozzáadja a $C351 cím tartalmát, és az eredményt kiküldi a $C352 címre. A program betöltője:

```
100 FOR I=0 TO 9
110 READ A
120 POKE 49152+I,A
130 NEXT I
140 DATA 173,80,195,109,81,195,141,82,195,96
```

A Basic betöltő lefuttatása után elkezdhetünk kísérletezni. Először mindig elhelyezünk két összeadandót, lefuttatjuk a programot, és megnézzük az eredményt:

```
POKE 50000, 15

READY.
POKE 50001, 200

READY.
SYS 49152

READY.
? PEEK (50002)
 215

READY.
■
```

vagyis 15+200=215. Továbbá:

```
POKE 50000, 200

READY.
POKE 50001, 60

READY.
SYS 49152

READY.
? PEEK (50002)
  4

READY.
■
```

vagyis 200+60=4. Ez így nem igaz, de mindjárt meg is találhatjuk rá a magyarázatot:

200	$C8
+ 60	+$3C
260	$104

Ebből nyilván csak a 04 maradt az A rekeszben, hiszen ott csak 1 bájt fér el, és ez lett a $C352 rekeszbe kiírva. Továbbá:

```
POKE 50000, 3

READY.
POKE 50001, 4

READY.
SYS 49152

READY.
? PEEK (50002)
 8

READY.
■
```

vagyis 3+4=8. Ezt nem így tanultuk az iskolában! Futtassuk le még egyszer:

```
POKE 50000, 3

READY.
POKE 50001, 4

READY.
SYS 49152

READY.
? PEEK (50002)
  7

READY.
■
```

vagyis 3+4=7. Mi okozhatta ezt a furcsaságot? Nyilván nem az, hogy a gép ugyanazt a programot kétféleképpen is végrehajthatná — ez képtelenség. A gép, amikor összead, az eredmény alsó nyolc bitjét teszi A-ba. Ha a kilencedik bit nulla, akkor nincs is gond. De ha a kilencedik bit egy, akkor bizony van, és ezen úgy teszi magát túl, hogy a kilencedik bittől függően meggyújtja a C jelzőbitet. Ez történt akkor, amikor összeadott 200-at és 60-at. Ez összesen 260 ($104). Ebből az A-ban maradt $04, és kigyulladt a C. De az a tulajdonsága is megvan, hogy nemcsak a kívánt számokat adja össze, hanem hozzáadja a C értékét is. Ha a C jelzőbit nem ég az összeadás előtt, akkor egyszerűen csak összeadja a két számot. Ha viszont ég, akkor ehhez az eredményhez hozzáad még egyet. Mivel az előző összeadás eredményeként égett a C, és azt még senki sem törölte (nem oltotta el), ezért a következő összeadásnál az eredményhez még egyet hozzáadott, így lett 3+4(+1)=8.
Összeadáskor tehát összeadja a két számot, s ha ég a C jelzőbit, akkor hozzáad még egyet, és így keletkezik az eredmény. Az eredmény nyolc alsó bitje bekerül az A regiszterbe. Ha a kilencedik bit egyes, akkor a C kigyullad, egyébként pedig elalszik. Éppen ezért, ha azt akarjuk, hogy az összeadás helyes eredményt adjon, akkor előtte a C jelzőbitet (CLC utasítással) törölni kell, hogy a két számhoz ne adódjon hozzá még egy. Ezt a gép magától nem tenné meg. Összeadás után a C jelzőbit állapotából lehet következtetni arra, hogy az összeadás eredménye nagyobb lett-e 255-nél.
Alakítsuk át a programunkat! Az eredményt továbbra is tároljuk a $C352 címen, de ezen kívül tároljuk a C jelzőbit értékét is — a $C353 címen. Ha az összeadás következtében nem gyulladt ki, akkor legyen $C353-on nulla, ha pedig kigyulladt, akkor $01! Ezzel a helyes eredményt is ki tudjuk majd olvasni. Vennünk kell a $C353 cím tartalmát, meg kell szoroznunk 256-tal (egy kilencbites szám legfelső bitje ugyanis pontosan 256-ot ér, ha ég a lámpa), majd hozzá kell adnunk a $C352 cím tartalmát.

```
 cím     kód        utasítás
 ---     ---        --------
C000   AD 50 C3     LDA $C350
C003   18           CLC
C004   6D 51 C3     ADC $C351
C007   8D 52 C3     STA $C352
C00A   B0 06        BCS $C012
C00C   A9 00        LDA #$00
C00E   8D 53 C3     STA $C353
C011   60           RTS
C012   A9 01        LDA #$01
C014   8D 53 C3     STA $C353
C017   60           RTS
```

A betöltő:

```
100 FOR I=0 TO 23
110 READ A
120 POKE 49152+I,A
130 NEXT I
140 DATA 173,80,195,24,109,81,195,141,82,195,176,6,169,0
150 DATA 141,83,195,96,169,1,141,83,195,96
```

A betöltő lefuttatása után kipróbálhatjuk tökéletesített összeadó programunkat:

```
POKE 50000, 15

READY.
POKE 50001, 200

READY.
SYS 49152

READY.
? PEEK (50003*256+PEEK (50002)
 215
READY.
■
```

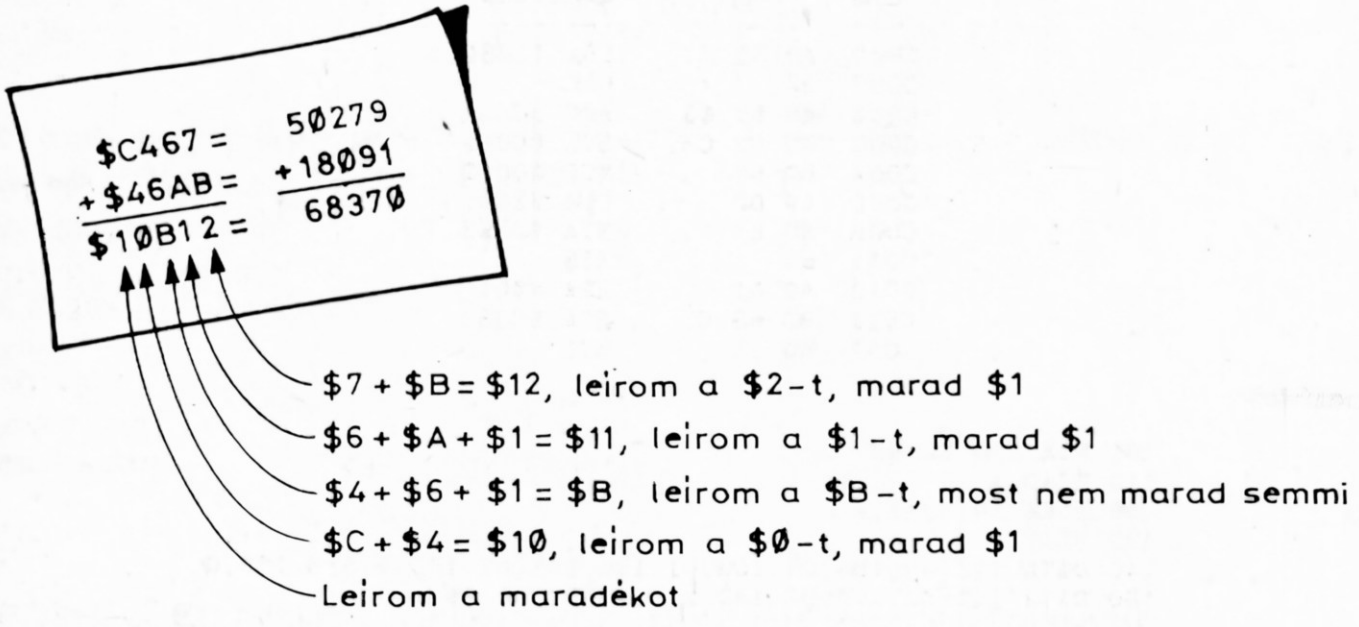
Ez az előbb is jó volt.

```
POKE 50000,200
READY.
POKE 50001,60
READY.
SYS 49152
READY.
? PEEK (50003)*256+PEEK (50002)
 260
READY.
■
```

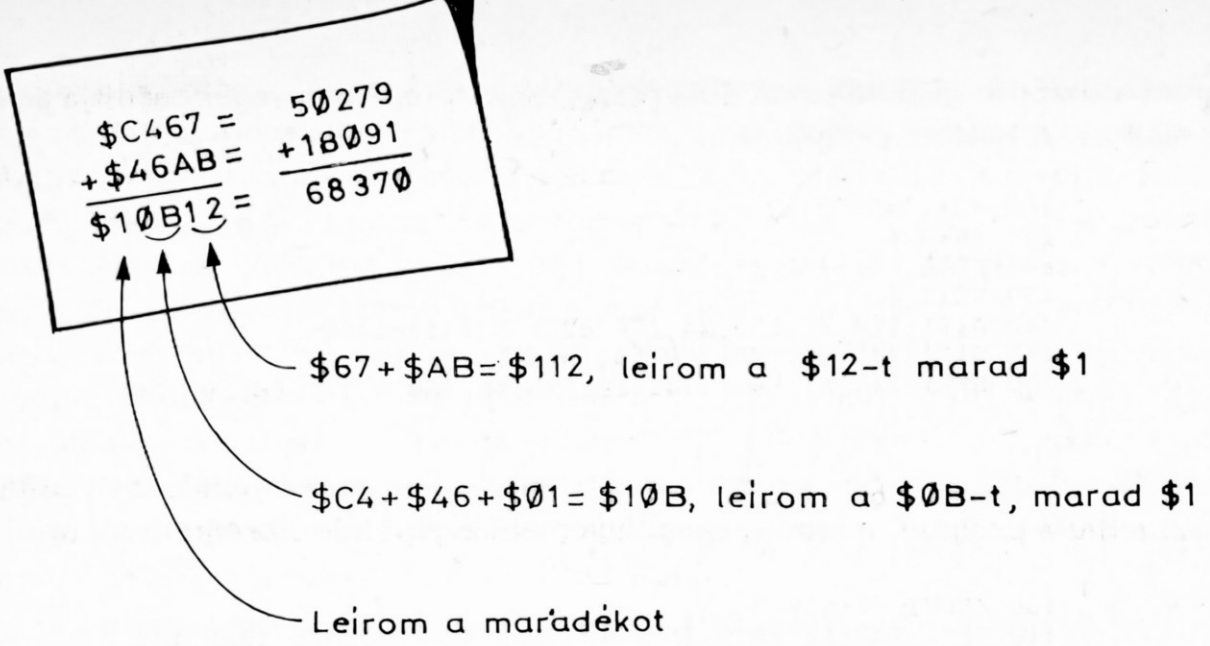
Lám, már ez is jó eredményt ad.

```
POKE 50000,3
READY.
POKE 50001,4
READY.
SYS 49152
READY.
? PEEK (50003)*256+PEEK (50002)
 7
READY
■
```

Ez is jól működik most.
A C jelzőbitre vigyázni kell, mert könnyen zavart okozhat! Egyszerű összeadások előtt mindig törölni kell egy CLC utasítással, majd az összeadás után meg kell vizsgálnunk ezt a jelzőbitet. Csak így lehetünk biztosak abban, hogy az eredmény helyes.



A kétbájtos számok tizenhatos számrendszerben négyjegyűek. Ha tizenhatos számrendszerben adjuk őket össze, akkor jobbról balra haladva, számjegyenként elvégezhető az összeadás. A probléma rövidebben is megoldható, összeadhatjuk a számokat bájtonként is. Ha keletkezett átvitel az alsó bájtok összeadásakor, akkor azt még hozzá kell adni a felső bájtok összegéhez. Nekünk ez már nem okozhat problémát, mivel az alsó bájtok összeadása után a C jelzőbit tartalmazza az átvitelt. A felső bájtokat tehát nyugodtan összeadhatjuk, hiszen ha előzőleg volt átvitel, akkor a C jelzőbit ég, így mindenképpen hozzáadódik



még egy. Ha pedig előzőleg nem keletkezett átvitel, akkor nem is kell hozzáadni semmit, és mert ekkor a C jelzőbit nem ég, nem is fog hozzáadni semmit. Nekünk csupán az alsó bájtok összeadása előtt kell törölnünk C-t, és utána összeadhatjuk először az alsó, majd a felső bájtokat, anélkül, hogy közben az átvitellel kellene foglalkoznunk. Végül pedig a C állapotától függően egy harmadik, legfelső bájtba nullát vagy egyet írunk. Ha a C nem ég, akkor az eredmény is kétbájtos, és ekkor a legfelső bájtba nulla kerül. Ha a C ég, akkor az eredmény hárombájtos (17-bites), így a legfelső, harmadik bájtba egyet írunk. Azt szoktuk mondani, hogy a C jelzőbit értéke 1, ha ég, és 0, ha nem ég. Összeadáskor a C értéke mindig hozzáadódik az összeadandókhoz. Az elmondottak ugyanígy érvényesek három-, négy-, öt- és akárhány bájtos számok összeadására, így voltaképpen akármilyen nagy számokat kezelhetünk, nem vagyunk kötve egybájtos számokhoz. Következő példaprogramunk kétbájtos számokat ad össze. Ehhez tartozik egy Basic program, amely az adatbevitelt és az eredmény kiírását végzi, de magát az összeadást a gépi kódú program meghívásával intézi el. A két összeadandót az alábbi címeken tároljuk:

$C350 (50000) — A alsó bájtja
$C351 (50001) — A felső bájtja
$C352 (50002) — B alsó bájtja
$C353 (50003) — B felső bájtja
$C354 (50004) — A+B alsó bájtja
$C355 (50005) — A+B felső bájtja
$C356 (50006) — A+B legfelső bájtja

A gépi kódú program:

```
 cím     kód          utasítás
 ---     ---          --------
C000  AD 50 C3     LDA $C350
C003  18           CLC
C004  6D 52 C3     ADC $C352
C007  8D 54 C3     STA $C354
C00A  AD 51 C3     LDA $C351
C00D  6D 53 C3     ADC $C353
C010  8D 55 C3     STA $C355
C013  90 06        BCC $C01B
C015  A9 01        LDA #$01
C017  8D 56 C3     STA $C356
C01A  60           RTS
C01B  A9 00        LDA #$00
C01D  8D 56 C3     STA $C356
C020  60           RTS
```

A program először az alsó bájtokat adja össze, majd a felsőket, végül beállítja az eredmény legfelső bájtját. A betöltő program:

```
100 FOR I=0 TO 32
110 READ A
120 POKE 49152+I,A
130 NEXT I
140 DATA 173,80,195,24,109,82,195,141,84,195
150 DATA 173,81,195,109,83,195,141,85,195
160 DATA 144,6,169,1,141,86,195,96,169,0,141,86,195,96
```

Futtassuk le a betöltőt, hogy programunk bekerüljön a memóriába! Ezek után begépelhetjük azt a Basic programot, amely meghívja majd a gépi kódú programunkat.

```
100 INPUT "A=";A
110 IF A=INT(A) THEN IF A>=0 THEN IF A<=65535 THEN 130
120 GOTO 100
130 INPUT "B=";B
140 IF B=INT(B) THEN IF B>=0 THEN IF B<=65535 THEN 160
150 GOTO 130
160 AF=INT(A/256) : AA=A-(256*AF)
170 BF=INT(B/256) : BA=B-(256*BF)
180 POKE 50000,AA
190 POKE 50001,AF
200 POKE 50002,BA
210 POKE 50003,BF
220 SYS 49152
230 ER=PEEK(50004)+(256*PEEK(50005))+(65536*PEEK(50006))
240 PRINT "A+B=";ER
250 PRINT
260 RUN
```

A program bekéri a két összeadandó egész számot, melyek sem negatívak, sem 65535-nél nagyobbak nem lehetnek. A számok felső és alsó bájtokra bontva tárolódnak a memóriában. Ekkor meghívja az összeadó gépi kódú programot, majd az eredményt kijelzi a képernyőre. Futtassuk le és próbáljuk ki!
Látható, hogy nem kell mindig az egész programot gépi kódban írni, lehet azt is csinálni, hogy csak bizonyos részeket írunk gépi kódban. Ez most nem volt túl szerencsés, mert egyszerűen összeadhattuk volna a két számot Basicben is. De komolyabb problémáknál, ahol fontos egyes programrészek gyors futása, bizony sokszor használt módszer.
Nézzünk most négy példát kivonásra:

$A4	$8B	$23	$AA
–$43	–$7C	–$54	–$FF
$61	$0F	–$31	–$55

Az utolsó két példa eredménye negatív lett. Hogy ezt elkerüljük, ismételjük meg a kivonásokat, de úgy, hogy hozzáadunk $100-t a kisebbítendőhöz!

$1A4	$18B	$123	$1AA
–$ 43	–$ 7C	–$ 54	–$ FF
$161	$10F	$ CF	$ AB

Most már nincs probléma. Ezzel a módszerrel a kivonás eredménye mindig pozitív. Egy kilencbites számból kivonva egy nyolcbites számot, az eredmény mindig pozitív lesz.
Számítógépünk is tud kivonni. Természetesen csak az A regiszterből, a kivonás eredménye is mindig oda kerül. Kivonáskor az előjelproblémát az előbb leírt módon oldja meg. Mielőtt kivonná A-ból a kivonandót, először hozzáad 256-ot, így voltaképpen mindig egy kilencbites, 255-nél nagyobb számból von ki egy nyolcbites számot. Az eredmény tehát mindig pozitív lesz (mivel mindig a nagyobb számból vonja ki a kisebbet). Mivel azonban az összeadásnál is hozzáadta az összeadandókhoz a C értékét, most kivonja, de nem a C-t, hanem annak a negáltját. Ha tehát a kivonás előtt C értéke 0 volt, akkor a végén levon még egyet, ha pedig 1 volt, akkor nullát, vagyis semmit. Erre is a több-bájtos számok kivonása miatt van szükség. Ez a trükk azonban csak akkor működik helyesen, ha kivonáskor nem a C értékét, hanem annak negáltját vonja ki.
Az eredmény lehet kilenc bites, de ebből csak az alsó nyolc bit kerül az A-ba, a legfelső bit a kivonás után a C-be kerül. Összefoglalva tehát:
Kivonáskor először hozzáad az A-hoz $100-t, majd levonja belőle a kivonandót és a C negáltját. Az eredmény alsó nyolc bitje visszakerül az A-ba, a legfelső pedig bekerül a C-be. Kivonáskor a helyes eredmény érdekében ezért a C jelzőbitet a művelet előtt nem kioltani, hanem meggyújtani kell!
A kivonásra szolgáló utasítás az SBC, amelynek hatására az A-ból a részletezett módon levonja a cím által meghatározott memóriarekesz tartalmát.
Nézzünk egy egyszerű programot, amely kivonja az 50000. rekeszből az 50001. rekesz tartalmát, és az eredményt az 50002. rekeszbe teszi!

```
 cím     kód         utasítás
 ---     ---         --------
C000   AD 50 C3     LDA $C350
C003   38           SEC
C004   ED 51 C3     SBC $C351
C007   8D 52 C3     STA $C352
C00A   60           RTS
```

A program betöltője:

```
100 FOR I=0 TO 10
110 READ A
120 POKE 49152+I,A
130 NEXT I
140 DATA 173,80,195,56,237,81,195,141,82,195,96
```

Futtassuk le a betöltőt, és utána próbálkozzunk egy kicsit!

```
POKE 50000,100
READY.
POKE 50001,10
READY.
SYS 49152
READY.
?PEEK (50002)
 90
READY.
■
```

Nézzük meg, mi történt! Tudjuk, hogy 100=$64, és hogy 10=$0A.

1) $64+$100=$164
2) $164–$0A=$15A
3) $15A–$00=$15A (A SEC miatt C=1, ennek negáltja nulla.)
4) C=1, az A regiszter=$5A

Mivel $5A=90, és a PEEK is ezt találta, a program jól működött. Nézzünk egy újabb példát:

```
POKE 50000, 10
READY.
POKE 50001. 100
READY.
SYS 49152
READY.
? PEEK (50002)
 166
READY.
? 166–256
–90
READY.
```

A folyamat:

1) $0A+$100=10A
2) $10A–$64=$A6
3) $A6–$00=A6 (C=1, a negáltja nulla)
4) C=0, A regiszter=$A6

Mivel $A6=166, most is jól működött a program.
Ha még tovább próbálkozunk, akkor észrevehetjük, hogy minden olyan esetben, amikor nagyobb számból vonunk ki kisebbet, a PEEK a helyes eredményt adja vissza. Ha pedig kisebb számból vonunk ki nagyobbat, akkor az eredményből 256-ot levonva kapjuk meg a helyes végeredményt.
A kivonás is megoldható a több-bájtos számokkal, csak itt a legalsó bájtok kivonása előtt nem törölni, hanem kigyújtani kell a C jelzőbitet. Nézzünk most egy példát olyan gépi kódú programra, mely két tizenhatbites számot von ki egymásból. Ezt a programot is egy Basic programból hívjuk meg. Itt azonban, ha kisebb számból vonunk ki nagyobbat, akkor az eredmény helytelen, abból 65536-ot kivonva kaphatjuk csak meg a helyes végeredményt. A gépi kódú program:

```
 cím    kód        utasítás
 ---    ---        --------
C000  AD 50 C3    LDA $C350
C003  38          SEC
C004  ED 52 C3    SBC $C352
C007  8D 54 C3    STA $C354
C00A  AD 51 C3    LDA $C351
C00D  ED 53 C3    SBC $C353
C010  8D 55 C3    STA $C355
C013  90 06       BCC $C01B
C015  A9 01       LDA #$01
C017  8D 56 C3    STA $C356
```

```
C01A  60          RTS
C01B  A9 00       LDA #$00
C01D  8D 56 C3    STA $C356
C020  60          RTS
```

A betöltő:

```
100 FOR I=0 TO 32
 110 READ A
120 POKE 49152+I,A
130 NEXT I
140 DATA 173,80,195,56,237,82,195,141,84,195
150 DATA 173,81,195,237,83,195,141,85,195
160 DATA 144,6,169,1,141,86,195,96,169,0,141,86,195,96
```

A betöltő lefuttatása után a programot az alábbi Basic programmal ellenőrizhetjük:

```
100 INPUT "A=";A
110 IF A=INT(A) THEN IF A>=0 THEN IF A<=65535 THEN 130
120 GOTO 100
130 INPUT "B=";B
140 IF B=INT(B) THEN IF B>=0 THEN IF B<=65535 THEN 160
150 GOTO 130
160 AF=INT(A/256) : AA=A-(256*AF)
170 BF=INT(B/256) : BA=B-(256*BF)
180 POKE 50000,AA
190 POKE 50001,AF
200 POKE 50002,BA
210 POKE 50003,BF
220 SYS 49152
230 ER=PEEK(50004)+(256*PEEK(50005))
240 PRINT "EREDMENY=";ER
250 IF A<B THEN PRINT "VALODI EREDMENY=";ER-65536
260 PRINT
270 RUN
```

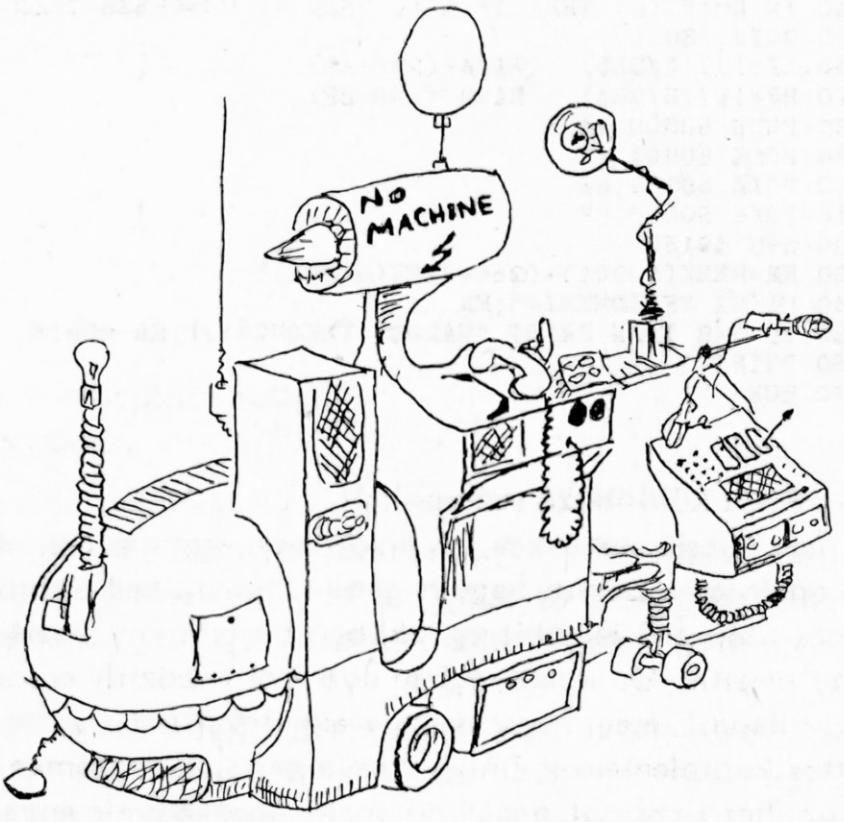
Próbáljuk ki a programot különböző számokkal!
Kivonásnál tehát egy kicsit nehézkes az eredmény értelmezése. Azt meg tudjuk állapítani, hogy az eredménynek pozitív vagy negatív számnak kell-e lennie, pusztán a C jelzőbitből. Ha a kivonás után a C jelzőbit ég, akkor az eredmény pozitív, ha viszont nem ég, akkor az eredmény negatív. De a valóságban ilyenkor is pozitív eredményt kapunk. A helyes eredményt úgy kapjuk meg, hogy negatív előjelet írunk a szám kettes komplemense elé. Egy szám kettes komplemensét (most nem a kettes komplemens kódról van szó) úgy kapjuk meg, ha az illető számot negáljuk, majd hozzáadunk egyet. Nézzük a korábbi példánkat!

166 = $A6 = b 10100110
Ennek negáltja: 89 = $59 = b 01011001
A negált +1 = 90 = $5A = b 01011010

S ha ezt vesszük negatív előjellel, akkor –90-et kapunk, ez valóban helyes eredmény. Ahhoz tehát, hogy a kivonást is mindig korrektül elvégezhessük, pusztán negálni kell tudni. A negálás ellentettképzést jelent, vagyis a negáltban minden olyan bit, ami az eredetiben nulla volt, egy lesz, és fordítva. A negáláshoz már csak a matematikai logika alapműveleteit és az azokat megvalósító utasításokat kell ismerni.

A MATEMATIKAI LOGIKÁRÓL

Hosszú évek kísérletező és tervező munkája eredményeként előttünk áll egy gép. Ha azt mondjuk neki, IGEN, arra azt válaszolja, hogy NEM, ha pedig közöljük a géppel, hogy NEM, akkor perceken belül rávágja, hogy IGEN, vagyis következetesen ellentmond alko-

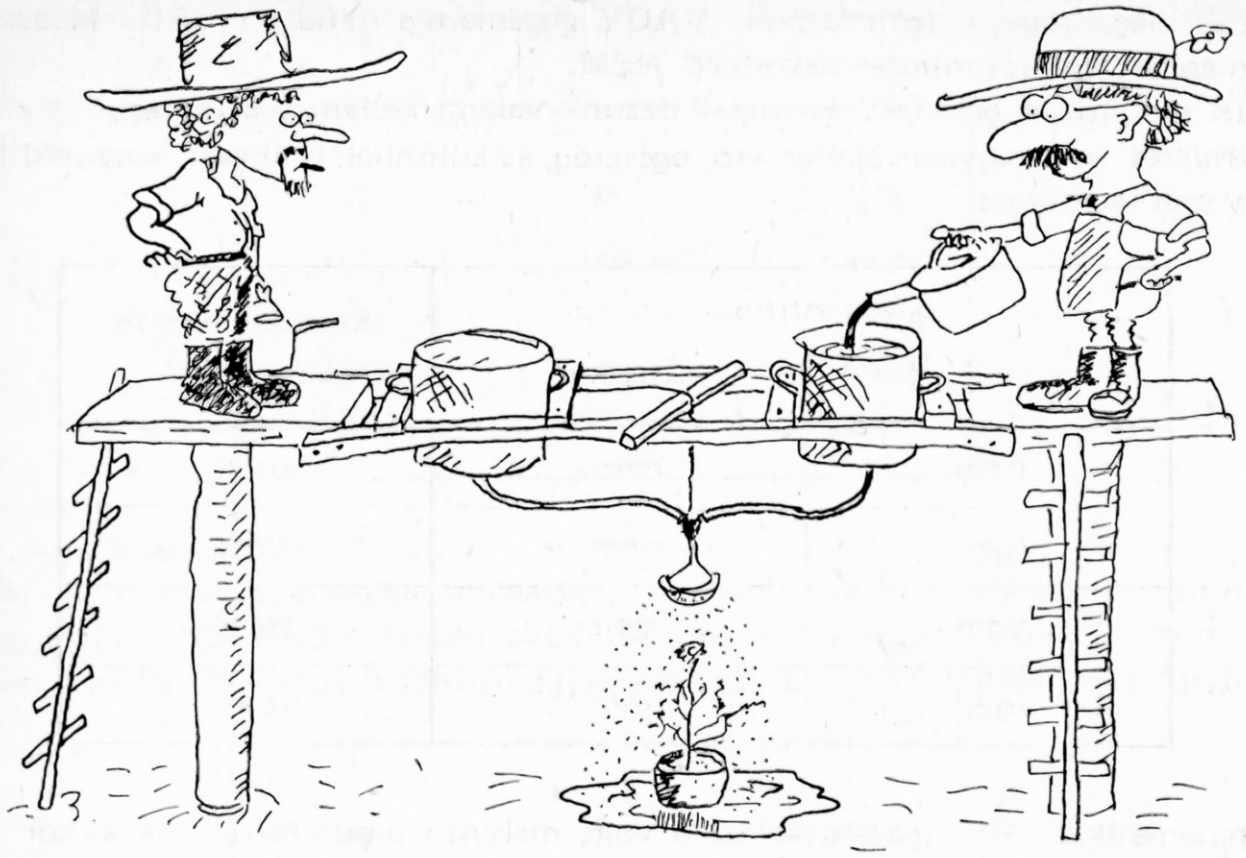


tójának. Ez csak abban az esetben engedhető meg, ha az illető gép egy tagadógép. A művelet pedig, amit a gép végez: tagadás.

IGEN tagadása = NEM
NEM tagadása = IGEN

Szerkesszünk egy másik gépet, melynek működése jóval egyszerűbb, mint a tagadógépé, mivel nem igényel elektronikus alkatrészeket!

A gép korszakalkotó jelentősége abban áll, hogy egyszerre két kertész számára teszi lehetővé az öntözést. Milyen esetek merülhetnek fel?



1. Senki nem önt sehová semmit, így a növény nem kap vizet.
2. Az egyik kertész vizet önt a tölcsérbe, így a virág kap vizet.
3. A másik kertész önt vizet a másik tölcsérbe, így is kap a virág vizet.
4. Mind a két kertész egyszerre önt a tölcsérekbe vizet, és a virág így sem hal szomjan.

Önt vizet a tölcsérbe:		Kap vizet a virág ?
az 1. számú kertész?	a 2. számú kertész?	
nem	nem	nem
igen	nem	igen
nem	igen	igen
igen	igen	igen

Ez a gép a matematikai VAGY műveletet valósítja meg. A virág akkor kap vizet, ha vagy az egyik kertész, vagy a másik, vagy mindkettő egyszerre önt vizet neki.

A VAGY művelet tehát:

IGEN vagy IGEN = IGEN
IGEN vagy NEM = IGEN
NEM vagy IGEN = IGEN
NEM vagy NEM = NEM

Jegyezzük meg, hogy a matematikai VAGY eredménye mindig IGEN, kivéve azt az egyetlen esetet, amikor minden összetevő NEM.
Felejtsük el most a gépeket, és foglalkozzunk valami kellemesebbel, együnk meg egy vajas kenyeret, mert ugye a vaj élet, erő, egészség, és különben is éhesek vagyunk! Ismételten négy eset lehetséges:

Van otthon		Ehetünk vajas kenyeret?
vaj?	kenyér?	
nem	nem	nem
igen	nem	nem
nem	igen	nem
igen	igen	igen

Ez a matematikai ÉS igazságtáblázata volt, melynek eredménye csak akkor IGEN, ha minden összetevő IGEN. Akkor tudunk csak vajas kenyeret kenni, ha van vaj is ÉS van kenyér is.
Helyettesítsük a szavakat számokkal! IGEN = 1 NEM = 0
A tagadás igazságtáblázata:

A	¬A
nem = Ø	igen = 1
igen = 1	nem = Ø

¬A = A ellentéte (angolul ezt úgy hívják, hogy NOT).

A matematikai VAGY művelet:

A	B	AVB
nem = Ø	nem = Ø	nem = Ø
nem = Ø	igen = 1	igen = 1
igen = 1	nem = Ø	igen = 1
igen = 1	igen = 1	igen = 1

AVB = A vagy B (angolul OR)

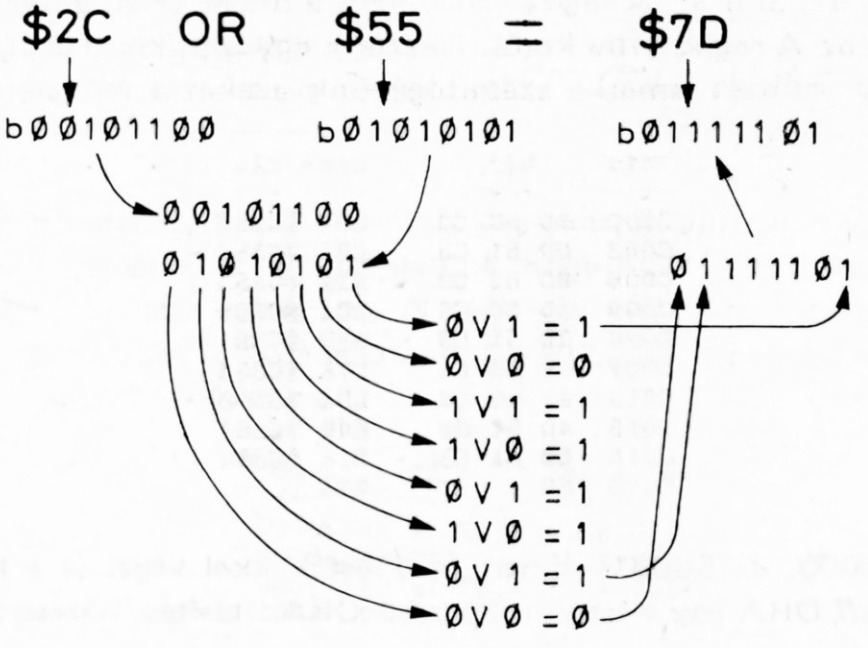
A matematikai ÉS művelet:

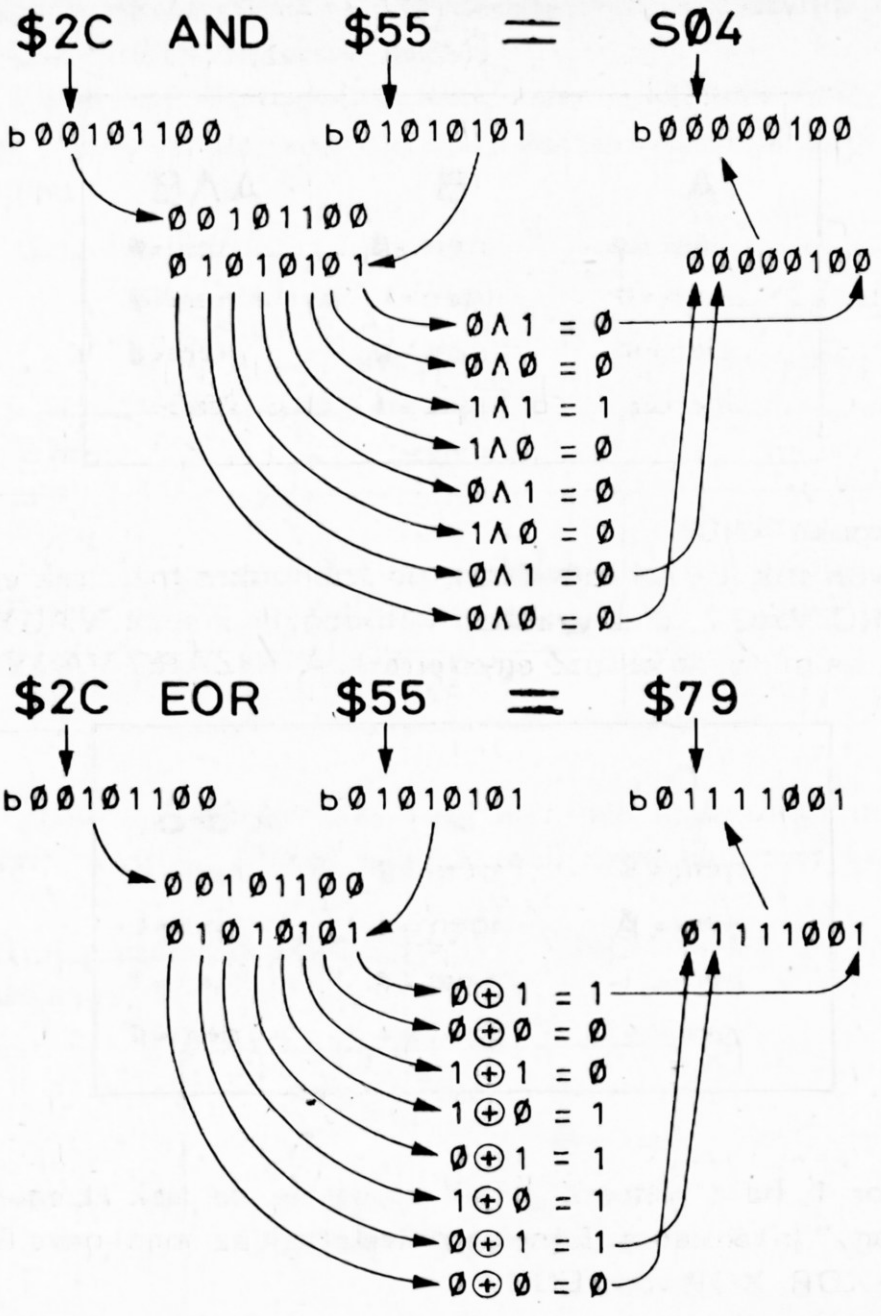
A	B	A∧B
nem = Ø	nem = Ø	nem = Ø
nem = Ø	igen = 1	nem = Ø
igen = 1	nem = Ø	nem = Ø
igen = 1	igen = 1	igen = 1

A∧B = A és B (angolul AND).
Ismerünk még nagyon sok logikai műveletet, de számunkra már csak egy fontos maradt, mégpedig a KIZÁRÓ VAGY, amely abban különbözik a sima VAGY-tól, hogy abban az esetben is nulla, ha mindkét változó egyszerre 1. A KIZÁRÓ VAGY igazságtáblázata:

A	B	A ⊕ B
nem = Ø	nem = Ø	nem = Ø
nem = Ø	igen = 1	igen = 1
igen = 1	nem = Ø	igen = 1
igen = 1	igen = 1	nem = Ø

Ez tehát csak akkor 1, ha a változók közül az egyik, de csak az egyik 1. A ⊕ B = A vagy B, de itt a „vagy" kizárólagos. Ennek a műveletnek az angol neve EXCLUDING OR, aminek rövidítése EXOR, XOR vagy EOR.
Vegyünk két nyolcbites számot, és végezzük el az egyes műveleteket! Legyen a két szám $2C = b00101100 és $55 = b01010101.





Ezt a három műveletet a számítógépünk is el tudja végezni, mégpedig úgy, hogy az egyik szám mindig az, ami az A regiszterben van, a másik szám a címzési módtól függ, az eredmény pedig az A regiszterbe kerül. Nézzünk egy gépi kódú programot, mellyel ellenőrizhetjük, hogy valóban ismeri-e számítógépünk ezeket a műveleteket. A program:

```
cím     kód         utasítás
---     ---         --------
C000   AD 50 C3     LDA $C350
C003   0D 51 C3     ORA $C351
C006   8D 52 C3     STA $C352
C009   AD 50 C3     LDA $C350
C00C   2D 51 C3     AND $C351
C00F   8D 53 C3     STA $C353
C012   AD 50 C3     LDA $C350
C015   4D 51 C3     EOR $C351
C018   8D 54 C3     STA $C354
C01B   60           RTS
```

Programunk az 50000. és 50001. címen levő értékekkel végzi el a három műveletet. A $C003 címen levő ORA egy abszolút címzésű ORA utasítás, hatására az A regiszterbe

kerül az A és a $C351 cím tartalmával végzett vagy művelet eredménye. Hasonlóan működik a $C00C címen levő abszolút címzésű AND utasítás is, amely azonban a matematikai ÉS műveletet végzi el. A $C015 címen levő abszolút címzésű EOR utasítás pedig a kizárólagos vagy műveletet valósítja meg. A programunkat betöltő Basic program:

```
100 FOR I=0 TO 27
110 READ A
120 POKE 49152+I,A
130 NEXT I
140 DATA 173,80,195,13,81,195,141,82,195
150 DATA 173,80,195,45,81,195,141,83,195
160 DATA 173,80,195,77,81,195,141,84,195,96
```

A betöltő lefuttatása után az alábbi Basic programmal ellenőrizhetünk:

```
100 INPUT "A=";A
110 IF A=INT(A) THEN IF A>=0 THEN IF A<=255 THEN 130
120 GOTO 100
130 INPUT "B=";B
140 IF B=INT(B) THEN IF B>=0 THEN IF B<=255 THEN 160
150 GOTO 130
160 POKE 50000,A
170 POKE 50001,B
180 SYS 49152
190 PRINT "      A=";  : N=A : GOSUB 250
200 PRINT "      B=";  : N=B : GOSUB 250
210 PRINT "A OR  B=";  : N=PEEK(50002) : GOSUB 250
220 PRINT "A AND B=";  : N=PEEK(50003) : GOSUB 250
230 PRINT "A EOR B=";  : N=PEEK(50004) : GOSUB 250
240 PRINT : GOTO 100
250 FOR I=7 TO 0 STEP -1
260 IF N<2↑I THEN PRINT "0"; : GOTO 280
270 N=N-(2↑I) : PRINT "1";
280 NEXT I
290 PRINT : RETURN
```

A számítógép a NOT utasítást nem ismeri, de a rafinált programozónak ez nem okoz gondot, mert tudja, hogy: $\neg A = A \oplus 1$. A kétkedők meggyőzésére íme a bizonyíték:

A	¬A	A ⊕ 1
Ø	1	Ø ⊕ 1 = 1
1	Ø	1 ⊕ 1 = Ø

→ ¬A = A ⊕ 1

Ha tehát egy bájt minden bitjét az ellenkezőjére akarjuk állítani (vagyis ha negálni akarjuk), akkor azt az utasítást kell végrehajtanunk, hogy EOR #$FF! Az EOR #$FF közvetlen címzésű EOR utasítás, amely az A regiszter tartalmát „kizárólagosan vagyolja" $FF-el (minden egyes bitet az ellenkezőjére fordít), vagyis negálja az A regisztert. Nézzünk erre is egy ellenőrző programot:

```
cím    kód        utasítás
---    ---        --------
C000   AD 50 C3   LDA $C350
C003   49 FF      EOR #$FF
C005   8D 51 C3   STA $C351
C008   60         RTS
```

A betöltő:

```
100 FOR I=0 TO 8
110 READ A
120 POKE 49152+I,A
130 NEXT I
140 DATA 173,80,195,73,255,141,81,195,96
```

A betöltő futtatása után az alábbi Basic programmal ellenőrizhetünk:

```
100 INPUT "A=";A
110 IF A=INT(A) THEN IF A>=0 THEN IF A<=255 THEN 130
120 GOTO 100
130 POKE 50000,A
140 SYS 49152
150 PRINT "    A="; : N=A : GOSUB 180
160 PRINT "NOT A="; : N=PEEK(50001) : GOSUB 180
170 PRINT : GOTO 100
180 FOR I=7 TO 0 STEP -1
190 IF N<2↑I THEN PRINT "0"; : GOTO 210
200 N=N-(2↑I) : PRINT "1";
210 NEXT I
220 PRINT : RETURN
```

Hogy mire jók ezek az utasítások? Alapvetően arra, hogy bizonyos memóriacímek bizonyos bitjeit 1-re vagy 0-ra állítsuk. Erre pedig a nagyfelbontású BIT-térképes grafika kezelésénél van égető szükség, ami már elég összetett kérdés (megtalálható bármely Commodore–64 leírásban).

Tegyük fel, hogy nem tudjuk, mi van az illető memóriacímen, de mindenképpen azt akarjuk, hogy a 3. bitje* 1 legyen, a többi pedig ne változzon.

Rejtélybájt:

?	?	?	?	?	?	?	?

?	?	?	?	1	?	?	?

A dolog egyszerű: ORA #$08
Csupán azt kell belátni, hogy valami OR 0 = valami, és valami OR 1 = 1

A	B	A OR B
Ø	Ø	Ø OR Ø = Ø
1	Ø	1 OR Ø = 1
Ø	1	Ø OR 1 = 1
1	1	1 OR 1 = 1

→ A OR Ø = A
A OR 1 = 1

	?	?	?	?	?	?	?	?	– ismeretlen tartalmú bájt
OR	Ø	Ø	Ø	Ø	1	Ø	Ø	Ø	– $Ø8
	?	?	?	?	1	?	?	?	– a bájt 3. bitje 1-re változott, a többi változatlan maradt

*A biteket jobbról balra számozzuk, a számozás nullával kezdődik.

Az ORA utasítással tehát egy nyolcbites szám tetszőleges bitjét 1-re állíthatjuk úgy, hogy a többi bit ne változzék meg. Felmerül a kérdés: lehet-e egy bájt tetszőleges bitjét 0-ra állítanunk anélkül, hogy a többi bit megváltozna? A válasz igen, mégpedig az AND utasítással. Itt elég belátni, hogy valami AND 0 = 0, és valami AND 1 = valami. A változatosság kedvéért most az ismeretlen bájt 5. bitjét állítsuk nullára a többi bit megváltoztatása nélkül.. Ekkor a bűvös mondat az, hogy AND #$DF. Ugyanis:

A	B	A AND B
Ø	Ø	Ø AND Ø = Ø
1	Ø	1 AND Ø = Ø
Ø	1	Ø AND 1 = Ø
1	1	1 AND 1 = 1

→

A AND Ø = Ø
A AND 1 = A

	?	?	?	?	?	?	?	?	– ismeretlen tartalmú bájt
AND	1	1	Ø	1	1	1	1	1	– $DF
	?	?	Ø	?	?	?	?	?	– a bájt 5. bitje Ø-ra változott, a többi változatlan maradt

Most már bármely memóriacímen levő szám bármelyik bitjét képesek vagyunk 1-re vagy 0-ra állítani úgy, hogy a többi bit értéke közben ne változzon. (Gondoljuk végig a különböző eseteket, hogy mikor, milyen utasítással állítható 1-re vagy 0-ra egy bizonyos cím valamelyik bitje!) És most, mivel már tudunk negálni, nem okozhat gondot a kettes komplemens képzése. (Nem a kettes komplemens kódról beszélünk, hanem a kettes komplemens képzésről.) Ezzel a kivonást végző programunkat is tökéletesíthetjük. Emlékezzünk vissza, hogyan tároltuk a memóriában az adatokat:

$C350 (50000) — A szám alsó bájtja
$C351 (50001) — A szám felső bájtja
$C352 (50002) — B szám alsó bájtja
$C353 (50003) — B szám felső bájtja
$C354 (50004) — A—B alsó bájtja
$C355 (50005) — A—B felső bájtja
$C356 (50006) — C jelzőbit értéke a kivonás után. Ha ez 1, akkor jó az eredmény. Ha ez 0, akkor nagyobbat vontunk ki a kisebből, így képezni kell az eredmény kettes komplemensét, és azt negatív előjellel kell kijelezni.

Nézzük most a gépi kódú programot, közbeszúrt magyarázatokkal.

Először el kell végezni a kivonást:

```
 cím     kód          utasítás
 ---     ---          --------
C000   AD 50 C3      LDA $C350
C003   38           SEC
C004   ED 52 C3      SBC $C352
C007   8D 54 C3      STA $C354
C00A   AD 51 C3      LDA $C351
C00D   ED 53 C3      SBC $C353
C010   8D 55 C3      STA $C355
```

Ha a C jelzőbit ég, akkor az eredmény helyes, ágazzunk el $C033-ra.

```
 cím     kód          utasítás
 ---     ---          --------
C013   B0 1E         BCS $C033
```

Ha a C jelzőbit nem égett, akkor tároljunk nullát, jelezvén, hogy a C jelzőbit nulla.

```
 cím     kód          utasítás
 ---     ---          --------
C015   A9 00         LDA #$00
C017   8D 56 C3      STA $C356
```

Most pedig képezzük az eredmény kettes komplemensét! Először negáljuk, majd növeljük

```
 cím     kód          utasítás
 ---     ---          --------
C01A   AD 54 C3      LDA $C354
C01D   49 FF         EOR #$FF
C01F   8D 54 C3      STA $C354
C022   AD 55 C3      LDA $C355
C025   49 FF         EOR #$FF
C027   8D 55 C3      STA $C355
```

meg az egészet eggyel. Először az alsó bájtot, és ha az ettől nulla lett, akkor a felső bájtot is.

```
 cím     kód          utasítás
 ---     ---          --------
C02A   EE 54 C3      INC $C354
C02D   D0 03         BNE $C032
C02F   EE 55 C3      INC $C355
```

Vége a programnak:

```
 cím     kód          utasítás
 ---     ---          --------
C032   60            RTS
```

El kell még intéznünk azt az esetet, amikor égett a C jelzőbit, mert ebben az esetben ide ágaztattunk. Tároljunk egyet, és legyen vége a programnak!

```
 cím     kód          utasítás
 ---     ---          --------
C033   A9 01         LDA #$01
C035   8D 56 C3      STA $C356
C038   60            RTS
```

A program futása után tehát az eredmény helyes, az 50006. cím tartalma pedig jelzi, hogy milyen az előjele. Ha ez egy, akkor pozitív, ha pedig nulla, akkor negatív. A betöltő:

```
100 FOR I=0 TO 56
110 READ A
120 POKE 49152+I,A
130 NEXT I
```

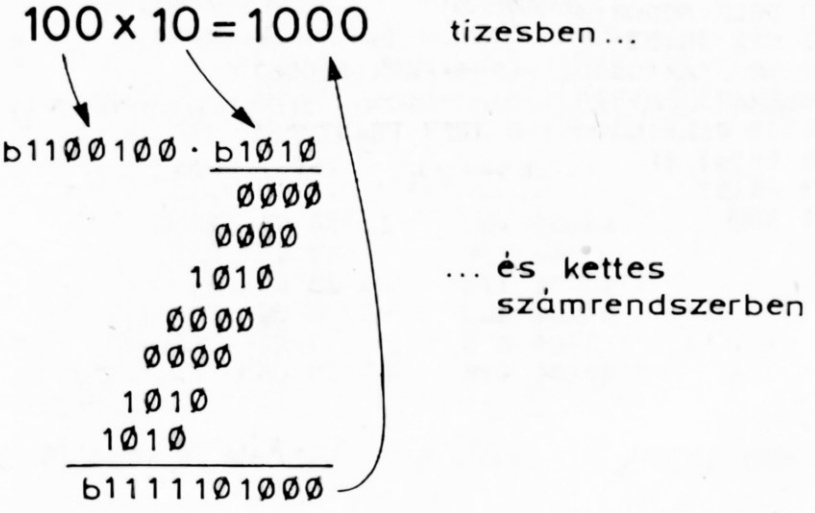
```
140 DATA 173,80,195,56,237,82,195,141,84,195
150 DATA 173,81,195,237,83,195,141,85,195
160 DATA 176,30,169,0,141,86,195,173,84,195,73,255
170 DATA 141,84,195,173,85,195,73,255,141,85,195
180 DATA 238,84,195,208,3,238,85,195,96,169,1
190 DATA 141,86,195,96
```

Ha lefuttatjuk a pontosan begépelt betöltőt, akkor a következő programmal ellenőrizhetjük is:

```
100 INPUT "A=";A
110 IF A=INT(A) THEN IF A>=0 THEN IF A<=65535 THEN 130
120 GOTO 100
130 INPUT "B=";B
140 IF B=INT(B) THEN IF B>=0 THEN IF B<=65535 THEN 160
150 GOTO 130
160 AF=INT(A/256) : AA=A-(256*AF)
170 BF=INT(B/256) : BA=B-(256*BF)
180 POKE 50000,AA
190 POKE 50001,AF
200 POKE 50002,BA
210 POKE 50003,BF
220 SYS 49152
230 ER=PEEK(50004)+(256*PEEK(50005))
240 PRINT "A-B=";
250 IF PEEK(50006)=0 THEN PRINT "-";
260 PRINT ER
270 PRINT
280 RUN
```

A SZORZÁS

Nézzük meg, hogyan lehet kettes számrendszerben két számot összeszorozni! Szorozzunk össze százat és tízet, vagyis b 1100100-át és b 1010-át.



Hasonlóan járjunk el, mint ahogyan tízes számrendszerben tennénk. Az egyik számot megszorozzuk a másik szám soron következő számjegyével. Mivel kettes számrendszerben ez a számjegy vagy nulla, vagy egy, nem kell szorozni, csak összeadni, ha az illető számjegy egyes. Ez pedig azt jelenti, hogy ha kettes számrendszerben akarunk szorozni, akkor elég ismerni az összeadást, valamint képesnek kell lennünk egy számot arrébbcsúsztatni, eltolni. Összeadni már tudunk, az eltolással pedig most ismerkedünk meg. Az eltolást angolul SHIFT-nek nevezik (sokszor szerepel így a szakkönyvekben). Ha egy nyolcbites számot akarunk eltolni egy bittel, akkor vagy ASL, vagy LSR utasítást alkalmazhatunk. Az ASL utasítással balra tudunk eltolni, az LSR utasítással pedig jobbra. Azt, hogy az utasítások jól működnek, az alábbi program bizonyítja:

```
cím     kód         utasítás
---     ---         --------
C000  0E 50 C3    ASL $C350
C003  4E 51 C3    LSR $C351
C006  60          RTS
```

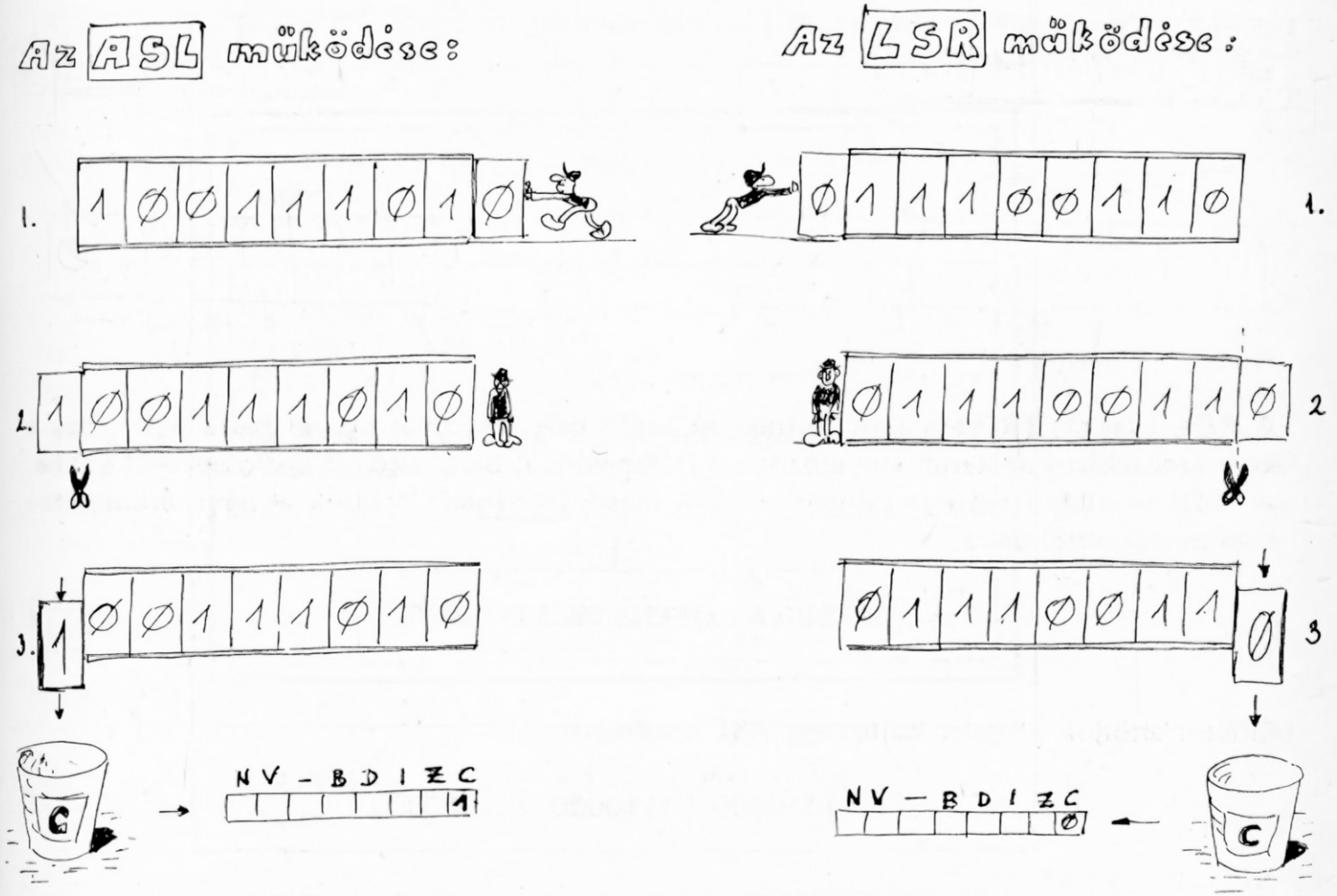
A betöltő:

```
100 FOR I=0 TO 6
110 READ A
120 POKE 49152+I,A
130 NEXT I
140 DATA 14,80,195,78,81,195,96
```

A betöltő lefuttatása után az alábbi Basic programmal ellenőrizhetünk:

```
100 INPUT "A=";A
110 IF A=INT(A) THEN IF A>=0 THEN IF A<=255 THEN 130
120 GOTO 100
130 POKE 50000,A
140 POKE 50001,A
150 PRINT "     A=";
160 N=A : GOSUB 230
170 SYS 49152
180 PRINT "ASL A=";
190 N=PEEK(50000) : GOSUB 230
200 PRINT "LSR A=";
210 N=PEEK(50001) : GOSUB 230
220 PRINT : RUN
230 FOR I=7 TO 0 STEP -1
240 IF N<2↑I THEN PRINT "0"; : GOTO 260
250 N=N-(2↑I) : PRINT "1";
260 NEXT I
270 PRINT : RETURN
```

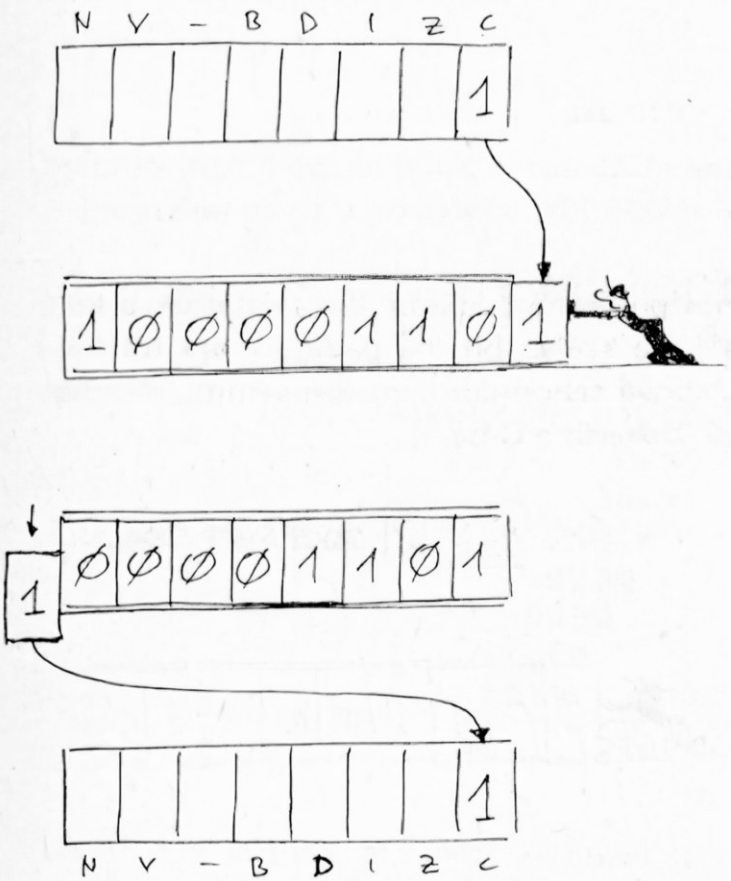
Ha többször, különböző számokon próbáljuk ki a programot, akkor észrevehetjük a következőt: ha balra toltuk el a számot, akkor a jobb szélső bit, ha pedig jobbra toltuk, akkor a bal szélső bit lett nulla. Az a bit tehát, ahova sehonnan nem jön semmi, mindig nulla lesz. Az pedig, amelyik nem tud hova menni, bekerül a C-be.



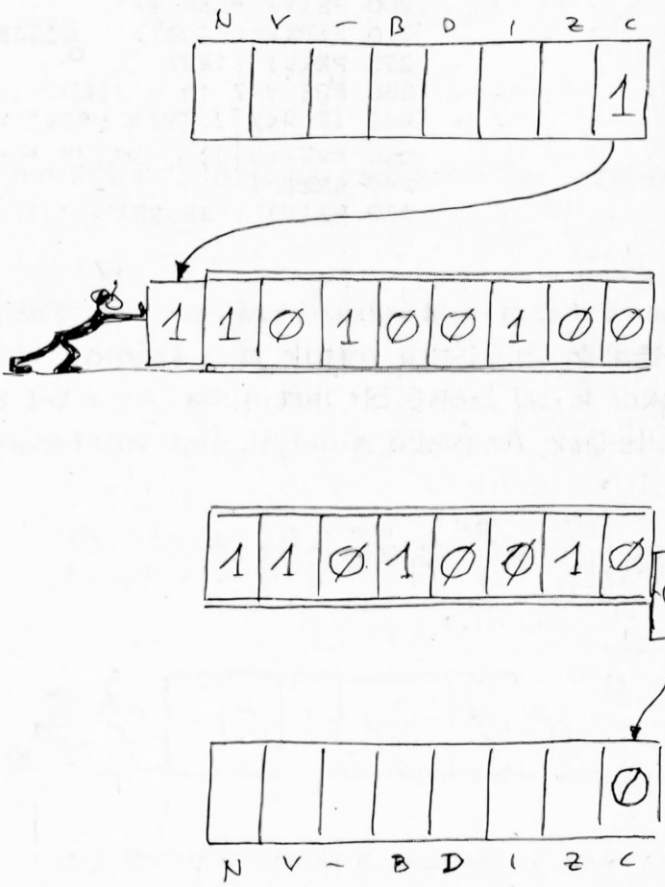
Nyolcbites számokat tudunk már jobbra vagy balra tologatni, de sokszor kell tizenhatbites (kétbájtos) vagy annál hosszabb számokat is. Erre a feladatra találták ki a rotáló uta-

sításokat, amelyek csak egy kicsit különböznek az eltoló utasításoktól. Ugyanúgy, ahogyan az eltolásnál, rotálásnál is egy bittel elmozdul a bájt tartalma, és a legszélső bit a C-be kerül. De a másik szélső bitbe nem feltétlenül kerül nulla, és nem is feltétlenül egy, hanem az, ami a C jelzőbitben előzőleg volt. Így tulajdonképpen ezek az utasítások egy körkörös mozgásnak is felfogható műveletet végeznek.

A ROL működése:



A ROR működése:



A ROL utasítás (ROtate Left) eltolja az illető bájt tartalmát eggyel balra úgy, hogy a jobb szélső bitbe az kerül, ami előzőleg a C-ben volt. A bal szélső bit pedig bekerül a C-be. A ROR utasítás ugyanezt csinálja, csak a másik irányban. Toljunk el egy hárombájtos számot egy bittel balra:

11110000 01110000 11110000

Először eltoljuk a legalsó bájtot egy ASL utasítással:

11110000 01110000 11100000

A bal szélső egyes bekerült a C-be. Most hajtsunk végre egy ROL utasítást a középső bájton:

11110000 11100001 11100000

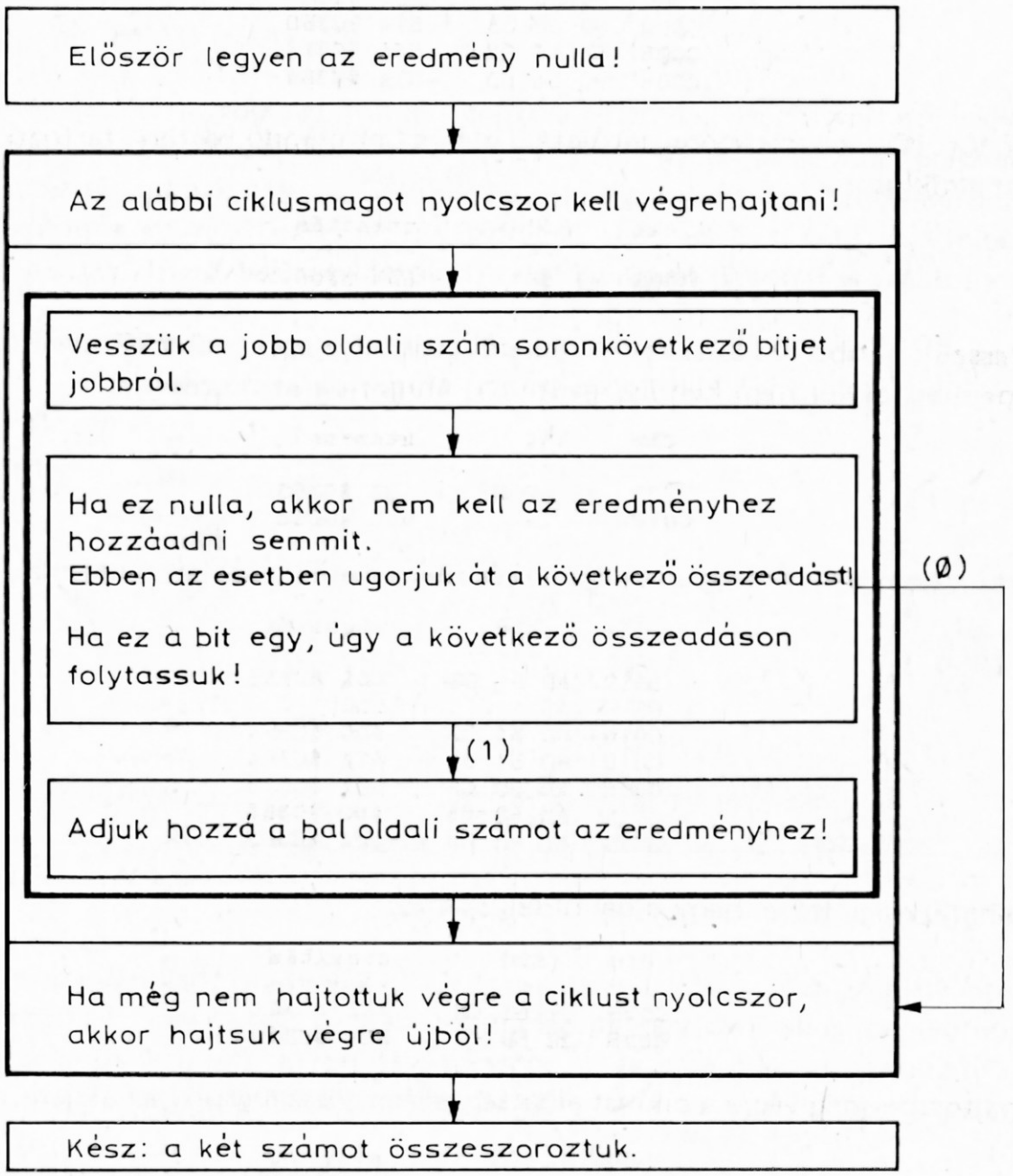
A bájt eltolódott balra, a jobb szélső bitbe bekerült az egyes, ami a C-ben volt, a bal szélső bit pedig bekerült a C-be. Hajtsunk most a bal szélső bájton is végre egy ROL műveletet:

11100000 11100001 11100000

A bájt eltolódott, a bal szélső bitbe bekerült a C-ben levő nulla, és a bal szélső egyes bekerült a C-be. Ezt a C-ben levő egyest írjuk a szám elé!

1 11100000 11100001 11100000

Hasonlítsuk ezt össze az eredetivel! Mintha egy hárombájtos ASL utasítást végeztünk volna el (ilyen persze nincs), pedig csak egy nyolcbites ASL és két, szintén nyolcbites ROL utasítást végeztünk. Ha ugyanezt jobbra szerettük volna végezni, akkor először egy LSR utasítást kellett volna végrehajtani a bal szélsőn, majd két ROR utasítást. Először a középsőn, majd a jobb szélsőn, pontosan fordított sorrendben, mint ahogyan a balra tolásnál. Ezek után visszatérhetünk a szorzáshoz.



Egyelőre szorozzunk össze két egybájtos számot, például $FF-et $FF-el! Két egybájtos szám összeszorzásakor tehát az eredmény mindig legfeljebb kétbájtos lehet.

```
 11111111  *  11111111
-----------------------
 11111111
  11111111
   11111111
    11111111
     11111111
      11111111
       11111111
        11111111
-----------------------
1111111100000001
```

Ha a bal oldali számot az 50001. címen, a jobb oldalit az 50002. címen, az eredményt pedig a 50003. és 50004. címeken tároljuk, akkor a feladatot az alábbi gépi kódú programmal oldhatjuk meg:

```
cím    kód         utasítás
---    ---         --------
C000   A9 00       LDA #$00
C002   8D 50 C3    STA $C350
C005   8D 53 C3    STA $C353
C008   8D 54 C3    STA $C354
```

Idáig csak töröltük az eredmény területét, meg az eltolandó bájthoz tartozó felső bájtot. Most jöhet a ciklus.

```
cím    kód         utasítás
---    ---         --------
C00B   A2 08       LDX #$08
```

Először vesszük a jobb oldali szám következő bitjét egy egyszerű LSR utasítással. Ha ezek után a C nem ég, akkor nem kell összeadnunk, átugorjuk az összeadást.

```
cím    kód         utasítás
---    ---         --------
C00D   4E 52 C3    LSR $C352
C010   90 13       BCC $C025
```

Ha ezt a részt nem kellett átugranunk, hozzáadjuk az eredményhez a számot.

```
cím    kód         utasítás
---    ---         --------
C012   AD 51 C3    LDA $C351
C015   18          CLC
C016   6D 54 C3    ADC $C354
C019   8D 54 C3    STA $C354
C01C   AD 50 C3    LDA $C350
C01F   6D 53 C3    ADC $C353
C022   8D 53 C3    STA $C353
```

Most eltolhatjuk egy bittel balra a bal oldali számot:

```
cím    kód         utasítás
---    ---         --------
C025   0E 51 C3    ASL $C351
C028   2E 50 C3    ROL $C350
```

Ha nem hajtottuk még végre a ciklust elégszer, akkor visszaugrunk az elejére.

```
cím    kód         utasítás
---    ---         --------
C02B   CA          DEX
C02C   D0 DF       BNE $C00D
C02E   60          RTS
```

A betöltő:

```
100 FOR I=0 TO 46
110 READ A
120 POKE 49152+I,A
130 NEXT I
140 DATA 169,0,141,80,195,141,83,195,141,84,195
150 DATA 162,8,78,82,195,144,19,173,81,195,24
160 DATA 109,84,195,141,84,195,173,80,195,109,83,195
170 DATA 141,83,195,14,81,195,46,80,195,202,208,223,96
```

A betöltő után az alábbi programot futtassuk le ellenőrzés végett:

```
100 INPUT "A=";A
110 IF A=INT(A) THEN IF A>=0 THEN IF A<=255 THEN 130
120 GOTO 100
130 INPUT "B=";B
140 IF B=INT(B) THEN IF B>=0 THEN IF B<=255 THEN 160
150 GOTO 130
160 POKE 50001,A
170 POKE 50002,B
180 SYS 49152
190 PRINT "A*B=";PEEK(50004)+(256*PEEK(50003))
200 PRINT : RUN
```

Helyes begépelés esetén a program minden esetben tökéletes eredményt ad, vagyis a gépi kódú programunk valóban összeszorozza a két számot. Így tehát minden alapvető matematikai műveletet el tudunk végezni. Üljünk hát le, pihenjünk, és közben kérdezzük ki a gyereket a tizenhatos számrendszerből! Győződjünk meg arról, hogy tud-e fejben összeadni, kivonni és szorozni egybájtos számokat, természetesen tizenhatos számrendszerben!

Kínos dolog, ha a gyerektől olyat kell kikérdezni, amit magunk sem tudunk egészen jól, és így nem tudjuk a választ fejben ellenőrizni. Írjunk egy gépi kódú programot, amely két megadott számot összead, kivon és összeszoroz, és mindhárom eredményt kijelzi a képernyőre! A program alapvetően hat lépésből áll:

1. összeadja a megadott két számot
2. kijelzi az eredményt
3. kivonja egymásból a két számot

4. kijelzi az eredményt
5. összeszorozza a két számot
6. kijelzi az eredményt.

Összesen háromszor kell kijelezni az eredményt, és mivel ez mindig ugyanúgy történik, háromszor kell ugyanazt a programrészletet leírni. Mennyivel rövidebb lenne, csak egyszer megírni, és mindhárom esetben annyit mondani, tegyen úgy a gép, mintha oda is le lenne írva az illető programrészlet. Erre is képesek leszünk, ha elolvassuk a következő fejezetet a veremről és a gépi kódú szubrutinhívásokról.

A VEREM

A „verem" azon számítástechnikai szakkifejezések egyike, amelyeket a mindennapi életben is használunk — csak ott kicsit más a jelentése.

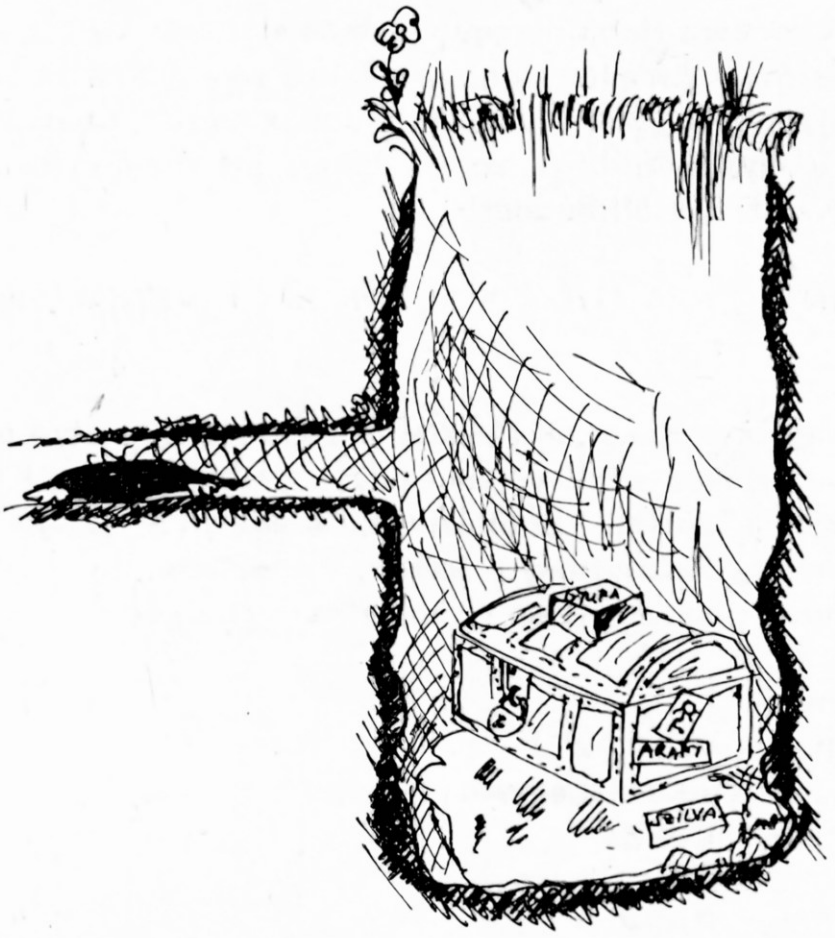
Ahhoz, hogy megértsük a számítógépes verem fogalmát, a köznyelvi verem logikai sémáját kell felderítenünk. Példaként dobjunk egy verembe egy zsák szilvát, egy láda aranyat és egy doboz gyufát. És most jön a lényeg! Vegyük ki a veremből a három tárgyat! Ezt csak úgy tudjuk megtenni, hogy először kivesszük a doboz gyufát, utána a láda aranyat, végül a zsák szilvát. Vagyis éppen fordított sorrendben, mint ahogyan bedobáltuk. Megfogalmazhatjuk tehát a veremtan első főtételét:

„A VEREMBŐL MINDIG AZT TUDJUK ELSŐKÉNT KIVENNI, AMIT UTOLSÓKÉNT DOBTUNK BE."

Lássunk most egy nagyon érdekes alkalmazást! Gondoljunk azokra a játékprogramokra, amelyek a játék megkezdése előtt kiírják, hogy „VIGYÁZZ", „KÉSZ", „RAJT", és csak ezután lehetséges a kezdés. Készítsünk mi is egy ilyen programot! Minden egyes kiírás után illik várni egy kicsit, hogy a játékos elolvashassa, mi van a képernyőn, majd törölni kell. Programunk tehát az alábbi terv szerint fog működni:

1. Töröljük a képernyőt
2. Kiírjuk középre, hogy VIGYAZZ!
3. Picit várunk, majd töröljük a képernyőt
4. Kiírjuk középre, hogy KESZ!
5. Picit várunk, majd töröljük a képernyőt
6. Kiírjuk középre, hogy RAJT!
7. Picit várunk, majd töröljük a képernyőt
8. Kezdődhet a program

Mi most foglalkozzunk csupán az első hét ponttal. Összesen négyszer lesz szükség képernyőtörlésre, és háromszor kell várakozni. A képernyőt törlő programrész maga is több utasításból áll, így programunkban négyszer, négy helyen szerepel ugyanaz az utasítássorozat, amely törli a képernyőt. Sokkal rövidebb lenne, ha csak egyszer szerepelne külön egy képernyőt törlő programrész. (Az ilyen jellegű programrészeket alprogramoknak, idegen szóval szubrutinoknak hívják.) Minden esetben, amikor képernyőt kell törölni, egyszerűen csak utasítanánk a számítógépet, hogy jegyezze meg, hol tart, hajtsa végre



a képernyőt törlő programrészt, s ha végzett, folytassa tovább a programot a megjegyzett címen. Ezt aztán mind a négyszer megtehetnénk, csak azt kellene biztosítani, hogy mindig megjegyezze, honnan kell később folytatnia a főprogramot. Programrészletünk módosult terve az alábbi:

Főprogram

1. Hajtsuk végre az alprogramot a második pontjától (várakozni most nem kell)
2. Írjuk ki a képernyő közepére, hogy VIGYAZZ!
3. Hajtsuk végre a teljes alprogramot!

4. Írjuk ki a képernyő közepére, hogy KESZ!
5. Hajtsuk végre a teljes alprogramot!
6. Írjuk ki a képernyő közepére, hogy RAJT!
7. Hajtsuk végre a teljes alprogramot!

(A nyolcadik pont most az egyszerűség kedvéért legyen az, hogy álljon meg a program!)

Alprogram

1. Várjunk egy kicsit — hajtsunk végre két egymásba ágyazott ciklust! Nyilvánvaló ugyanis, hogy ha a ciklusok végrehajtási számát egy bájtban számoljuk, akkor a végrehajtások maximális száma 256. Egy 256-szor végrehajtandó üres ciklus ugyanakkor csak egy villanásnyi ideig várakoztatná a programot. A nagy ötlet: ismételgessük most ezt a ciklust, azaz az egészet tegyük egy másik ciklusba! Ezt hívják egymásbaágyazott ciklusnak. Figyeljünk a megoldás módjára, mert ez a sablon más programjainkban is felhasználható.
2. Töröljük a képernyőt!
3. Folytassuk a főprogramot a megjegyzett utasításon!

A főprogramból azonban nem ágazhatunk az alprogramra JMP utasítással, mert akkor az alprogram végén nem tudnánk, hol kell folytatni a főprogramot, hiszen az alprogram nem tudhatja, hogy a főprogram mely részéről hívták. Szükségünk lenne valamiféle emlékező JMP utasításra, ami azon kívül, hogy átadja a vezérlést, még megjegyzi az elágazás helyét is. Így aztán az alprogramot befejezhetnénk egy olyan utasítással, amelynek lényege: „térj vissza a főprogramba arra a címre, amit a legutolsó emlékező JMP megjegyzett!" Szerencsére, létezik emlékező JMP utasítás és alprogramot (szubrutint) befejező utasítás is. Emlékező JMP utasítás a JSR, az alprogramot befejező utasítás pedig a RTS. Nézzük meg most, hogy ezek az utasítások valóban működőképesek-e! Írjuk meg programunkat! A főprogram első utasításával hajtsuk végre az alprogramot a képernyő törlésétől kezdve!

```
 cím     kód        utasítás
 ---     ---        --------
C000   20 60 C0     JSR $C060
```

Most írjuk ki a képernyő közepére, hogy „VIGYAZZ!"

```
 cím     kód        utasítás
 ---     ---        --------
C003   A9 16        LDA #$16
C005   8D C9 05     STA $05C9
C008   A9 09        LDA #$09
C00A   8D CA 05     STA $05CA
C00D   A9 07        LDA #$07
C00F   8D CB 05     STA $05CB
C012   A9 19        LDA #$19
C014   8D CC 05     STA $05CC
C017   A9 01        LDA #$01
C019   8D CD 05     STA $05CD
C01C   A9 1A        LDA #$1A
C01E   8D CE 05     STA $05CE
C021   8D CF 05     STA $05CF
```

Hajtsuk végre a teljes alprogramot!

```
cím     kód        utasítás
---     ---        --------
C024  20 56 C0     JSR $C056
```

Írjuk ki a képernyő közepére, hogy „KESZ!"

```
cím     kód        utasítás
---     ---        --------
C027  A9 0B        LDA #$0B
C029  8D CB 05     STA $05CB
C02C  A9 05        LDA #$05
C02E  8D CC 05     STA $05CC
C031  A9 13        LDA #$13
C033  8D CD 05     STA $05CD
C036  A9 1A        LDA #$1A
C038  8D CE 05     STA $05CE
```

Hajtsuk végre az alprogramot!

```
cím     kód        utasítás
---     ---        --------
C03B  20 56 C0     JSR $C056
```

Írjuk ki, hogy „RAJT!"

```
cím     kód        utasítás
---     ---        --------
C03E  A9 12        LDA #$12
C040  8D CB 05     STA $05CB
C043  A9 01        LDA #$01
C045  8D CC 05     STA $05CC
C048  A9 0A        LDA #$0A
C04A  8D CD 05     STA $05CD
C04D  A9 14        LDA #$14
C04F  8D CE 05     STA $05CE
```

Hajtsuk végre az alprogramot!

```
cím     kód        utasítás
---     ---        --------
C052  20 56 C0     JSR $C056
```

Fejezzük be a főprogramot!

```
cím     kód        utasítás
---     ---        --------
C055  60           RTS
```

Most jön az alprogram. Először egymásba ágyazott ciklusokkal késleltetünk.

```
cím     kód        utasítás
---     ---        --------
C056  A2 FF        LDX #$FF
C058  A0 FF        LDY #$FF
C05A  88           DEY
C05B  D0 FD        BNE $C05A
C05D  CA           DEX
C05E  D0 F8        BNE $C058
```

Szintén ciklus segítségével töröljük a képernyőt:

```
cím     kód        utasítás
---     ---        --------
C060  A2 00        LDX #$00
C062  A9 20        LDA #$20
C064  9D 00 04     STA $0400,X
C067  9D 00 05     STA $0500,X
C06A  9D 00 06     STA $0600,X
C06D  9D 00 07     STA $0700,X
C070  E8           INX
C071  D0 F1        BNE $C064
```

Térjünk végül vissza az alprogramból a főprogram soron következő utasítására. Ennek az utasításnak a címét a JSR megjegyezte, így most csak RTS utasítást kell kiadni.

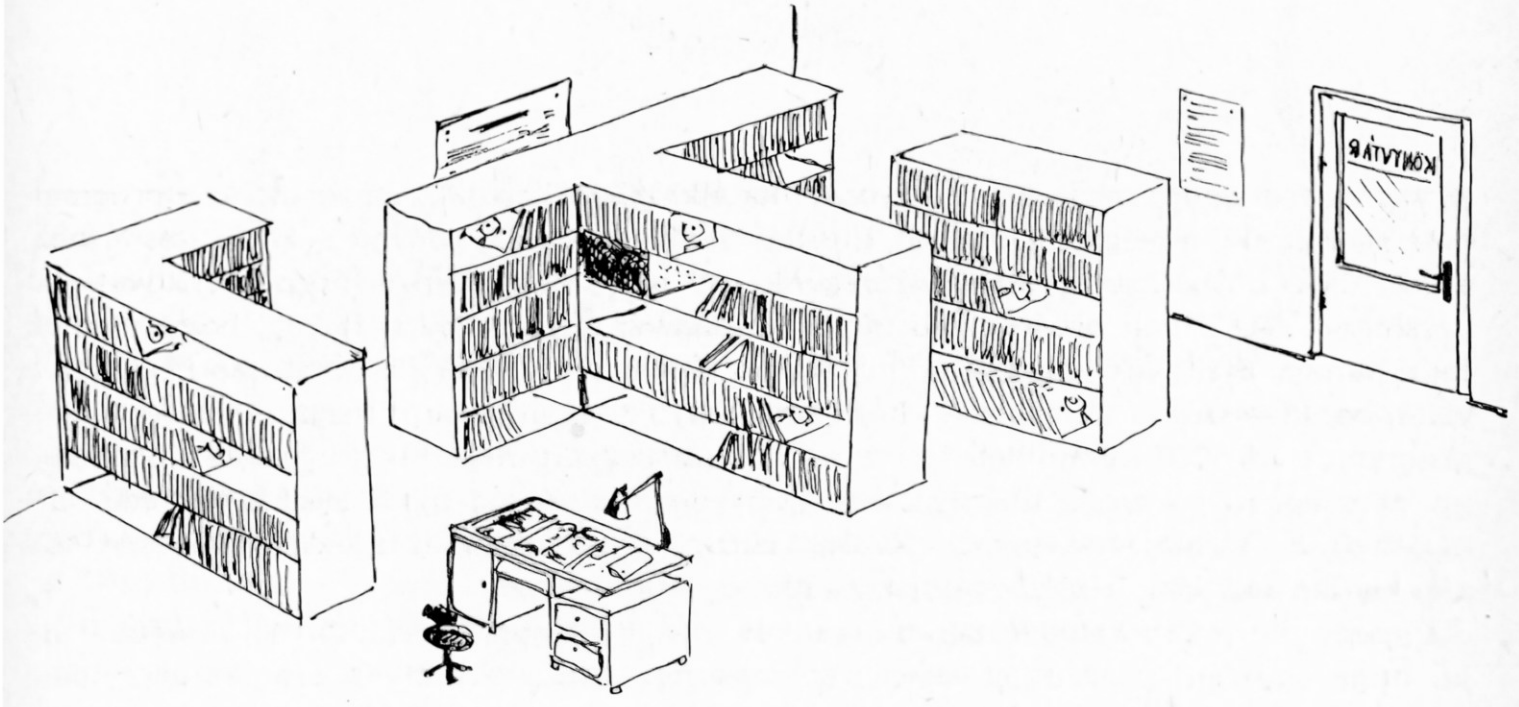
cím	kód	utasítás
C073	60	RTS

A program betöltője:

```
100 S=0
110 FOR I=0 TO 115
120 READ A : S=S+A : POKE 49152+I,A
130 NEXT I : IF S<>13043 THEN PRINT "HIBA VOLT !"
140 DATA 32,96,192,169,22,141,201,5,169,9
150 DATA 141,202,5,169,7,141,203,5,169,25
160 DATA 141,204,5,169,1,141,205,5,169,26
170 DATA 141,206,5,141,207,5,32,86,192,169
180 DATA 11,141,203,5,169,5,141,204,5,169
190 DATA 19,141,205,5,169,26,141,206,5,32
200 DATA 86,192,169,18,141,203,5,169,1,141
210 DATA 204,5,169,10,141,205,5,169,20,141
220 DATA 206,5,32,86,192,96,162,255,160,255
230 DATA 136,208,253,202,208,248,162,0,169,32
240 DATA 157,0,4,157,0,5,157,0,6,157
250 DATA 0,7,232,208,241,96
```

A betöltő futtatása után a program SYS 49152-vel futtatható. Látható, hogy programunk jól működik, hiszen éppen azt csinálja, amit akartunk.
Egy szubrutint „meghívni" a JSR utasítással lehet. A JSR utasítás azt jelenti: „Ugorj a megadott címre, de úgy, hogy jegyezd meg, hová kell visszatérni!"

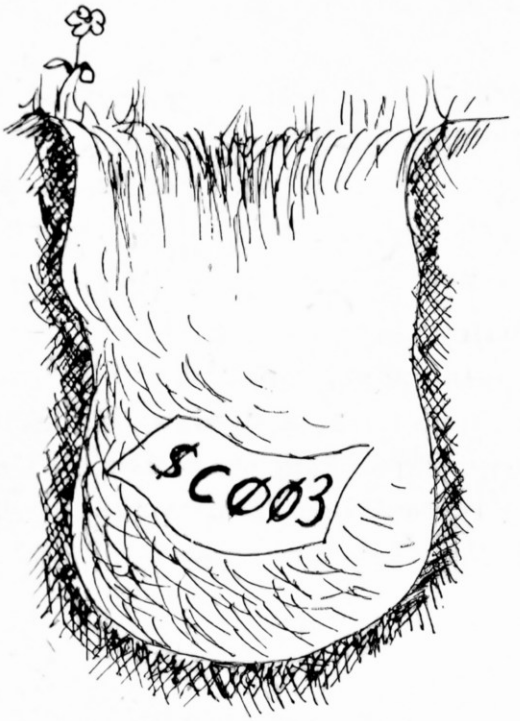
Az RTS utasítás jelentése: „Térj vissza arra a címre, amelyet legutóbb megjegyeztél — aztán ezt a címet el is felejtheted!"
Programunk négy esetben hívta meg ugyanazt a szubrutint, mely minden esetben oda tért vissza, ahonnan a főprogram meghívta. A gyakran ismételt feladatot szubrutinként kezelve rengeteg memóriát takarítottunk meg, és sok felesleges gépeléstől kíméltük meg magunkat. Ezentúl bármilyen hosszú programrészeket is elég lesz egyszer, szubrutinként megírni, majd minden szükséges esetben meghívni egy egyszerű JSR utasítással. Sőt,



az előrelátók a gyakorta használt szubrutinjaikat akár lemezre is menthetik. Így aztán már egyéb programjaikban sem kell újból begépelniük, szükség esetén egyszerűen betölthetik a lemezről. Az igazán előrelátók pedig minden fontos szubrutinukkal megteszik ezt, így az évek során valóságos szubrutinkönyvtáruk keletkezik.

Egy terjedelmes, komoly szubrutinkönyvtárral rendelkező programozó tulajdonképpen szinte már nem is programoz, csak kiválogatja a könyvtárából a szükséges szubrutinokat. Ezzel a módszerrel sokszor törtrészére csökkenhet a programíráshoz szükséges idő. Becsüljük meg a szubrutinokat!

S miért van szükség mindezekhez a veremre? Képzeljük magunkat a programot végrehajtó személy szerepébe, és lesz, ami lesz, a biztonság kedvéért vigyünk magunkkal egy összehajtható „útivermet"! Kezdjük el végrehajtani a programot a $C000 címtől — itt rögtön szembetaláljuk magunkat egy embert próbáló feladattal. Tudjuk, a JSR utasítás azt jelenti, hogy a program végrehajtását a megadott címen kell folytatni, de előbb még a főprogram következő utasításának címét is meg kell jegyezni. Az elágazás nem okoz gondot, de vajon hogyan jegyezzük meg a következő címet? Egyszerű! Írjuk fel egy cédulára, hogy $C003, és dobjuk bele a verembe. Ez is egy módja a megjegyzésnek.



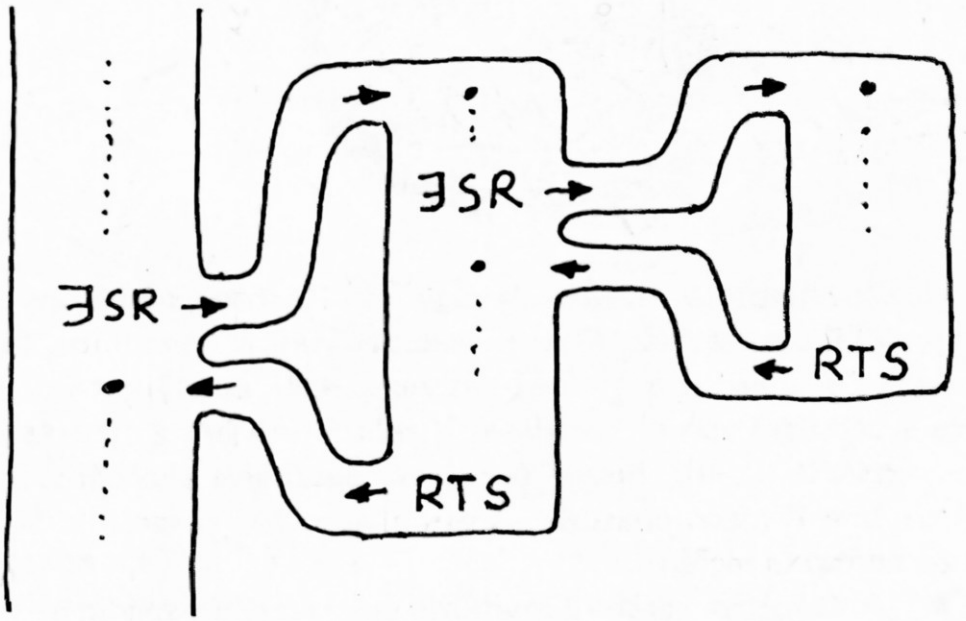
A visszatérési címet megjegyeztük, most már elkezdhetjük az alprogramot. Az alprogramban elvégezzük a képernyőt törlő utasítássorozatot, majd elérünk a RTS utasításhoz, amely újból próbára teszi leleményességünket. Vissza kell térnünk a főprogram következő utasítására. Az álnok programozó most biztosan árgus szemekkel figyeli, hogyan adjuk fel a harcot. Bizonyára tomboló dührohamot kap, amikor lazán és elegánsan benyúlva a verembe, kivesszük a visszatérési címet tartalmazó cédulát, amiről megtudjuk, hogy a főprogramot a $C003 címen kell folytatni. A cédulát pedig elégetjük, hogy a későbbiekben ne okozhasson kavarodást. Folytatjuk a főprogramot, majd újból elérkezünk egy JSR utasításhoz. Természetesen most is úgy járunk el, ahogyan az előbb, csak most más cím kerül a cédulára, később majd erre a címre térünk vissza.

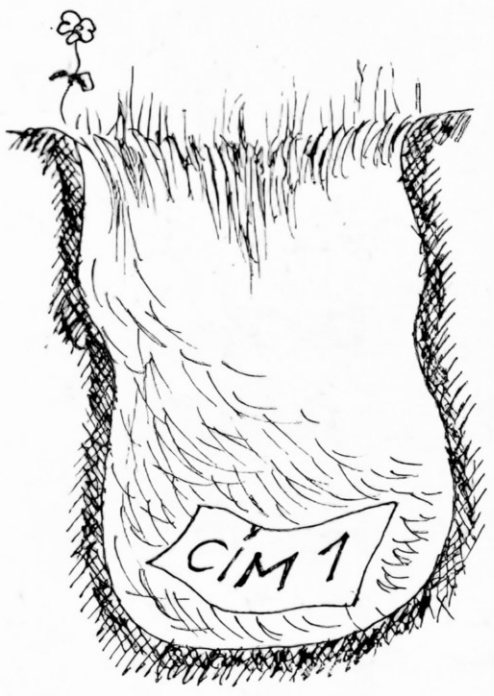
Az igazán álnok programozó persze ilyenkor már újabb gonosz trükkön töri a fejét. Tudja, hogy egyszerű szubrutinhívással nem zavarhat meg, ezért olyan programot ír, ahol

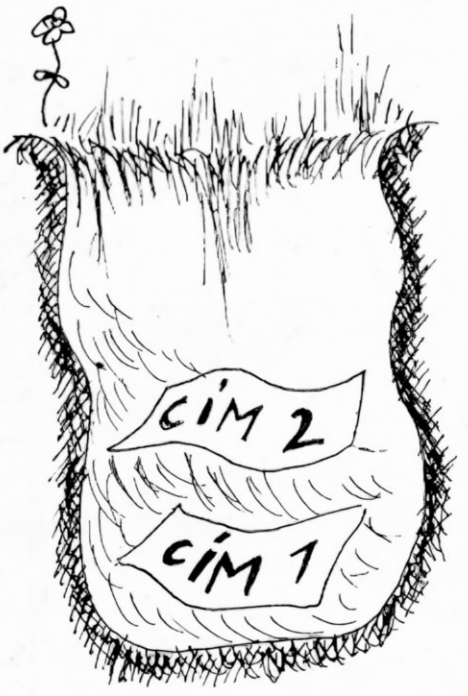
egymásba ágyazott szubrutinok vannak. Van tehát egy főprogram, amely meghív egy szubrutint, de most a szubrutinból még meghívunk egy másik szubrutint is, mielőtt visszatérnénk a főprogramba.

Fogjuk hát a vermünket, és induljunk útnak! Amikor a főprogramból elugrunk a szubrutinba, bekerül a verembe a főprogram következő utasításának címe.

A szubrutint végrehajtva, újból elérkezünk egy JSR utasításhoz. Megnézzük a következő utasítás címét, felírjuk egy cédulára. Ezt a cédulát is bedobjuk a verembe. Ezután elugrunk a szubrutinból hívott másik szubrutinba.

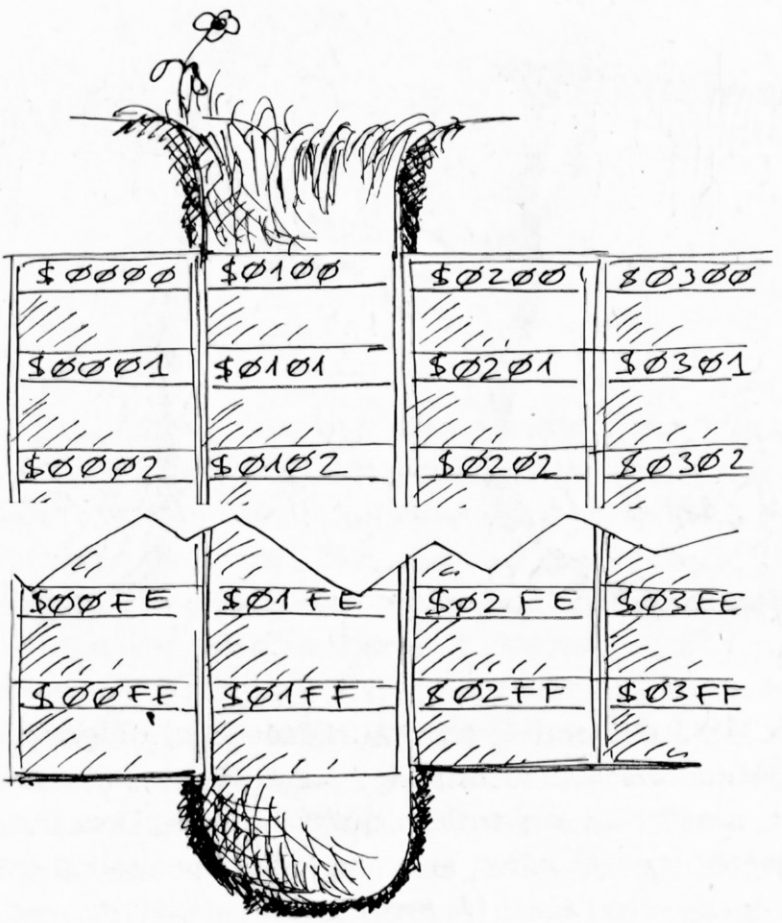






A programozó most kezd el izzadni. Ne törődjünk vele, dolgozzunk a második szubrutinon, amíg el nem érünk az RTS utasításig! Az RTS utasításnál kivesszük a veremből a legfelső cédulát, azt, amelyiket legutoljára dobtunk bele. Erre a cédulára van felírva a legutóbbi elágazást követő utasítás címe, az a cím, amire vissza kell térni. Térjünk hát vissza, vermünkben már csak egy cédulával! (A programozó egyre idegesebb.)

Most tehát az elsőként hívott szubrutinban vagyunk. Itt még van néhány utasítás, majd itt is elérkezünk egy RTS utasításhoz. Most is csak benyúlunk a verembe, és megtudhatjuk a visszatérési címet. Most ugyanis azt a cédulát vesszük ki, amelyiket először dobtuk be, és most éppen erre a cédulára van szükségünk. Visszatérünk hát a főprogramba, ismét üres veremmel. Nyugodtak lehetünk, hiszen a vermet használva akárhányszorosan egymásba ágyazott szubrutinokat is megoldhatunk. A verem automatikusan mindig a szükséges visszatérési címet tartalmazza legfelül.
Természetesen a számítógépes verem a memória egy része, mégpedig az 1. számú oszlopa, vagyis a $0100-tól $01FF-ig terjedő rekeszek.



Ezt a speciális, a valóságostól kissé eltérő vermet a vezérlő speciális módon használja. A vezérlő asztalán van egy SP jelű számláló (Stack Pointer, magyarul veremmutató), amely nyolcbites, azaz hexadecimálisan két számjegyű. Az SP jelzi, hogy melyik a következő szabad rekesz a veremben. Teljesen üres verem esetében az SP $FF-re, azaz a verem aljára, a $01FF rekeszre mutat. Az SP értékéhez ugyanis mindig hozzá kell adni a $0100 értéket, hogy megkapjuk a rekesz valódi címét.
Nézzük most meg, mit mond a kódkönyv a JSR és az RTS utasításokról!

Leírás

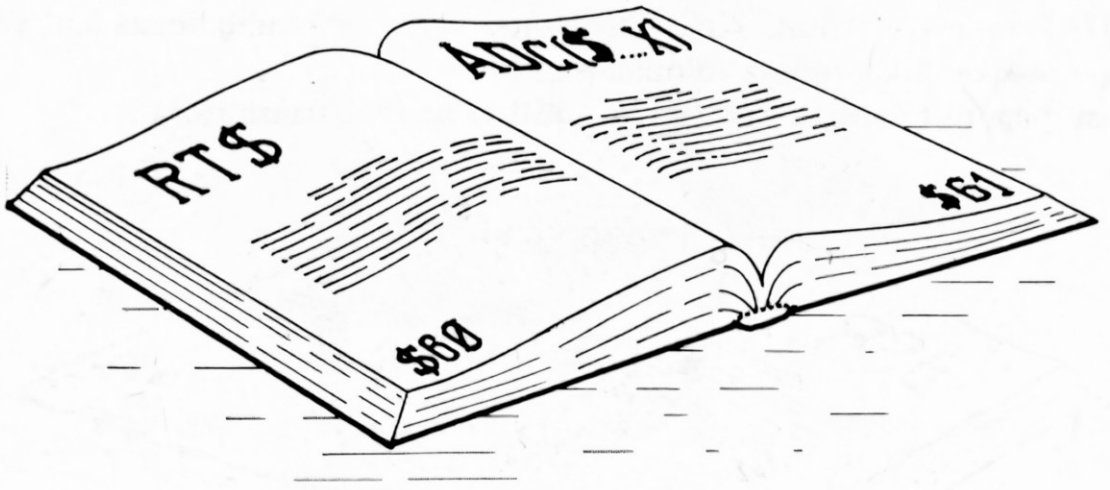
1) Hozassuk be a PC által jelzett címről (az utasításkódot követő bájtról) az ott található számot, és írjuk le egy cédulára!
2) Tekerjünk egyet a PC-n!
3) Hozassuk be a PC által jelzett címről (az utasításkódot követő második bájtról) az ott található számot, és írjuk az előbb leírt szám elé!
4) A PC első két számjegyét küldjük ki az SP által jelzett címre! (SP+$0100) A PC tartalma most éppen a JSR utasítás utolsó bájtjának a címe.
5) Tekerjük vissza eggyel az SP-t!
6) Küldjük ki a PC második két számjegyét az SP által jelzett címre (SP+$0100)!
7) Tekerjük vissza eggyel az SP-t!
8) Állítsuk be a PC-t a cédulán található értékre!

Jelzőbitek kezelése

Ennél az utasításnál a jelzőbiteket nem kezeljük!

A JSR utasítás tehát elteszi a következő utasítás első bájtját megelőző címet a verembe, majd elugrik a meghatározott címre. Amikor ugyanis a vezér nekikezd a következő utasításnak, már az újonnan beállított PC alapján kezd dolgozni.
A veremben a tárolt cím (visszatérési cím-1) felső bájtja van alul, és alsó bájtja van felül.

Ahhoz tehát, hogy a visszatérési cím előtti címet tároljuk, két rekeszt használunk el a veremből.
Nézzük most az RTS utasítást!



Leírás

1) Tekerjük előre eggyel az SP-t!
2) Hozassuk be az SP által meghatározott címről (SP+$0100) az ott található számot, és írjuk fel egy cédulára!
3) Tekerjük előre eggyel az SP-t!
4) Hozzassuk be az SP által meghatározott címről (SP+$0100) az ott található számot, és írjuk az előbb behozott szám elé!
5) Állítsuk be a PC-t a cédulán lévő értékre!
6) Tekerjünk egyet a PC-n!
   (A PC most a legutolsó JSR utasítás utani utasítás kezdőcímét tartalmazza!)

Jelzőbitek kezelése

Ennél az utasításnál a jelzőbiteket nem kezeljük!

Az RTS utasítás hatására a vezérlő behozat a veremből két számot, ezekből összeállít egy kétbájtos számot, ezt megnöveli eggyel, és ezt tekinti a következő végrehajtandó utasítás címének. A programvégrehajtás tehát a JSR utasítást követő utasításon folytatódik.
Mivel a verem összesen 256 rekeszből áll, ezért maximálisan 128-szorosan egymásba ágyazott szubrutinokat tudunk kezelni. Meg kell azonban jegyezni, hogy ilyen szövevényes és komplikált programot nem szokás írni.
Az SP értéke nemcsak a JSR és RTS utasítások hatására változik, hanem mi magunk is beállíthatjuk bármilyen értékre a TXS utasítással. A TSX utasítással pedig kiolvashatjuk, hogy mennyit mutat az SP.

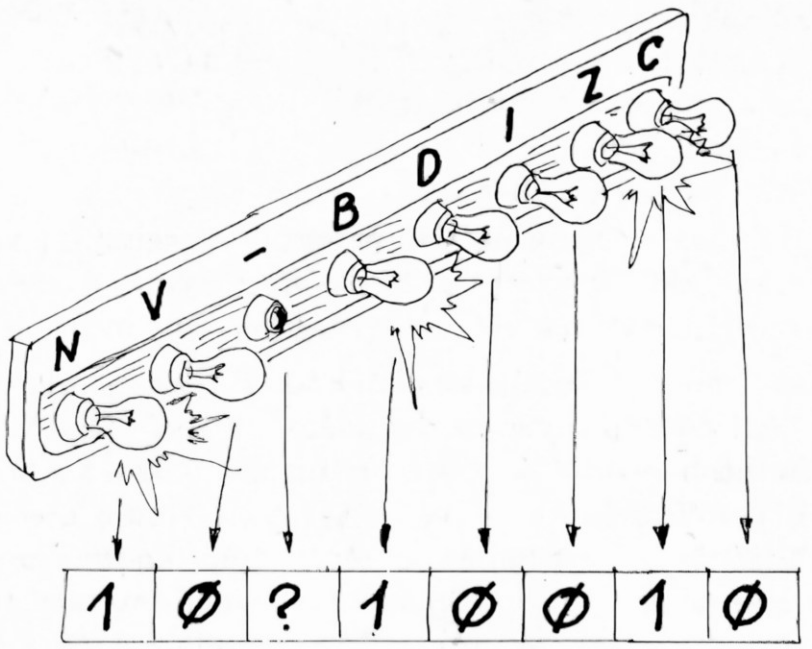
A TXS utasítás az X regiszter értékét átírja az SP-be (beállítja SPt-t az X-ben megadott értékre).
A TSX utasítás az SP értékét átírja az X-be (kiolvassa X-be, hogy milyen értéken áll az SP).

A veremmutató szemszögéből nézve tehát az X regiszter különleges, mert csak vele tart fenn ilyen közvetlen adatátviteli kapcsolatot. Az SP-ből csak az X-be, az SP-be csak az X-ből lehet írni.
A veremben nemcsak visszatérési címeket tárolhatunk. Elmenthetjük benne az A regiszter tartalmát és a jelzőbiteket is. Ezekben az esetekben azonban az SP csak egy értékkel módosul, hiszen mind az A regiszter, mind a jelzőbitek elférnek egy rekeszben. Az A regiszterrel kapcsolatos utasítások:

PHA : Az A regiszter értékét menti el a verembe, és a veremmutató értékét ennek megfelelően eggyel csökkenti.

PLA : Kiolvas a veremből egy értéket az A regiszterbe, és a veremmutató értékét ennek megfelelően eggyel növeli.

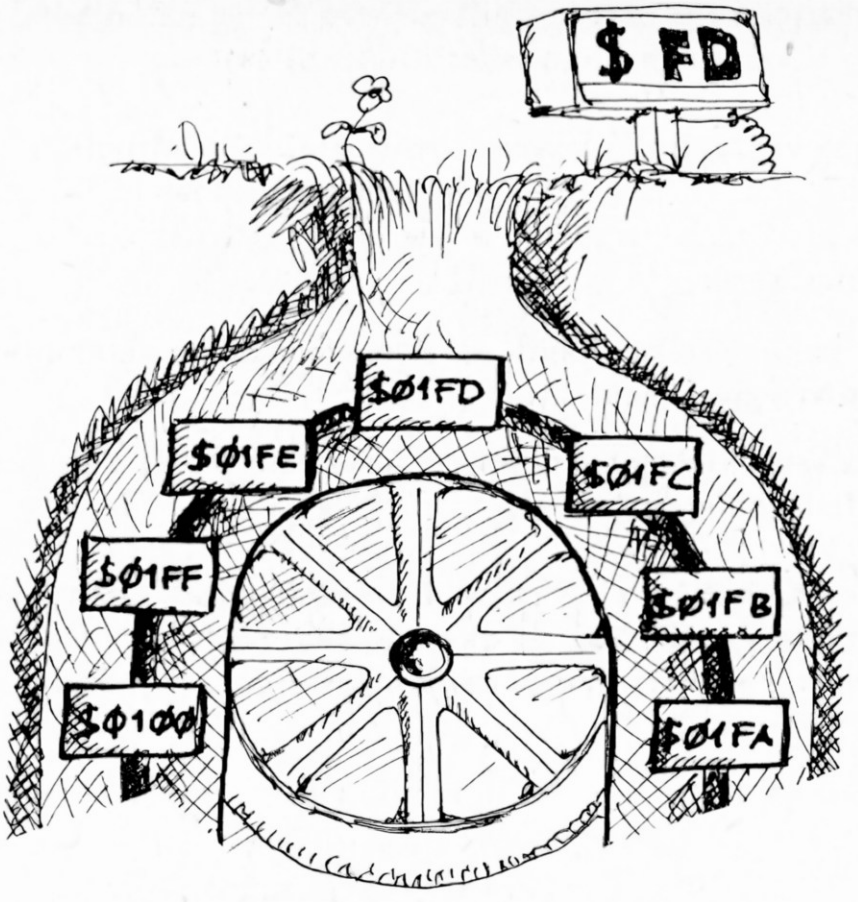
A jelzőbiteket is elmenthetjük a verembe. Egyszerűen csak számokkal kell megjelölni a jelzőbitek állapotát. Ha egy jelzőbit ég, akkor a neki megfelelő helyre egyet írunk, egyébként pedig nullát. Így nyolcbites számot kapunk, amelyet már könnyű a verembe írni.



PHP : Elmenti a verembe a jelzőbitek állapotát leíró számot.

PLP : A veremből kivett szám alapján állítja be a jelzőbiteket.

Illik tudni, hogy az SP verembe íráskor $00 után ismét $FF-et mutat, kiolvasáskor $FF után pedig $00-át. Úgy kell elképzelni, mintha a verem nem oszlopszerű lenne, hanem körkörös. Így tehát soha nem akadhat meg a program azért, mert „betelt a verem". Ennek azonban megvan az a veszélye, hogy ha túl sok mindent akarunk tarolni a veremben, akkor felülírhatunk olyan adatokat, amelyeket régebben tettünk bele. Ez pedig az információ elveszítéséhez vezet, ami adott esetben végzetes is lehet. Éppen ezért, ha lehetséges, a veremben csak a legszükségesebb dolgokat tároljuk, és csak a lehető legrövidebb ideig! Az STA, STX, STY utasításokkal (amelyekkel bármilyen memóriacímet, tehát



a verem területére eső címet is feltölthetünk) ne tároljunk semmit a veremterületre, mert ezzel felülírhatunk még „élő'' visszatérési címeket. Tartsuk tiszteletben az 1. számú oszlopot, hiszen a memóriában ezen kívül is rengeteg hely van még!

Hamarosan az is kiderül, miért fejezzük be gépi kódú programjainkat RTS utasítással, mi történik a gépben a gépi kódú programok futtatása előtt és után, és a gép bekapcsolása után milyen titokzatos mechanizmus hatására tud bemutatkozni a gép a képernyőn.

A számítógép csak a memóriában elhelyezett számokat tudja utasításként értelmezni, és csak a gépi kódú utasítások végrehajtására képes. Gépi kódban programozni azonban nehézkes és kissé hosszadalmas is. A programozás megkönnyítése érdekében írtak egy Basic rendszerprogramot. Ez is gépi kódú program, mindig megtalálható a gép memóriájában, s a gép bekapcsolásakor azonnal futni kezd. Amikor Basic nyelven programozunk, akkor ez a rendszerprogram fut, és ez teszi lehetővé, hogy az utasítások beépelésére a gép többé-kevésbé értelmesen válaszoljon. Ez a program figyeli a billentyűzetet, raktározza az általunk írt szövegeket, értelmezi őket, és a szövegek tartalmától függően különböző szubrutinokat hajt végre. A mi Basic programjaink csak adatai a rendszerprogramnak, ezen adatok alapján jutnak szóhoz a rendszer különböző szubrutinjai. A rendszer temérdek szubrutin többszintű, szerteágazó hierarchiája.

A memóriateremben két típusú rekesz található. Az egyik típus tartalma bármikor megváltoztatható, a memóriakezelő ezekben bármikor átrendezheti a mérlegeket. Ez a memóriatípus a RAM (Random Access Memory = közvetlen elérésű, írható és olvasható memória). Vannak azonban olyan rekeszek is, melyekben nincs mérleg. Ezek a rekeszek csupán a falra vannak felrajzolva. Tartalmuk bármikor, bármelyik oldalról megtekinthető, de meg nem változtatható, ezért van az, hogy egyes gépi kódú STA, STX, STY utasítások hatástalanok. Ilyen memóriarekeszekben nem lehet számot tartani. Ez a terület a ROM

(Read Only Memory = csak olvasható memória). Ezekben az örök rekeszekben van pl. a Basic rendszerprogram, és ez teszi lehetővé, hogy minden egyes bekapcsoláskor ugyanaz a Basic rendszer fusson. Ez a rendszerprogram a $FCE2, azaz a 64738 címen kezdődik, ez a magyarázata annak, hogy a SYS 64738 hatására ugyanaz történik, mintha a gépet bekapcsolnánk. (Erről bővebben a COMMODORE 64 ROM programja c. könyvben olvashatunk, kiadta az Ipari Informatika Központ, Bp. 1985.)

A számítógépben alaphelyzetben a Basic rendszerprogram fut. Ha begépeljük pl. a SYS 49152 utasítást, akkor azt is a rendszerprogram veszi kezelésbe. Megállapítja, hogy ez gépi kódú programhívás, ezért (JSR utasítással) átadja a vezérlést a megjelölt címre, nem törődve azzal, hogy ott mi történik. Programunk tehát a Basic rendszerprogram alprogramjaként működik! Ha azt akarjuk, hogy a rendszerprogram visszakapja a vezérlést (pl. fusson tovább a gépi kódú programunkat hívó Basic program), a gépi programot RTS utasítással kell befejeznünk.

A rendszer tehát maga is szubrutinok kusza szövevénye, áttekintéséhez sok idő és fáradság szükséges. A Basic rendszer a memóriában a $A000–$BFFF, valamint a $E000–$FFFF címek közötti területen található. A rendszer által gyakran hívott, fontos szubrutinokhoz a gépi kódú programozó is könnyen hozzáférhet. Ezeket a fontos szubrutinokat nemcsak a kezdőcímük alapján jegyzik, hanem nevet is szoktak nekik adni. Található például olyan szubrutin, amely az A regiszterben tárolt ASCII-kódnak megfelelő karaktert nyomtatja ki a képernyőre. Ez a CHROUT nevű rutin, amely a $FFD2 címen kezdődik. (Az egyes karakterek ASCII-kódjait ld. a függelékben.) Ha tehát a gépi kódú programozó törölni akarja a képernyőt (ASCII-kódja $93), akkor csak ennyit ír:

```
LDA  #$ 93
JSR  $FFD2
```

Ezeket a gyakran használt szubrutinokat gyűjtőnévvel KERNAL rutinoknak nevezzük. (A leggyakrabban használt KERNAL rutinokat ld. 138. old.) Ezek a szubrutinok nagymértékben megkönnyítik és lerövidítik a programozói munkát és a programokat is. Segítségükkel kezelhetjük gépi kódból a lemezegységet, a nyomtatót és minden egyéb perifériát.

A SUPER 64 MON monitorprogram számos mechanikus, könnyen téveszthető munkát megkönnyít vagy teljesen levesz a vállunkról: a gépelést, az utasításkódok keresését, megjegyzését stb. Aki igazán szeretne előrehaladni a gépi kódú programozásban, nem pazarolhatja az idejét DATA sorok szakadatlan gépelésére.

Pihenésképpen ismerjük meg a legegyszerűbb gépi kódú utasítást. A rövidítése NOP (No Operation – nincs művelet), és nincsen semmilyen hatása (eltekintve persze a PC továbblépésétől). Tehát, ha a vezérlő NOP utasítással találkozik, akkor egyszerűen, minden egyéb tevékenység nélkül veszi a következő utasítást.

Ha egy programból utasításokat törölnénk, és ezért a későbbi utasítások előbbre csúsznának, minden cím megváltozna. Így a JSR és JMP utasítások címeit is át kellene írni, ez pedig körülményes. Ilyenkor írjunk inkább NOP-okat a kihagyott utasítások helyére. A NOP utasításokon ugyanis a vezérlő áthalad, mintha ott se lennének, mintha kiemeltük volna a programból a nem kívánt utasításokat. Az előrelátó programozó programjaiba előre is elhelyezhet NOP-okat, ezzel helyet hagyva a későbbi esetleges betoldások számára. A most közölt esetben mi is ezt tesszük.

Írjuk most meg a gyerek kikérdezését segítő programot!

Ki szeretnénk tehát kérdezni a gyereket a tizenhatos számrendszerből. Ehhez elsősorban egy hexadecimális kijelzőre van szükségünk, majd a tizenhatos számrendszerbeli számjegyek ASCII-kódjaira. Helyezzük el először ezeket az első 16 rekeszbe! Kezdjük a programot a 49152. ($C000) címen. A memóriába először az alábbi számokat kell tenni:

$C000 – $30 A "0" kódja = 48
$C001 – $31 Az "1" kódja = 49
$C002 – $32 A "2" kódja = 50
$C003 – $33 A "3" kódja = 51
$C004 – $34 A "4" kódja = 52
$C005 – $35 Az "5" kódja = 53
$C006 – $36 A "6" kódja = 54
$C007 – $37 A "7" kódja = 55
$C008 – $38 A "8" kódja = 56
$C009 – $39 A "9" kójda = 57
$C00A – $41 Az "A" kódja = 65
$C00B – $42 A "B" kódja = 66
$C00C – $43 A "C" kódja = 67
$C00D – $44 A "D" kódja = 68
$C00E – $45 Az "E" kódja = 69
$C00F – $46 Az "F" kójda = 70

Ezeken a számokon kívül szükségünk lesz még néhány memóriarekeszre:

$C010 – segédrekesz a szorzáshoz
$C011 – az egyik szám (A)
$C012 – a másik szám (B)
$C013 – az eredmény előjele ($00, ha negatív – $01, ha pozitív)
$C014 – az eredmény felső bájtja
$C015 – az eredmény alsó bájtja

A $C016 címen kezdődik az eredmény kijelzése. Ez most a szubrutin, amelyet többször fogunk meghívni a főprogramból. Ha az eredmény előjele negatív, akkor kiírunk egy "–" jelet, ha pedig pozitív, akkor egy üres karaktert. A – jel ASCII kódja: $2D, az üres karakteré pedig $20. A karaktereket a $FFD2 címen levő KERNAL CHROUT rutinnal írjuk ki a képernyőre.

```
cím     kód         utasítás
---     ---         --------
C016  AD 13 C0    LDA $C013
C019  D0 05       BNE $C020
C01B  A9 2D       LDA #$2D
C01D  4C 22 C0    JMP $C022
C020  A9 20       LDA #$20
C022  20 D2 FF    JSR $FFD2
```

Nyomtassunk most ki egy $ jelet, jelezve, hogy hexadecimális számról van szó! (A dollárjel kódja $24.)

```
 cím     kód         utasítás
 ---     ---         --------
C025  A9 24         LDA #$24
C027  20 D2 FF      JSR $FFD2
```

Most következik az eredmény legelső számjegyének kinyomtatása. Az eredmény legelső számjegye a $C014 cím felső négy bitjén foglal helyet. Először tehát betöltjük az A regiszterbe, és lehozzuk annak alsó négy bitjére:

```
 cím     kód         utasítás
 ---     ---         --------
C02A  AD 14 C0      LDA $C014
C02D  4A            LSR
C02E  4A            LSR
C02F  4A            LSR
C030  4A            LSR
```

Mivel egy szám ASCII kódja nem azonos az illető számmal, ezért mielőtt kinyomtathatnánk, be kell töltenünk az A regiszterbe az adott szám kódját. Most jön jól a memóriában levő táblázatunk.

```
 cím     kód         utasítás
 ---     ---         --------
C031  A8            TAY
C032  B9 00 C0      LDA $C000,Y
C035  20 D2 FF      JSR $FFD2
```

A második számjegy kiküldése csak annyiban tér el, hogy az eredmény felső bájtjának alsó négy bitjén helyezkedik el. Ennek megfelelően, először nullázni kell a felső négy bitet, csak utána kereshető a kód. Végül ezt a karaktert is kinyomtatjuk.

```
 cím     kód         utasítás
 ---     ---         --------
C038  AD 14 C0      LDA $C014
C03B  29 0F         AND #$0F
C03D  A8            TAY
C03E  B9 00 C0      LDA $C000,Y
C041  20 D2 FF      JSR $FFD2
```

Az eredmény utolsó két bájtját ugyanúgy intézzük el, mint az első kettőt, de ezeket a $C015 címről vesszük.

```
 cím     kód         utasítás
 ---     ---         --------
C044  AD 15 C0      LDA $C015
C047  4A            LSR
C048  4A            LSR
C049  4A            LSR
C04A  4A            LSR
C04B  A8            TAY
C04C  B9 00 C0      LDA $C000,Y
C04F  20 D2 FF      JSR $FFD2
C052  AD 15 C0      LDA $C015
C055  29 0F         AND #$0F
C057  A8            TAY
C058  B9 00 C0      LDA $C000,Y
C05B  20 D2 FF      JSR $FFD2
```

A további fejleszthetőség érdekében most kihagyunk három bájtot, vagyis három NOP utasítás következik. Ezzel vége is van a kijelzésnek. Az eddig megírt programrész kijelzi hexadecimálisan az eredményt, így lezárhatjuk egy RTS utasítással!

Most kezdődik el a főprogram. Ez a program belépési pontja. Először is nullázzuk az eredmény felső bájtját, majd az A jelű számot áthelyezzük a $C011 címről a $C015 cím-

```
 cím     kód        utasítás
 ---     ---        --------
C05E  EA           NOP
C05F  EA           NOP
C060  EA           NOP
C061  60           RTS
```

re, vagyis az eredmény alsó bájtjába. Az eredmény előjelét pozitívra állítjuk azzal, hogy a $C013 címre $01-et írunk.

```
 cím     kód        utasítás
 ---     ---        --------
C062  A9 00        LDA #$00
C064  8D 14 C0     STA $C014
C067  AD 11 C0     LDA $C011
C06A  8D 15 C0     STA $C015
C06D  A9 01        LDA #$01
C06F  8D 13 C0     STA $C013
```

Táblázatunk a két számot, összegüket, különbségüket és szorzatukat fogja tartalmazni. Töröljük a képernyőt, mielőtt bármit kiírnánk!

```
 cím     kód        utasítás
 ---     ---        --------
C072  A9 93        LDA #$93
C074  20 D2 FF     JSR $FFD2
```

A forma rendezettsége és szépsége kedvéért nyomtatunk először öt szóközt, majd azt a szöveget, hogy: 'A=$'. (Ez az A szám kiírásának az eleje.)

```
 cím     kód        utasítás
 ---     ---        --------
C077  A9 20        LDA #$20
C079  20 D2 FF     JSR $FFD2
C07C  20 D2 FF     JSR $FFD2
C07F  20 D2 FF     JSR $FFD2
C082  20 D2 FF     JSR $FFD2
C085  20 D2 FF     JSR $FFD2
C088  A9 41        LDA #$41
C08A  20 D2 FF     JSR $FFD2
C08D  A9 3D        LDA #$3D
C08F  20 D2 FF     JSR $FFD2
C092  A9 24        LDA #$24
C094  20 D2 FF     JSR $FFD2
```

Hívjuk meg a kijelző szubrutint, de nem az elejétől kezdve, mert az eredménynek csupán utolsó két számjegyét nyomtatjuk ki. Miután megtörtént a kijelzés, és visszatértünk a főprogramba, sort emelünk a $0D kóddal. Ezzel az A szám kijelzése teljes.

```
 cím     kód        utasítás
 ---     ---        --------
C097  20 44 C0     JSR $C044
C09A  A9 0D        LDA #$0D
C09C  20 D2 FF     JSR $FFD2
```

Ugyanezt elvégezzük az értelemszerű változtatásokkal a B jelű számra is.

```
 cím     kód        utasítás
 ---     ---        --------
C09F  AD 12 C0     LDA $C012
C0A2  8D 15 C0     STA $C015
C0A5  A9 20        LDA #$20
C0A7  20 D2 FF     JSR $FFD2
C0AA  20 D2 FF     JSR $FFD2
C0AD  20 D2 FF     JSR $FFD2
```

```
C0B0  20 D2 FF    JSR $FFD2
C0B3  20 D2 FF    JSR $FFD2
C0B6  A9 42       LDA #$42
C0B8  20 D2 FF    JSR $FFD2
C0BB  A9 3D       LDA #$3D
C0BD  20 D2 FF    JSR $FFD2
C0C0  A9 24       LDA #$24
C0C2  20 D2 FF    JSR $FFD2
C0C5  20 44 C0    JSR $C044
C0C8  A9 0D       LDA #$0D
C0CA  20 D2 FF    JSR $FFD2
```

A két kiindulási szám kijelzése után nézzük az összegüket! Először is adjuk össze a számokat úgy, hogy az összeg az eredmény felső és alsó bájtjába kerüljön!

```
 cím     kód        utasítás
 ---     ---        --------
C0CD  AD 11 C0    LDA $C011
C0D0  18          CLC
C0D1  6D 12 C0    ADC $C012
C0D4  8D 15 C0    STA $C015
C0D7  A9 00       LDA #$00
C0D9  8D 14 C0    STA $C014
C0DC  90 03       BCC $C0E1
C0DE  EE 14 C0    INC $C014
```

Most következhet az összeg kinyomtatása. Először kiírjuk a szöveget: 'A+B='. Ezek után meghívjuk az eredményt kijelző szubrutint, és végül sort emelünk.

```
 cím     kód        utasítás
 ---     ---        --------
C0E1  A9 41       LDA #$41
C0E3  20 D2 FF    JSR $FFD2
C0E6  A9 2B       LDA #$2B
C0E8  20 D2 FF    JSR $FFD2
C0EB  A9 42       LDA #$42
C0ED  20 D2 FF    JSR $FFD2
C0F0  A9 3D       LDA #$3D
C0F2  20 D2 FF    JSR $FFD2
C0F5  20 16 C0    JSR $C016
C0F8  A9 0D       LDA #$0D
C0FA  20 D2 FF    JSR $FFD2
```

Lássuk a kivonást! Abban a reményben, hogy az eredmény pozitív lesz, elvégezzük a kivonást. Ha az eredmény valóban pozitív, rögtön elugrunk a kijelző programrészre.

```
 cím     kód        utasítás
 ---     ---        --------
C0FD  A9 01       LDA #$01
C0FF  8D 13 C0    STA $C013
C102  AD 11 C0    LDA $C011
C105  38          SEC
C106  ED 12 C0    SBC $C012
C109  8D 15 C0    STA $C015
C10C  A9 00       LDA #$00
C10E  8D 14 C0    STA $C014
C111  B0 20       BCS $C133
```

Ha viszont az eredmény negatív lett, akkor a felső bájtot csökkentenünk kell (a felső bájtnak most $FF-nek kell lennie, hogy a kettes komplemens képzése helyes eredményt adjon). Jelezzük, hogy az eredmény negatív, és negáljuk.

```
cím     kód         utasítás
---     ---         --------
C113  CE 14 C0      DEC $C014
C116  A9 00         LDA #$00
C118  8D 13 C0      STA $C013
C11B  AD 14 C0      LDA $C014
C11E  49 FF         EOR #$FF
C120  8D 14 C0      STA $C014
C123  AD 15 C0      LDA $C015
C126  49 FF         EOR #$FF
C128  8D 15 C0      STA $C015
```

Hátra van még a kettes komplemens képzésből az eggyel való növelés:

```
cím     kód         utasítás
---     ---         --------
C12B  EE 15 C0      INC $C015
C12E  D0 03         BNE $C133
C130  EE 14 C0      INC $C014
```

Most már nyugodt lélekkel írathatjuk ki az eredményt. Először kiírjuk, hogy 'A–B=', majd meghívjuk a kijelző rutint, végül sort emelünk:

```
cím     kód         utasítás
---     ---         --------
C133  A9 41         LDA #$41
C135  20 D2 FF      JSR $FFD2
C138  A9 2D         LDA #$2D
C13A  20 D2 FF      JSR $FFD2
C13D  A9 42         LDA #$42
C13F  20 D2 FF      JSR $FFD2
C142  A9 3D         LDA #$3D
C144  20 D2 FF      JSR $FFD2
C147  20 16 C0      JSR $C016
C14A  A9 0D         LDA #$0D
C14C  20 D2 FF      JSR $FFD2
```

Az utolsó feladat a két szám összeszorzása. Először beállítjuk az előjelet pozitívra, majd nullázzuk a szorzáshoz használt segédrekeszt és az eredményt majdan tartalmazó rekeszeket:

```
cím     kód         utasítás
---     ---         --------
C14F  A9 01         LDA #$01
C151  8D 13 C0      STA $C013
C154  A9 00         LDA #$00
C156  8D 10 C0      STA $C010
C159  8D 14 C0      STA $C014
C15C  8D 15 C0      STA $C015
```

A szorzást most is ugyanúgy végezzük, mint a már ismertetett példaprogramban. Maga a szorzás egy nyolcszor végrehajtandó ciklusból áll:

```
cím     kód         utasítás
---     ---         --------
C15F  A0 08         LDY #$08
```

Eltoljuk eggyel jobbra a B számot. Ha ezek után a C jelzőbit üres, akkor nem kell semmit hozzáadnunk az eredményhez (ebben az esetben átugorhatjuk ezt a programrészt).

```
cím     kód         utasítás
---     ---         --------
C161  4E 12 C0      LSR $C012
C164  90 13         BCC $C179
```

Ha a C jelzőbit égett, akkor hozzá kell adnunk az eredményhez az eltolt A számot.

```
cím     kód         utasítás
---     ---         --------
C166   AD 11 C0     LDA $C011
C169   18           CLC
C16A   6D 15 C0     ADC $C015
C16D   8D 15 C0     STA $C015
C170   AD 10 C0     LDA $C010
C173   6D 14 C0     ADC $C014
C176   8D 14 C0     STA $C014
```

Az összeadás után eltoljuk balra egy bittel az A számot.

```
cím     kód         utasítás
---     ---         --------
C179   0E 11 C0     ASL $C011
C17C   2E 10 C0     ROL $C010
```

Ha ez még nem a ciklus nyolcadik végrehajtása volt, akkor megismételjük.

```
cím     kód         utasítás
---     ---         --------
C17F   88           DEY
C180   D0 DF        BNE $C161
```

Ha a program eléri az ezutáni utasítást, akkor nyilván kész a ciklus nyolc végrehajtása. Már csak kijelezzük a szorzatot. Kiírjuk, hogy 'A*B=', meghívjuk a kijelző rutint, és ezzel (több elvégzendő művelet nem lévén) le is zárjuk a programot.

```
cím     kód         utasítás
---     ---         --------
C182   A9 41        LDA #$41
C184   20 D2 FF     JSR $FFD2
C187   A9 2A        LDA #$2A
C189   20 D2 FF     JSR $FFD2
C18C   A9 42        LDA #$42
C18E   20 D2 FF     JSR $FFD2
C191   A9 3D        LDA #$3D
C193   20 D2 FF     JSR $FFD2
C196   20 16 C0     JSR $C016
C199   60           RTS
```

Aki a programot nem monitorprogramból írta be, gépelje be az alábbi betöltőt:

```
100 S=0
110 FOR I=0 TO 409
120 READ A
130 S=S+A : POKE 49152+I,A
140 NEXT I
150 IF S<>50931 THEN PRINT "GEPELESI HIBA VOLT !"
160 DATA 48,49,50,51,52,53,54,55,56,57
170 DATA 65,66,67,68,69,70,0,0,0,0
180 DATA 0,0,173,19,192,208,5,169,45,76
190 DATA 34,192,169,32,32,210,255,169,36,32
200 DATA 210,255,173,20,192,74,74,74,74,168
210 DATA 185,0,192,32,210,255,173,20,192,41
220 DATA 15,168,185,0,192,32,210,255,173,21
230 DATA 192,74,74,74,74,168,185,0,192,32
240 DATA 210,255,173,21,192,41,15,168,185,0
250 DATA 192,32,210,255,234,234,234,96,169,0
260 DATA 141,20,192,173,17,192,141,21,192,169
270 DATA 1,141,19,192,169,147,32,210,255,169
280 DATA 32,32,210,255,32,210,255,32,210,255
290 DATA 32,210,255,32,210,255,169,65,32,210
300 DATA 255,169,61,32,210,255,169,36,32,210
310 DATA 255,32,68,192,169,13,32,210,255,173
```

```
320 DATA 18,192,141,21,192,169,32,32,210,255
330 DATA 32,210,255,32,210,255,32,210,255,32
340 DATA 210,255,169,66,32,210,255,169,61,32
350 DATA 210,255,169,36,32,210,255,32,68,192
360 DATA 169,13,32,210,255,173,17,192,24,109
370 DATA 18,192,141,21,192,169,0,141,20,192
380 DATA 144,3,238,20,192,169,65,32,210,255
390 DATA 169,43,32,210,255,169,66,32,210,255
400 DATA 169,61,32,210,255,32,22,192,169,13
410 DATA 32,210,255,169,1,141,19,192,173,17
420 DATA 192,56,237,18,192,141,21,192,169,0
430 DATA 141,20,192,176,32,206,20,192,169,0
440 DATA 141,19,192,173,20,192,73,255,141,20
450 DATA 192,173,21,192,73,255,141,21,192,238
460 DATA 21,192,208,3,238,20,192,169,65,32
470 DATA 210,255,169,45,32,210,255,169,66,32
480 DATA 210,255,169,61,32,210,255,32,22,192
490 DATA 169,13,32,210,255,169,1,141,19,192
500 DATA 169,0,141,16,192,141,20,192,141,21
510 DATA 192,160,8,78,18,192,144,19,173,17
520 DATA 192,24,109,21,192,141,21,192,173,16
530 DATA 192,109,20,192,141,20,192,14,17,192
540 DATA 46,16,192,136,208,223,169,65,32,210
550 DATA 255,169,42,32,210,255,169,66,32,210
560 DATA 255,169,61,32,210,255,32,22,192,96
```

Ha lefuttattuk a betöltőt, akkor a programot az alábbi Basic programmal használhatjuk. A listában szereplő inverz karaktereket a következőképpen gépelhetjük be:

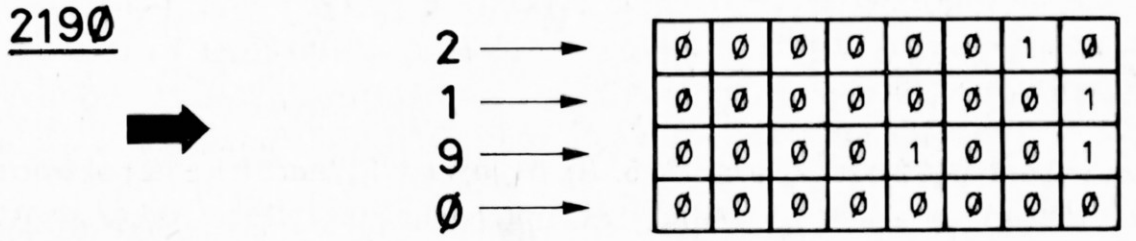
INVERZ SZÍV : SHIFT+CLR/HOME
INVERZ 'S' : CLR/HOME
INVERZ 'Q' : CRSR ⇒

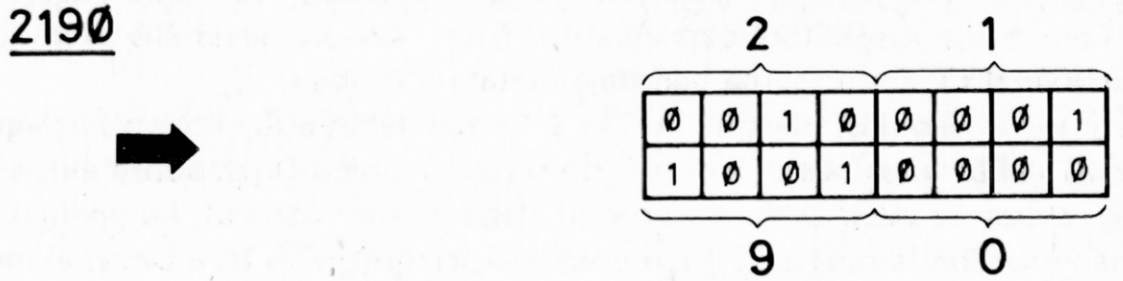
```
100 PRINT "♥"
110 INPUT "SQQQQQQQQQQQQQQQQA=";A
120 IF A=INT(A) THEN IF A>=0 THEN IF A<=255 THEN POKE 49169,A : GOTO 140
130 GOTO 110
140 INPUT "B=";B
150 IF B=INT(B) THEN IF B>=0 THEN IF B<=255 THEN POKE 49170,B : GOTO 170
160 GOTO 140
170 PRINT "♥"
180 SYS 49250
190 PRINT "SQQQQQQQFOLYTASSUK? I/N"
200 GET A$ : IF A$="I" THEN 110
210 IF A$<>"N" THEN 200
220 END
```

Most már nyugodtan kikérdezhetjük a gyereket, bár a programunkon még van mit finomítani (erről is szól a következő fejezet).

BINÁRISAN KÓDOLT DECIMÁLIS

A cím a számok tárolásának egy módját jelenti, amely adott esetben megkönnyítheti a dolgunkat. Vegyünk példának egy tízes számrendszerbeli számot, mondjuk a 2190-et! Kettes számrendszerben ennek b 100010001110, tizenhatos számrendszerben pedig $88E felel meg. De mi van akkor, ha nem akarjuk átalakítani se kettes, se tizenhatos számrendszerbe, és mégis szeretnénk valahogyan tárolni? Egyszerű! Tároljuk számjegyenként, mindegyik számjegyet egy külön bájton! A példánál maradva ehhez 4 bájtra lesz szükségünk. Ez a módszer gyönyörű, csak van egy hibája: irtózatosan pazarolja a memóriát. Ugyanis egy tízes számrendszerbeli számjegy kódolásához nincs szükség nyolc bitre, hiszen már 4 biten is több mint tízféle jel elfér. A tíz lehetséges számjegy tárolására fél bájt bőven elég, egy bájton tehát két számjegyet is tudunk tárolni, ami már jóval gazdaságosabb.

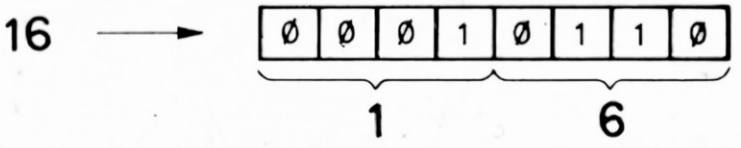


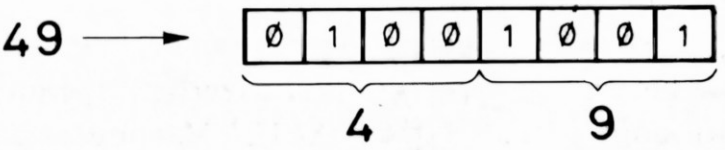


Ez az, amit a szaknyelv „binárisan kódolt decimális" néven emleget (angol rövidítése és neve: BCD = Binary Coded Decimal). A tízes számrendszerbeli számot tehát kódoljuk számjegyenként négy biten.

Ennek a kódolási módnak vannak előnyei. Igaz, hogy a gép kettes számrendszerben számol, de az adatok beírása és kijelzése a megszokott tízes számrendszerben történik. BCD alkalmazása esetén ilyenkor nem kell oda-vissza alakítgatni a számrendszerek között, s így rengeteg idő takarítható meg. Ezért ezt a számábrázolást főképp ott alkalmazzák, ahol nem kell sokat számolni. Kijelzés esetén sokkal könnyebb a dolgunk, mert csak

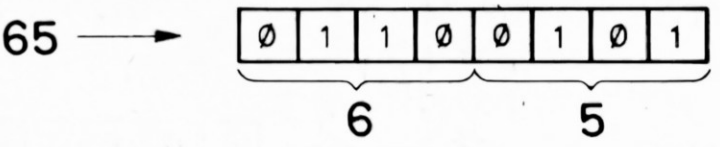
vennünk kell mindig a következő négy bitet, hogy meglegyen a következő kinyomtatandó számjegy. De megjegyzendő, hogy itt is van egy komoly probléma: binárisan kódolt dedimális számokat nem tudunk a hagyományos bináris módon összeadni.
Nézzünk egy példát! Kódoljuk a 16-ot és a 49-et binárisan kódolt decimálisban!





Ha ezeket tizenhatos számrendszerbeli számnak néznénk, akkor azt mondanánk rá, hogy $16, illetve $49.

De mi tudjuk, hogy ezek binárisan kódolt decimális számok, ezért azt is tudjuk, hogy az összegük 16+49=65, ami binárisan kódolva:



Erre persze mondhatjuk azt is, hogy $65. És itt jön a baj. Mert ha a két számot figyelmeztetés nélkül odaadja a vezérlő az ALU-nak, hogy adja őket össze, akkor az ALU a következő módon fog gondolkodni: $16+$49=$5F, és a $5F számot fogja visszaadni, amely még ráadásul nem is nevezhető binárisan kódolt decimális számnak. Vagyis az ALU önmagában nem tudja megkülönböztetni a binárisan kódolt decimális számokat a bináris vagy hexadecimális számoktól, ha nem tájékoztatjuk időben.
Szerencsére van megoldás, mert az ALU-t értesíteni lehet a turpisságról az egybájtos, burkolt címzésű **SED** utasítással, ami tulajdonképpen csak a D jelzőbitet állítja 1-re. Ha a D jelzőbit ég, akkor az ALU binárisan kódolt decimálisban számol, ha pedig nem ég, akkor a hagyományos bináris módon. Innen ered a D jelzőbit neve is, a Decimal mode flag (decimális mód jelzőbitje), ennek rövidítése a D betű.
Ha tehát a két szám összeadása előtt végrehajtunk egy SED utasítást, akkor az ALU az eredményt is binárisan kódolt decimálisban adja meg, $16+$49=$65 lesz.
A SED utasítástól kezdve az ALU decimális módon számol, majd a CLD utasítástól kezdve (ami a D-t nullázza) újból binárisan, kettes számrendszerben.
A binárisan kódolt decimálisnak megvan az az előnye, hogy a bájt felső négy bitje tartalmazza a nagyobb helyiértékű decimális számjegyet, a bájt alsó 4 bitje pedig a kisebb helyiértékű decimális számjegyet. Így a bájtot pontosan a közepénél kettévágva, a számjegyek könnyen megvizsgálhatók és kijelezhetők a képernyőre. Nézzük meg egy példán még egyszer a binárisan kódolt decimális legfőbb jellemzőjét!

binárisban			binárisan kódolt decimálisban		
\$25		\$86	\$25		\$86
+\$25		−\$48	+\$25		−\$48
\$4A	és	\$3E	\$50	és	\$38

Az INC, INX, INY, DEC, DEX és DEY utasítások továbbra is binárisan értendők; a SED hatása csupán az ADC és SBC utasításokra vonatkozik!

Figyelem! Decimális módban a Z jelzőbit nem működik helyesen, így nem használható arra, hogy megnézzük, vajon az eredmény nulla lett-e, vagy sem!

Most pedig lássunk egy példaprogramot, mely két bájton binárisan tárolt számot alakít át három bájton binárisan kódolt decimálisban tárolt számmá, s ezt ki is jelzi a képernyőre. A program a könnyebb áttekinthetőség kedvéért a lehető legegyszerűbb, és így kissé primitív módszert alkalmazza a probléma megoldására. Elképzelhető, hogy bárki gyorsabb és rövidebb programot írjon ugyanerre a feladatra.
Programunk a már meglevő gyerekvizsgáztató program kiegészítője. A kijelző szubrutin végén szabadon hagyott NOP-okat kell csak átírni, és már futtatható is. A három NOP utasítás helyett a \$C05E címen helyezzünk el egy JSR \$C19D utasítást. A már meglevő betöltőnkben csak a 110., 150. és 250. sort kell javítani, valamint hozzáírni a további sorokat. Nézzük az új szubrutint! Az előző program a \$C199 címen végződött. Ehhez a rutinhoz azonban szükség van még három bájtra, így nem a \$C19A, hanem a \$C19D címen kezdjük el. A közben kihagyott három bájt szerepe a következő:

\$C19A — a binárisan kódolt decimális szám legfelső számjegye
\$C19B — a binárisan kódolt decimális szám második és harmadik számjegye
\$C19C — a binárisan kódolt decimális szám negyedik és ötödik számjegye

Először is törölnünk kell ezeket a rekeszeket, mert nem tudhatjuk, hogy mi volt bennük előzőleg:

```
cím     kód        utasítás
---     ---        --------
C19D   A9 00       LDA #$00
C19F   8D 9A C1    STA $C19A
C1A2   8D 9B C1    STA $C19B
C1A5   8D 9C C1    STA $C19C
```

Ezek után vesszük a hexadecimális szám felső bájtját, és annyiszor adunk hozzá a decimális eredményhez 256-ot, amennyi a felső bájt. Ezt legcélszerűbb ciklussal megvalósítani.

```
cím     kód        utasítás
---     ---        --------
C1A8   AE 14 C0    LDX $C014
```

Ha a felső bájt nulla, akkor nincs szükség arra, hogy akár csak egyszer is hozzáadjunk az eredményhez 256-ot, így átugorjuk ezt a részt.

```
cím     kód         utasítás
---     ---         --------
C1AB  F0 1E         BEQ $C1CB
```

Ha mégis hozzáadunk, akkor először tájékoztatjuk az ALU-t arról, hogy decimális módban fog számolni.

```
cím     kód         utasítás
---     ---         --------
C1AD  F8            SED
```

Már semmi nem akadályozhat meg abban, hogy hozzáadjunk az alsó bájthoz 56-ot.

```
cím     kód         utasítás
---     ---         --------
C1AE  18            CLC
C1AF  AD 9C C1      LDA $C19C
C1B2  69 56         ADC #$56
C1B4  8D 9C C1      STA $C19C
```

A középső bájthoz kettőt adunk hozzá, ez jelképezi a 200-at.

```
cím     kód         utasítás
---     ---         --------
C1B7  AD 9B C1      LDA $C19B
C1BA  69 02         ADC #$02
C1BC  8D 9B C1      STA $C19B
```

A legfelső bájthoz hozzá kell adni nullát, hogy az esetleges átvitel is megtörténjen. (Ha ugyanis semmilyen művelet nem történne, az átvitelbit semmihez nem adódna hozzá csak úgy magától, csak maradna, amilyen volt, ezzel esetleg később még galibát is okozna.)

```
cím     kód         utasítás
---     ---         --------
C1BF  AD 9A C1      LDA $C19A
C1C2  69 00         ADC #$00
C1C4  8D 9A C1      STA $C19A
```

Most törölhetjük a D jelzőbitet:

```
cím     kód         utasítás
---     ---         --------
C1C7  D8            CLD
```

Amennyiben még nem elégszer hajtottuk végre a ciklust, akkor visszamegyünk, és még egyszer végrehajtjuk.

```
cím     kód         utasítás
---     ---         --------
C1C8  CA            DEX
C1C9  D0 E2         BNE $C1AD
```

A hexadecimális szám alsó bájtjával is hasonlóan járunk el. Most azonban az X-ben levő számnak nem 256-szorosát, hanem csak egyszeresét adjuk hozzá az eredményhez (jobb híján ezt is ciklussal tesszük).

```
cím     kód         utasítás
---     ---         --------
C1CB  AE 15 C0      LDX $C015
C1CE  F0 1E         BEQ $C1EE
C1D0  F8            SED
C1D1  18            CLC
C1D2  AD 9C C1      LDA $C19C
C1D5  69 01         ADC #$01
C1D7  8D 9C C1      STA $C19C
C1DA  AD 9B C1      LDA $C19B
```

```
C1DD  69 00       ADC #$00
C1DF  8D 9B C1    STA $C19B
C1E2  AD 9A C1    LDA $C19A
C1E5  69 00       ADC #$00
C1E7  8D 9A C1    STA $C19A
C1EA  D8          CLD
C1EB  CA          DEX
C1EC  D0 E2       BNE $C1D0
```

Binárisan kódolt decimális számunk a helyén van. Mivel a félbájtokon csak számjegyek szerepelnek, úgy küldjük ki őket a képernyőre, hogy előbb mindegyikhez hozzáadunk 48-at. A számok ASCII-kódjai ugyanis 48-cal nagyobbak, mint maguk a számok. Először azonban ki kell írnunk az egyenlőségjelet:

```
cím    kód        utasítás
---    ---        --------
C1EE  A9 3D       LDA #$3D
C1F0  20 D2 FF    JSR $FFD2
```

Meg kell oldanunk az előjel kérdését is, de ez már ismert probléma:

```
cím    kód        utasítás
---    ---        --------
C1F3  AD 13 C0    LDA $C013
C1F6  D0 05       BNE $C1FD
C1F8  A9 2D       LDA #$2D
C1FA  4C FF C1    JMP $C1FF
C1FD  A9 20       LDA #$20
C1FF  20 D2 FF    JSR $FFD2
```

Küldjük ki a képernyőre először a legfelső számjegyet!

```
cím    kód        utasítás
---    ---        --------
C202  AD 9A C1    LDA $C19A
C205  18          CLC
C206  69 30       ADC #$30
C208  20 D2 FF    JSR $FFD2
```

Küldjük ki a következő bájt felső négy bitjén tárolt második számjegyet!

```
cím    kód        utasítás
---    ---        --------
C20B  AD 9B C1    LDA $C19B
C20E  4A          LSR
C20F  4A          LSR
C210  4A          LSR
C211  4A          LSR
C212  18          CLC
C213  69 30       ADC #$30
C215  20 D2 FF    JSR $FFD2
```

Küldjük ki az illető cím alsó négy bitjén tárolt számjegyet is!

```
cím    kód        utasítás
---    ---        --------
C218  AD 9B C1    LDA $C19B
C21B  29 0F       AND #$0F
C21D  18          CLC
C21E  69 30       ADC #$30
C220  20 D2 FF    JSR $FFD2
```

Végül küldjük ki az utolsó két számjegyet is:

```
cím    kód        utasítás
---    ---        --------
C223  AD 9C C1    LDA $C19C
C226  4A          LSR
C227  4A          LSR
C228  4A          LSR
```

```
C229  4A          LSR
C22A  18          CLC
C22B  69 30       ADC #$30
C22D  20 D2 FF    JSR $FFD2
C230  AD 9C C1    LDA $C19C
C233  29 0F       AND #$0F
C235  18          CLC
C236  69 30       ADC #$30
C238  20 D2 FF    JSR $FFD2
```

A szubrutin befejezheti a működést:

```
 cím     kód        utasítás
 ---     ---        --------
C23B  60           RTS
```

A kibővített vizsgáztató program már decimálisan is kijelzi az értékeket. Erről a betöltő lefuttatása után meg is lehet győződni:

```
100 S=0
110 FOR I=0 TO 571
120 READ A
130 S=S+A : POKE 49152+I,A
140 NEXT I
150 IF S<>72123 THEN PRINT "GEPELESI HIBA VOLT !"
160 DATA 48,49,50,51,52,53,54,55,56,57
170 DATA 65,66,67,68,69,70,0,0,0,0
180 DATA 0,0,173,19,192,208,5,169,45,76
190 DATA 34,192,169,32,32,210,255,169,36,32
200 DATA 210,255,173,20,192,74,74,74,74,168
210 DATA 185,0,192,32,210,255,173,20,192,41
220 DATA 15,168,185,0,192,32,210,255,173,21
230 DATA 192,74,74,74,74,168,185,0,192,32
240 DATA 210,255,173,21,192,41,15,168,185,0
250 DATA 192,32,210,255,32,157,193,96,169,0
260 DATA 141,20,192,173,17,192,141,21,192,169
270 DATA 1,141,19,192,169,147,32,210,255,169
280 DATA 32,32,210,255,32,210,255,32,210,255
290 DATA 32,210,255,32,210,255,169,65,32,210
300 DATA 255,169,61,32,210,255,169,36,32,210
310 DATA 255,32,68,192,169,13,32,210,255,173
320 DATA 18,192,141,21,192,169,32,32,210,255
330 DATA 32,210,255,32,210,255,32,210,255,32
340 DATA 210,255,169,66,32,210,255,169,61,32
350 DATA 210,255,169,36,32,210,255,32,68,192
360 DATA 169,13,32,210,255,173,17,192,24,109
370 DATA 18,192,141,21,192,169,0,141,20,192
380 DATA 144,3,238,20,192,169,65,32,210,255
390 DATA 169,43,32,210,255,169,66,32,210,255
400 DATA 169,61,32,210,255,32,22,192,169,13
410 DATA 32,210,255,169,1,141,19,192,173,17
420 DATA 192,56,237,18,192,141,21,192,169,0
430 DATA 141,20,192,176,32,206,20,192,169,0
440 DATA 141,19,192,173,20,192,73,255,141,20
450 DATA 192,173,21,192,73,255,141,21,192,238
460 DATA 21,192,208,3,238,20,192,169,65,32
470 DATA 210,255,169,45,32,210,255,169,66,32
480 DATA 210,255,169,61,32,210,255,32,22,192
490 DATA 169,13,32,210,255,169,1,141,19,192
500 DATA 169,0,141,16,192,141,20,192,141,21
510 DATA 192,160,8,78,18,192,144,19,173,17
520 DATA 192,24,109,21,192,141,21,192,173,16
530 DATA 192,109,20,192,141,20,192,14,17,192
540 DATA 46,16,192,136,208,223,169,65,32,210
550 DATA 255,169,42,32,210,255,169,66,32,210
560 DATA 255,169,61,32,210,255,32,22,192,96
570 DATA 0,0,0,169,0,141,154,193,141,155
```

```
580 DATA 193,141,156,193,174,20,192,240,30,248
590 DATA 24,173,156,193,105,86,141,156,193,173
600 DATA 155,193,105,2,141,155,193,173,154,193
610 DATA 105,0,141,154,193,216,202,208,226,174
620 DATA 21,192,240,30,248,24,173,156,193,105
630 DATA 1,141,156,193,173,155,193,105,0,141
640 DATA 155,193,173,154,193,105,0,141,154,193
650 DATA 216,202,208,226,169,61,32,210,255,173
660 DATA 19,192,208,5,169,45,76,255,193,169
670 DATA 32,32,210,255,173,154,193,24,105,48
680 DATA 32,210,255,173,155,193,74,74,74,74
690 DATA 24,105,48,32,210,255,173,155,193,41
700 DATA 15,24,105,48,32,210,255,173,156,193
710 DATA 74,74,74,74,24,105,48,32,210,255
720 DATA 173,156,193,41,15,24,105,48,32,210
730 DATA 255,96
```

A betöltő program lefuttatása után ugyanazzal a Basic programmal ellenőrizhetünk, mint az előbbi esetben (ld. 104. old.). Programunk most már majdnem mindent gépi kódban végez. A továbbiakban a billentyűzet olvasását is gépi kódból fogjuk megvalósítani. Így jutunk el végül első komoly, teljesen gépi kódú programunk megírásához. Előtte azonban nézzünk még meg néhány további utasítást, trükköt és címzési módot!

KIEGÉSZÍTÉS

Foglaljuk össze, amit eddig a jelzőbitekről tudunk! Tudjuk, hogy az N jelzőbit az eredmény kettes komplemens kódban vett előjelét jelzi. Ha az eredmény negatív (vagyis a legfelső bitje 1), akkor az N jelzőbit 1, egyébként pedig nulla. Tudjuk továbbá, hogy a Z jelzőbit mindig azt jelzi, hogy az eredmény nulla lett-e vagy nem. A C jelzőbitet összeadásnál és kivonásnál használjuk az átvitel kezelésére, a D jelzőbit kigyújtásával pedig utasítani tudjuk az ALU-t, hogy BCD számokkal számoljon. Mire használható a V jelzőbit? A V jelzőbit (overflow flag) a kettes komplemens kódban létrejövő túlcsordulást jelzi. Ismételjük át ehhez az összeadást a korábbiaknál kicsit alaposabban!

```
  $24   – kettes komplemensben:  36
+$43    – kettes komplemensben:  67
------                          ---
=$67    – kettes komplemensben: 103
```

Az eredmény most kettes komplemens kódban is jó: 36+67=103. Nézzünk egy összeadást negatív számokkal is:

```
  $F4   – kettes komplemensben:  –12
+$E2    – kettes komplemensben:  –30
------
=$1D6
```

Az összeadás eredményeként a C jelzőbit 1 lesz, az A-ba pedig $D6 kerül. Kettes komplemens kódban $D6 megfelel –42-nek. Az összeadás jó eredményt adott, mert –12 + (–30) = –42. Így, bár binárisan az eredmény csak a C jelzőbit figyelembevételével lehet helyes, kettes komplemens kódban most C nélkül is az. Nézzünk két további esetet:

$35	– kettes komplemensben:	53
+$66	– kettes komplemensben:	102
=$9B	– kettes komplemensben:	–101

Két pozitív szám összeadása negatív számot eredményezett. De ez még mindig nem minden.

$ 85	– kettes komplemensben:	–123
+$ AB	– kettes komplemensben:	– 85
=$ 130		

Az összeadás eredményeként kigyullad a C jelzőbit, A-ba pedig $30 kerül. Kettes komplemensben $30 megfelel 48-nak. Most tehát két negatív szám összeadása pozitív számot eredményezett. Mi lehet ennek az oka? Semmi más, mint hogy az eredmény nagyobb, mint ami kettes komplemens kódban ábrázolható. Ennek igen szemléletes elnevezése a túlcsordulás.
Ha a matematikai művelet eredménye kettes komplemens kódban is helyes eredményt ad, akkor a művelet elvégzése után a V jelzőbit nem gyullad ki. Ellenkező esetben a V jelzőbit kigyulladása jelzi, hogy az eredmény csak binárisan helyes, kettes komplemens kódban már nem. Ezért nevezzük a V-t túlcsordulásjelző bitnek. Ha tehát kettes komplemens kódú számokkal végzünk műveleteket, akkor utána mindig meg kell vizsgálnunk a V jelzőbitet. Nyugodtak csak akkor lehetünk, ha a V jelzőbit nulla.
Mivel a leírt jelenség csak összeadásnál és kivonásnál jelentkezhet, így a V jelzőbitet az ADC és az SBC utasítás befolyásolja, de közvetlenül is nullázhatjuk a CLV utasítással. Használjuk még a különleges feladatokra alkalmas BIT utasításnál is. A BIT utasítással elég ritkán dolgozunk: a memóriarekesz egyes bitjeinek a vizsgálatakor. Nézzük a kódkönyv $2C. oldalát!

A BIT utasítás még sokadszorra is nyakatekertnek tűnik. Mindezek ellenére adódhat olyan probléma, ahol szükség lehet rá. Van azonban egy olyan terület, ahol nagyon gyakran használjuk, ez a több lehetséges kezdeti értékkel indított rutinok esete. Nézzünk erre egy példát!
A számkijelző rutinnal általában a szám előjelét is illik kiíratni. Egy szám előjele pozitív vagy negatív lehet, de ezért még nem kell két különböző rutint írnunk. Segítségül hívjuk a BIT utasítást, és alkalmazunk egy jól bevált trükköt. Ha a számunk előjele pozitív, akkor először betöltjük A-ba a + karakter kódját.

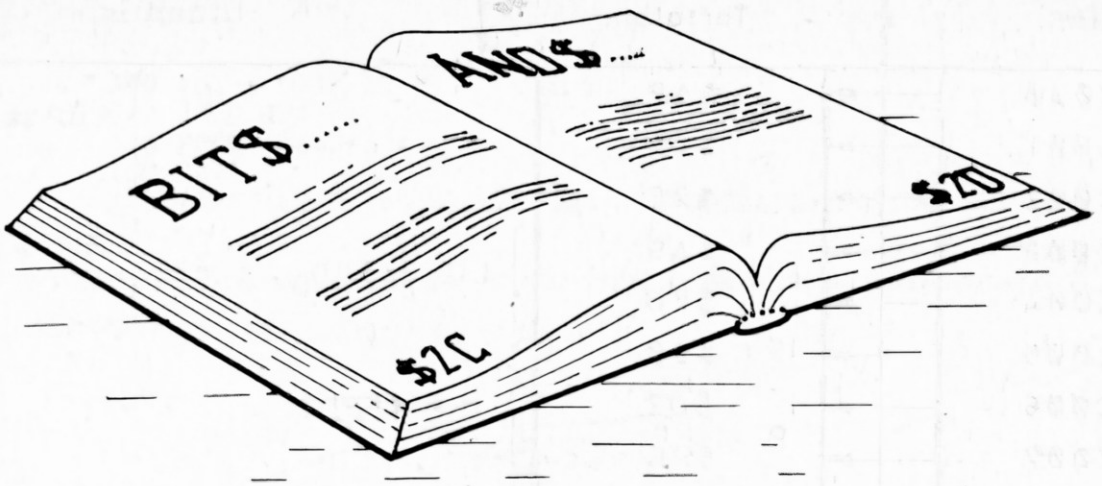
```
cím     kód        utasítás
---     ---        --------
C000  A9 2B        LDA #$2B
```

Hajtsunk most végre egy BIT utasítást!

```
cím     kód        utasítás
---     ---        --------
C002  2C A9 2D     BIT $2DA9
```

Ez semmit nem változtatott az A regiszter tartalmán, melyet így kijelezhetünk a képernyőre (a KERNAL CHROUT rutinját használjuk).

```
cím     kód        utasítás
---     ---        --------
C005  20 D2 FF     JSR $FFD2
```



Leírás

Hajtsunk végre egy logikai "ÉS" müveletet az A regiszter és a meghatározott memóriacim tartalmával, de az eredményt ne tegyük A-ba! Az A-ban maradjon továbbra is az, ami volt!

Jelzöbitek kezelése

1) Másoljuk át a cim által megadott memóriarekesz legfelső (128-at érö) bitjét az N jelzőbitbe!
2) Másoljuk át a cim á'tal megadott memóriarekesz második legfelső (64-et érö) bitjét a V jelzöbitbe!
3) Ha müveletünk eredménye zérus (Ø) lett, akkor gyújtsuk ki a Z jelzöbitet, egyébként pedig oltsuk el!

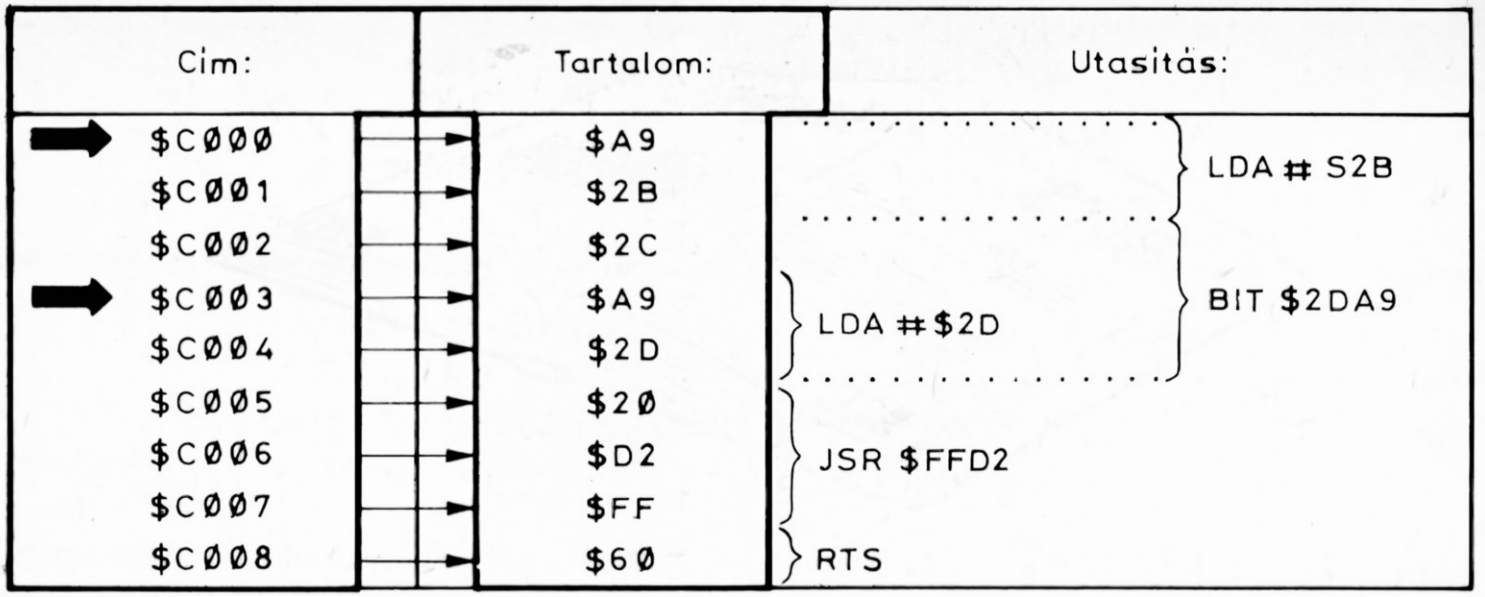
Végül az előjelet kijelző rutint lezárjuk egy RTS utasítással.

cím	kód	utasítás
---	---	--------
C008	60	RTS

Ez a rutin tehát kiírja a pozitív előjelet, és tartalmaz egy abszolút címzésű, abszolút felesleges BIT utasítást.
A program betöltője:

```
100 FOR I=0 TO 8
110 READ A
120 POKE 49152+I,A
130 NEXT I
140 DATA 169,43,44,169,45,32,210,255,96
```

Gépeljük be a betöltő lefuttatása után a rutint elindító SYS 49152 parancsot. A képernyőn megjelenik egy + karakter. Most próbálkozzunk meg egy SYS 49155 parancscsal is!
Ha idáig mindent helyesen csináltunk, akkor most egy — jel jelent meg a képernyőn. Mi volt ez? Varázslat? Dehogy, csak trükk. Nézzük meg, mit találunk a $C003 (49155) címen! $A9-et, ami a közvetlen címzésű LDA kódja. A $C004 címen $2D-t találunk, így ha a $C003 címtől összefüggően tekintjük a memóriát, úgy látjuk, hogy ott egy LDA #$2D utasítás áll. Ha tehát a BIT $D2A9 utasítás második bájtjától tekintjük a bájtsorozatot, ez egy LDA #$2D utasításnak felel meg.



Ha a $C000 címtől indítjuk a programot, akkor úgy érkezik el a $C005 címhez, hogy az A-ban $2B van. A $C003 címről indítva azonban úgy érkezik el a $C005 címhez, hogy az A-ban $2D van. Az ártalmatlan **BIT** utasítás közbeiktatásával így elértük, hogy programunk két különböző kezdeti értékkel is indítható úgy, hogy emiatt nem kellett két rutint megírni.

A Commodore—64 Basic rendszere is sok olyan rutint tartalmaz, amelyek több kezdeti értékkel futtathatók. Ezeket a rutinokat is abszolút címzésű BIT utasítások segítségével írták meg. Nagyon sokan és nagyon sokszor használják a most ismertetett trükköt. Ám amennyivel megkönnyíti a programozást, kétszer annyival nehezíti a program későbbi javítását, és tízszer annyival a visszafejtését. Bánjunk vele nagyon óvatosan!

De térjünk vissza a ciklusokhoz. Tudjuk, hogy ciklust szervezni indexelt címzéssel célszerű. Azonban az eddigi összes ciklusunkban a ciklusváltozó csökkent, és akkor lett vége a ciklusnak, amikor a ciklusváltozó (X vagy Y regiszter) elérte a nullát. Emlékezetünk felfrissítésére nézzünk egy példát, amely a képernyő középső sorába tesz kérdőjeleket.

```
 cím     kód        utasítás
 ---     ---        --------
C000   A2 28        LDX #$28
C002   A9 3F        LDA #$3F
C004   9D DF 05     STA $05DF,X
C007   CA           DEX
C008   D0 FA        BNE $C004
C00A   60           RTS
```

A betöltő:

```
100 FOR I=0 TO 10
110 READ A
120 POKE 49152+I,A
130 NEXT I
140 DATA 162,40,169,63,157,223,5,202,208,250,96
```

A program SYS 49152-re indul.

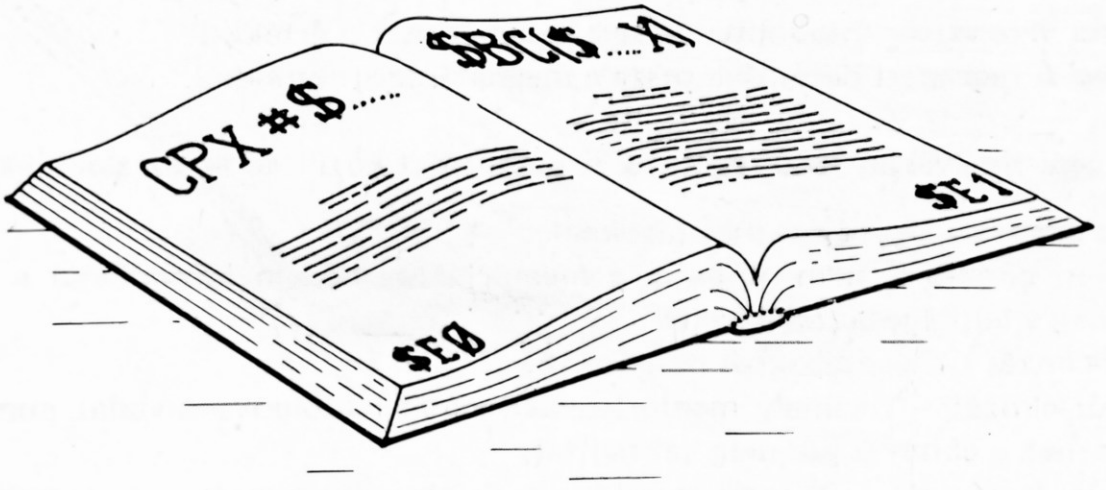
Nézzünk most egy másik megoldást, ahol a ciklusváltozó növekszik, és a program tartalmaz egy új utasítást is!

```
 cím     kód        utasítás
 ---     ---        --------
C000   A2 00        LDX #$00
C002   A9 3F        LDA #$3F
C004   9D E0 05     STA $05E0,X
C007   E8           INX
C008   E0 28        CPX #$28
C00A   D0 F8        BNE $C004
C00C   60           RTS
```

A betöltő:

```
100 FOR I=0 TO 12
110 READ A
120 POKE 49152+I,A
130 NEXT I
140 DATA 162,0,169,63,157,224,5,232,224,40,208,248,96
```

A program SYS 49152-re ugyanazt az eredményt produkálja, mint az előző. Vizsgáljuk meg a CPX utasítást!



Leírás

Hozassuk be a következő címről az ott lévő számot, majd vonjuk ki X+$0100-ból! Az eredményt jegyezzük le egy papírra, az X tartalma az utasítást követően maradjon az, ami előtte volt!

Jelzőbitek kezelése

1) Ha az eredmény nagyobb, mint $FF, akkor gyújtsuk ki a a C jelzőbitet, egyébként pedig oltsuk el!
2) Ha az eredmény alsó bájtja (alsó nyolc bitje) nulla, akkor gyújtsuk ki a Z jelzőbitet, egyébként oltsuk el!
3) Ha az eredmény alsó bájtjának legfelső (128-at érő) bitje 1, akkor gyújtsuk ki az N jelzőbitet, egyébként pedig oltsuk el!

A CPX a ComPare with X rövidítése (hasonlítsd össze X-szel). Ezzel az utasítással az X regiszter értékét hasonlítjuk össze egy adott számmal vagy egy adott memóriacím tartalmával, a címzési módtól függően. Az utasítás után, a leírt műveletek következtében a jelzőbitek a következőket jelzik:

A Z jelzőbit 1 lesz, ha az összehasonlított érték és az X tartalma megegyezik.

A C és a Z jelzőbit 0 lesz, ha az összehasonlított érték nagyobb, mint az X tartalma.

A C jelzőbit 1 lesz, a Z jelzőbit pedig 0, ha az összehasonlított érték kisebb, mint az X tartalma.

A CPX utasítással tehát azt állapíthatjuk meg, hogy az X regiszter tartalmánál nagyobb, kisebb vagy azzal egyenlő-e a címzési móddal meghatározott érték. Az összehasonlító utasítást legtöbbször arra szoktuk használni, hogy megnézzük, az X regiszter tartalma egyenlő-e valamely megadott értékkel. Ebben az esetben ugyanis (és csak ebben az esetben) a Z jelzőbit kigyullad. Ezt használtuk ki az előbbi ciklusunkban is. Egészen addig, amíg az X tartalma el nem érte a $28 értéket, a Z jelzőbit nem gyulladt ki (elaludt), így megtörtént a visszaágazás. Akkor azonban, amikor az X elérte a $28 értéket, a CPX #$28-nál kigyulladt a Z, így nem történt meg a visszaágazás, befejeződött a ciklus.
Összehasonlító utasítás a másik két regiszterhez is létezik:

CPY — Az Y regisztert hasonlítja össze a meghatározott értékkel.
CMP — Az A regisztert hasonlítja össze a meghatározott értékkel.

A C—64 gépi nyelvében lehetséges 13 címzési mód közül eddig az alábbiakat ismerjük:

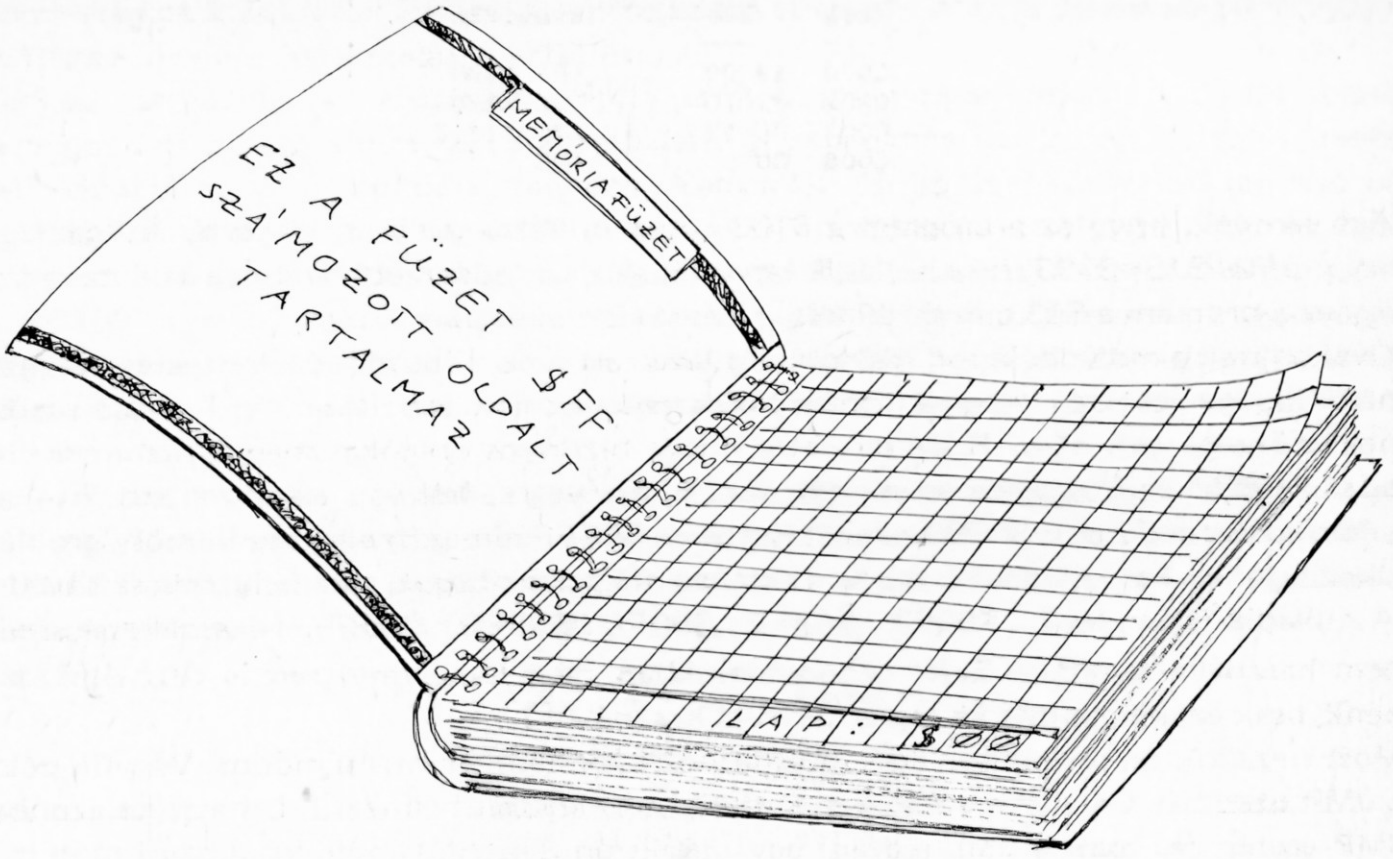
1. Burkolt címzés — egybájtos utasításoknál.
2. Közvetlen címzés — nem nyúlunk a memóriához, hanem közvetlenül a programból adjuk meg a feldolgozandó értéket.
3. Relatív címzés — elágazásoknál használjuk.
4. Abszolút címzés — valamely memóriacímen van a feldolgozandó adat, ennek a memóriarekesznek a címét adjuk meg két bájttal.
5. Abszolút, X címzés — X regiszterrel indexelt abszolút címzés — a megadott abszolút címhez hozzáadódik az X regiszter értéke is, így kapjuk meg a valódi címet.
6. Abszolút, Y címzés — Y regiszterrel indexelt abszolút címzés.

Nézzük most a további hét címzési módot is!
Különleges eset az, amikor a memóriafal nulladik (azaz legelső) oszlopában levő címekkel dolgozunk. Ezeknek a rekeszeknek a címe ugyanis mindig kisebb, mint 256, így az abszolút cím felső bájtja mindig nulla. Például:

cím	kód	utasítás
C000	AD 45 00	LDA $0045

Ha sokszor kezelünk ilyen nulladik oszlopban levő rekeszeket, akkor a címekben szokszor szerepel a 0 számjegy. Takarékossági okokból ezekre az esetekre kitaláltak egy új címzési módot, amelyet úgy hívnak, hogy nulladik lapos (nem oszlop, hanem lap) címzés. Azért hívják ezt a címzési módot nulladik lapos címzésnek, mert vannak, akik a memóriát úgy képzelik el, mint egy 256 oldalas füzetet. Szerintük a füzet minden oldalán van 256 darab rubrika. A füzet lapjait nullától számozzák $FF-ig, s az egyes lapokon levő rubrikákat is nullától $FF-ig jegyzik.
Az egyes rubrikák abszolút címe a lapszám és a rubrikaszám egymás utáni leírásával adódik. Azok a rubrikák tehát, amelyek ezen elmélet szerint a nulladik lapon vannak, megfelelnek a mi nulladik oszlopban levő rekeszeinknek.
A nulladik oszlopban levő rekeszek eléréséhez tehát megalkották a helytakarékos nulladik lapos címzést. Ennél a címzésnél a cím mindig csak egy bájt. Így maga a nulladik lapos utasítás csak két bájt memóriát foglal, szemben a három bájtos abszolút címzésű utasításokkal. Hasonlítsuk össze!

C000 8D 54 00 STA $0054



Ez egy abszolút címzésű STA utasítás, amely a $54 memóriarekeszre írja ki az A regiszter értékét. Az utasítás három bájtos.

cím	kód	utasítás
C000	85 54	STA $54

Ez pedig egy nulladik lapos címzésű STA utasítás, amely szintén a $54 címre írja ki az A regiszter értékét. Ez az utasítás azonban csak két bájtos.

Természetesen a két utasítás működése között semmilyen különbség nincs azon kívül, hogy a nulladik lapos címzésű változat hely- és időtakarékosabb. Ha tehát a nulladik oszlopban levő rekeszeket babráljuk, akkor használjuk bátran a nulladik lapos címzést! Nemcsak rövidebb, de gyorsabb is, és nem idegesíti fel a vezérlőt.

Persze ha az abszolút címzésnél lehetett indexelni, akkor kézenfekvőnek és logikusnak tűnik ugyanezt a nulladik lapos címzésnél is megtenni. Erre lehetőség is van. Nulladik lapos címzésnél is lehet indexelni, akár az X, akár az Y regiszterrel. Ezeket a címzéseket hívják nulladik lapos, indexelt címzéseknek. A következő programban például van egy X regiszterrel indexelt, nulladik lapos címzésű STA utasítás.

```
cím    kód       utasítás
---    ---       --------
C000   A9 00     LDA #$00
C002   A2 0F     LDX #$0F
C004   95 13     STA $13,X
C006   60        RTS
```

A program a $22 memóriarekesz tartalmát nullázza. Az X-be ugyanis bekerül $0F, így a STA $13,X utasítás a $13+$0F=$22 címre írja ki az A regiszter értékét. Vigyázzunk, indexelt nulladik lapos címzéssel nem léphetünk át laphatárt (oszlophatárt). Nézzük a következő programot!

```
 cím     kód        utasítás
 ---     ---        --------
C000   A9 00        LDA #$00
C002   A2 F0        LDX #$F0
C004   95 13        STA $13,X
C006   60           RTS
```

Azt várnánk, hogy ez a program a $103 címet nullázza, de nem ez történik. Igaz ugyan, hogy $F0+$13=$103, de a nulladik lapos jellegből adódóan ebből csak a $03 marad meg. Így ez a program a $03 cím tartalmát nullázza. Emlékezzünk erre!
Óvakodjunk a nulladik lapon (oszlopban) levő rekeszek túlbuzgó módosítgatásától, mert a nulladik lap (oszlop) rekeszei fontos rendszerváltozókat tartalmaznak. A Basic rendszernek szüksége van némi RAM-területre, hogy bizonyos dolgokat megjegyezhessen. Ilyen adat például az, hogy éppen melyik sort hajtja végre, hol van a kurzor stb. Ezeket az adatokat a rendszer a $0000 címtől kezdve a $03FF címig tárolja, így komoly problémát okozhatunk, ha a nulladik lapon vaktában megváltoztatjuk valamely rekesz tartalmát. A nulladik lapon levő $0002 $00FB $00FC $00FD $00FE rekeszeket a rendszer nem használja semmire. Saját programjainkban, ha a Basic rendszert is használni szeretnénk, csak ezeket a nulladik lapos címeket használjuk!
Most nézzünk néhány különleges, utasításhoz kapcsolódó címzési módot. Vegyük például a JMP utasítást, amelyről eddig azt tudtuk, hogy abszolút címzésű. Lehetséges azonban a JMP esetén (és csak a JMP esetén) egy másik, ún. abszolút indirekt címzési mód is. Az abszolút indirekt JMP kódja $6C, a következőképpen jelöljük:

```
 cím     kód        utasítás
 ---     ---        --------
C000   6C FC FF     JMP ($FFFC)
```

A program betöltője:

```
100 POKE 49152,108
110 POKE 49153,252
120 POKE 49154,255
```

A betöltő futtatása után begépelve a SYS 49152 parancsot, ugyanazt tapasztaljuk, amit a gép bekapcsolásakor vagy mint a SYS 64738 ($FCE2) utasítás végrehajtásakor. A JMP ($FFFC) tehát a következőt jelenti:

1. Vedd a $FFFC címen levő számot!
2. Írd elé a $FFFD címen levő számot!
3. Ugorj az így meghatározott címre!

Mivel a $FFFC és $FFFD rekeszek ROM-rekeszek, így bennük mindig ugyanazt, konkrétan $E2-t és $FC-t találunk. Az ugrási cím tehát $FCE2.

Az abszolút indirekt JMP utasításba nem a címet írjuk, hanem a cím címét: azt a memóriacímet, ahol a valódi ugrási cím megtalálható. Ezt a címzési módot csak a JMP utasításnál használhatjuk. Előnye, hogy használatával JUMP-vektorokat hozhatunk létre. Tegyük fel, hogy van egy programrészünk, amelyre nagyon sok helyről hivatkozunk. Ha ezt a programrészt később áthelyezzük más memóriaterületre, akkor az összes olyan JMP utasítást is át kell írnunk, amelyik erre a programrészre adja át a vezérlést. Ha azonban úgy írjuk meg a programot, hogy valahol két bájton tároljuk a programrész kezdőcímét, akkor mindenhol abszolút indirekt JMP utasítást használhatunk, amelyben ennek a kétbájtos címnek a címét kell megadnunk. Ha így helyezzük át a programrészt egy másik területre,

akkor elég csak ezt a két bájton tárolt mutatót átírni. Ezáltal az összes abszolút indirekt JMP is az új címre fogja átadni a vezérlést.
Nagyon vigyázzunk az abszolút indirekt JMP-ra, mert gyári hibás! A C–64 tervezői nem gondoltak egy esetre, és ezért hibásan gyártják még ma is. Az abszolút indirekt JMP laphatáron nem működik helyesen. Nem a lap utolsó és a következő lap első bájtjáról hozza be a címet (ahogyan logikus lenne), hanem a lap utolsó bájtjáról és ugyanazon lap első bájtjáról. Ha tehát azt írjuk, hogy JMP ($C0FF), akkor ez nem a $C0FF és $C100 címekről hozza be a címet, hanem a $C0FF és $C000 címekről. Igaz ugyan, hogy nem sokszor adódik olyan eset, amikor abszolút indirekt JMP utasításunkhoz a vektor éppen laphatárra kerül. Jó azonban tudni arról, hogy ilyenkor nem a várt módon működik.
Az abszolút indirekt címzéshez hasonló a nulladik lapos megoldásban is létezik, csak itt még indexelve is van. Ezt a címzést hívják indirekt indexelt címzésnek. Itt mindig egy nulladik lapos címet kell megadnunk, és mindig az Y regiszterrel indexelünk. Nézzünk egy példaprogramot a képernyő törlésére!

```
 cím    kód       utasítás
 ---    ---       --------
C000  A2 00       LDX #$00
C002  A0 04       LDY #$04
C004  86 FB       STX $FB
C006  84 FC       STY $FC
C008  A0 00       LDY #$00
C00A  A9 20       LDA #$20
C00C  91 FB       STA ($FB),Y
C00E  C8          INY
C00F  D0 FB       BNE $C00C
C011  E6 FC       INC $FC
C013  A5 FC       LDA $FC
C015  C9 08       CMP #$08
C017  D0 F1       BNE $C00A
C019  60          RTS
```

A betöltő:

```
100 FOR I=0 TO 25
110 READ A
120 POKE 49152+I,A
130 NEXT I
140 DATA 162,0,160,4,134,251,132,252,160,0,169,32
150 DATA 145,251,200,208,251,230,252,165,252
160 DATA 201,8,208,241,96
```

A program először elhelyezi a $FB és $FC címekre a $00 és $04 értékeket. Ezek után betölt Y-ba $00-át, X-be pedig $20-at. A $C00C címen levő indirekt indexelt címzésű STA hatására a következő történik:
Megnézi, hogy milyen címet tárol a $FB és $FC cím. Ott a $0400 címet találja. Ezek után hozzáadja ehhez az Y értékét, és az így kialakult címre kiírja az A regiszter tartalmát. Ezt a tevékenységet többször elvégzi egészen addig, amíg Y el nem éri a $FF értéket, majd amikor Y újból $00-vá válik, megnöveli a $FC tartalmát. Így a következő ciklusban már $0500-hoz adódik hozzá az Y értéke. Ha azonban a $FC tartalma eléri a $08-at, akkor leáll. Így aztán a $0400 címtől a $07FF címig mindenhová beírja a szóköz kódját.
Ez a címzési mód kissé bonyolult, de azért megérthető a működése. Jelölése: (Ind.),Y. Az indirekt indexelt címzést használjuk ugyan, de nem túl gyakran.
Van egy még nyakatekertebb címzési mód, amelyet szinte soha nem használunk. Ez az indexelt indirekt címzés. Jelölése: (Ind.,X). Itt a nulladik lapos címhez kell hozzáadni az

X regiszter értékét, majd az így létrejött címen és a rákövetkezőn található a valóban használt cím:

```
cím    kód       utasítás
---    ---       --------
C000   A2 00     LDX #$00
C002   A0 C8     LDY #$C8
C004   86 FB     STX $FB
C006   84 FC     STY $FC
C008   A2 7B     LDX #$7B
C00A   A9 20     LDA #$20
C00C   81 80     STA ($80,X)
C00E   60        RTS
```

A program hatására a $C800 címre kerül a $20. A $FB és $FC címekre $00-t és $C8-at teszünk. Ezek után betöltünk X-be $7B-t, A-ba $20-t, majd végrehajtjuk a STA ($80,X) utasítást. Először összeadódik $80 és az X értéke: $80+$7B=$FB. Ezek után megnézi a program, hogy mi van a $FB és az azt követő $FC címen. Ott $00-t és $C8-at talál, így a konkrét cím $C800 lesz.
A program betöltője:

```
100 FOR I=0 TO 14
110 READ A
120 POKE 49152+I,A
130 NEXT I
140 DATA 162,0,160,200,134,251,132,252,162,123
150 DATA 169,32,129,128,96
```

Futtassuk le a programot, majd helyezzünk el az 51200 címen nullát!

```
POKE 51200,0
READY.
SYS 49152
READY.
? PEEK (51200)
 32
READY.
■
```

Végül nézzük az utolsó címzési módot, az akkumulátoros címzést! Az akkumulátoros címzés az A regiszterről kapta a nevét, mivel az A regisztert szokták akkumulátornak is nevezni. Az akkumulátoros címzést csak az ASL, LSR, ROL és ROR utasításoknál használjuk.

```
cím       kód        utasítás
C000 0E 00 C8     ASL $C800
```

Ez egy abszolút címzésű ASL utasítás, a $C800 cím tartalmát csúsztatja el egy bittel balra.

```
cím       kód    utasítás
C000  0A         ASL
```

Ez egy akkumulátoros címzésű ASL utasítás, mely az A regiszter tartalmát egy bittel balra tolja.
Hasonlóan működik az akkumulátoros címzés az LSR, ROL és ROR utasításoknál is. Akkumulátoros címzésű utasításból összesen ez a négy darab létezik. Az akkumulátoros címzésű utasítások mindig egybájtosak, és az A regisztert csúsztatják vagy rotálják.

A most megismert új címzési módok:

7. Akkumulátoros címzés.
8. Nulladik lapos címzés.
9. Nulladik,X címzés — X regiszterrel indexelt nulladik lapos címzés.
10. Nulladik,Y címzés — Y regiszterrel indexelt nulladik lapos címzés.
11. (Ind.),Y címzés — Indirekt indexelt címzés.
12. (Ind.,X) címzés — Indexelt indirekt címzés.
13. Abszolút indirekt címzés — csak JMP utasításnál.

A tárban, ahol az egész programunk utasításkódokkal, címekkel, adatokkal együtt egyetlen számsorozat formájában jelenik meg, az értelmezés így zajlik:

— a programot elindító, vagy a szubrutint hívó utasításban megadott címen mindig utasításkód áll;
— minden utasításkódhoz egyetlen címzési mód tartozik, a függelékben közölt táblázat szerint;
— ez pedig egyértelműen eldönti, hogy az utasításhoz még hány bájt tartozik, ezeket hogyan kell értelmezni, és hol van a legközelebbi következő utasításkód.

Most, hogy már minden címzési módot és majdnem minden lehetséges gépi kódú utasítást ismerünk, finomítsuk oktatást segítő programunkat! Írjuk meg az adatbeviteli részt is gépi kódban! Ehhez csak annyit kell tudni, hogy a rendszer tartalmaz egy szubrutint a $FFE4 címen, amely behozza a billentyűzeten leütött karakter kódját. Ez a KERNAL GETIN rutin. Egyezzünk meg továbbá, hogy az egyik számot a $C23C címen, a másikat a $C23D címen fogjuk tárolni. Programunkat közvetlenül az eddig megírt részek után írjuk, kihagyva két bájtot a két számnak.

$C23C cím — egyik szám (A).
$C23D cím — másik szám (B).

Először legyen mind a két szám nulla!

```
cím     kód        utasítás
---     ---        --------
C23E  A9 00        LDA #$00
C240  8D 3C C2     STA $C23C
C243  8D 3D C2     STA $C23D
```

Közöljük az eddig megírt programrészekkel a számokat!

```
cím     kód        utasítás
---     ---        --------
C246  AE 3C C2     LDX $C23C
C249  AC 3D C2     LDY $C23D
C24C  8E 11 C0     STX $C011
C24F  8C 12 C0     STY $C012
```

Most meghívhatjuk az eddig megírt programrészt, amely számol és kijelez.

```
cím     kód        utasítás
---     ---        --------
C252  20 62 C0     JSR $C062
```

Először az első szám megváltoztatására adunk lehetőséget. Ezt kijelezzük a képernyőre is, teszünk egy nyilat az A szám elé.

```
cím     kód         utasítás
---     ---         --------
C255  A9 1F       LDA #$1F
C257  8D 03 04    STA $0403
```

Most behozunk a billentyűzetről egy karaktert a KERNAL GETIN rutinnal.

```
cím     kód         utasítás
---     ---         --------
C25A  20 E4 FF    JSR $FFE4
```

Ha a behozott karakter RETURN, akkor elágazunk arra a programrészre, amely a másik szám (B) változtatására ad lehetőséget:

```
cím     kód         utasítás
---     ---         --------
C25D  C9 0D       CMP #$0D
C25F  F0 2A       BEQ $C28B
```

Ha a behozott karakter nem SPACE (szóköz), akkor folytatódhat a program.

```
cím     kód         utasítás
---     ---         --------
C261  C9 20       CMP #$20
C263  D0 03       BNE $C268
```

Ha a SPACE billentyűt ütöttük le, akkor „megöljük" a programot. Programunkat a SPACE billentyű leütésével lehet megszüntetni.

```
cím     kód         utasítás
---     ---         --------
C265  4C E2 FC    JMP $FCE2
```

Most lefuttatunk egy ciklust, mely a behozott karaktert összehasonlítja a memóriában már tárolt hexadecimális számjegykódokkal:

```
cím     kód         utasítás
---     ---         --------
C268  A2 0F       LDX #$0F
C26A  DD 00 C0    CMP $C000,X
```

Ha a karakter egyezik valamelyik tárolt karakterrel, akkor elfogadjuk, folytatódik a program a $C275 címen.

```
cím     kód         utasítás
---     ---         --------
C26D  F0 06       BEQ $C275
```

Ha azonban nem egyezik meg, és még nem vizsgáltunk meg minden kódot, akkor még egyszer lefut a ciklus.

```
cím     kód         utasítás
---     ---         --------
C26F  CA          DEX
C270  10 F8       BPL $C26A
```

Ha minden karaktert megvizsgáltunk, de nem találtunk egyezést, akkor a behozott karakter nem hexadecimális számjegy volt, hozhatunk újból egy karaktert a billentyűzetről. Így érhetjük el, hogy a gépünk egyszerűen nem is reagál azokra a billentyűkre, amelyek az adott szituációban logikailag nem illenek oda. Ha beírtuk a programot, próbáljuk csak ki: még a STOP billentyűre sem reagál, mondván, hogy az nem lehet hexadecimális szám része.

```
cím     kód         utasítás
---     ---         --------
C272  4C 5A C2    JMP $C25A
```

Ha elfogadható karakterkód érkezett, akkor a neki megfelelő számot betöltjük az A regiszterbe.

```
cím     kód        utasítás
---     ---        --------
C275  8A           TXA
```

Az "A" szám első számjegyét „kitoljuk" balra, közben a második számjegyét eltoljuk négy bittel balra, így az lesz az első számjegy.

```
cím     kód        utasítás
---     ---        --------
C276  0E 3C C2     ASL $C23C
C279  0E 3C C2     ASL $C23C
C27C  0E 3C C2     ASL $C23C
C27F  0E 3C C2     ASL $C23C
```

A behozott új számjegyet elhelyezzük az alsó négy bitre, ez lesz a második számjegy. A számba tehát minden új számjegy jobbról jön majd be, miközben a bal oldali számjegy kimegy.

```
cím     kód        utasítás
---     ---        --------
C282  0D 3C C2     ORA $C23C
C285  8D 3C C2     STA $C23C
```

Kezdjük újból a programot a kijelző rutin meghívásával.

```
cím     kód        utasítás
---     ---        --------
C288  4C 46 C2     JMP $C246
```

A következő programrész, az értelemszerű különbségektől eltekintve, teljesen megegyezik az előzővel. Ezzel a programrésszel kezeljük a második (B) szám módosítását.

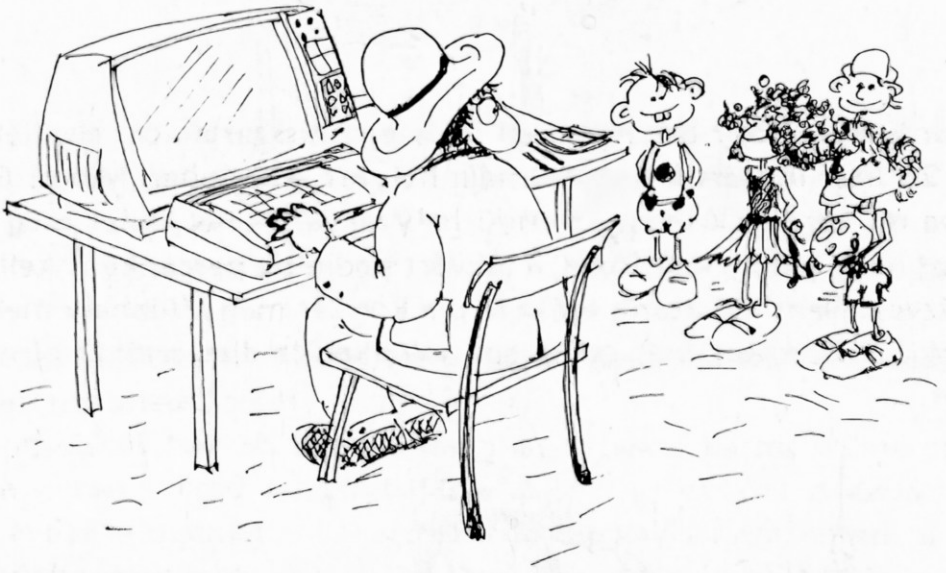
```
cím     kód        utasítás
---     ---        --------
C28B  AE 3C C2     LDX $C23C
C28E  AC 3D C2     LDY $C23D
C291  8E 11 C0     STX $C011
C294  8C 12 C0     STY $C012
C297  20 62 C0     JSR $C062
C29A  A9 1F        LDA #$1F
C29C  8D 2B 04     STA $042B
C29F  20 E4 FF     JSR $FFE4
C2A2  C9 0D        CMP #$0D
C2A4  F0 A0        BEQ $C246
C2A6  C9 20        CMP #$20
C2A8  D0 03        BNE $C2AD
C2AA  4C E2 FC     JMP $FCE2
C2AD  A2 0F        LDX #$0F
C2AF  DD 00 C0     CMP $C000,X
C2B2  F0 06        BEQ $C2BA
C2B4  CA           DEX
C2B5  10 F8        BPL $C2AF
C2B7  4C 9F C2     JMP $C29F
C2BA  8A           TXA
C2BB  0E 3D C2     ASL $C23D
C2BE  0E 3D C2     ASL $C23D
C2C1  0E 3D C2     ASL $C23D
C2C4  0E 3D C2     ASL $C23D
C2C7  0D 3D C2     ORA $C23D
C2CA  8D 3D C2     STA $C23D
C2CD  4C 8B C2     JMP $C28B
```

A programunk már teljes egészében gépi kódú. És nem is akármilyen. Sok kezdő Basices is megnyalná a tíz ujját, ha ilyet tudna írni Basicben!

Gépi kódú programunkat egy SYS 49726 paranccsal futtathatjuk le, és gyerek legyen a talpán, akit még így sem tudunk kikérdezni.

```
100 S=0
110 FOR I=0 TO 719
120 READ A
130 S=S+A : POKE 49152+I,A
140 NEXT I
150 IF S<>89603 THEN PRINT "GEPELESI HIBA VOLT !"
160 DATA 48,49,50,51,52,53,54,55,56,57
170 DATA 65,66,67,68,69,70,0,0,0,0
180 DATA 0,0,173,19,192,208,5,169,45,76
190 DATA 34,192,169,32,32,210,255,169,36,32
200 DATA 210,255,173,20,192,74,74,74,74,168
210 DATA 185,0,192,32,210,255,173,20,192,41
220 DATA 15,168,185,0,192,32,210,255,173,21
230 DATA 192,74,74,74,74,168,185,0,192,32
240 DATA 210,255,173,21,192,41,15,168,185,0
250 DATA 192,32,210,255,32,157,193,96,169,0
260 DATA 141,20,192,173,17,192,141,21,192,169
270 DATA 1,141,19,192,169,147,32,210,255,169
280 DATA 32,32,210,255,32,210,255,32,210,255
290 DATA 32,210,255,32,210,255,169,65,32,210
300 DATA 255,169,61,32,210,255,169,36,32,210
310 DATA 255,32,68,192,169,13,32,210,255,173
320 DATA 18,192,141,21,192,169,32,32,210,255
330 DATA 32,210,255,32,210,255,32,210,255,32
340 DATA 210,255,169,66,32,210,255,169,61,32
350 DATA 210,255,169,36,32,210,255,32,68,192
360 DATA 169,13,32,210,255,173,17,192,24,109
370 DATA 18,192,141,21,192,169,0,141,20,192
380 DATA 144,3,238,20,192,169,65,32,210,255
390 DATA 169,43,32,210,255,169,66,32,210,255
400 DATA 169,61,32,210,255,32,22,192,169,13
410 DATA 32,210,255,169,1,141,19,192,173,17
420 DATA 192,56,237,18,192,141,21,192,169,0
430 DATA 141,20,192,176,32,206,20,192,169,0
440 DATA 141,19,192,173,20,192,73,255,141,20
450 DATA 192,173,21,192,73,255,141,21,192,238
460 DATA 21,192,208,3,238,20,192,169,65,32
470 DATA 210,255,169,45,32,210,255,169,66,32
480 DATA 210,255,169,61,32,210,255,32,22,192
490 DATA 169,13,32,210,255,169,1,141,19,192
500 DATA 169,0,141,16,192,141,20,192,141,21
510 DATA 192,160,8,78,18,192,144,19,173,17
520 DATA 192,24,109,21,192,141,21,192,173,16
530 DATA 192,109,20,192,141,20,192,14,17,192
540 DATA 46,16,192,136,208,223,169,65,32,210
550 DATA 255,169,42,32,210,255,169,66,32,210
560 DATA 255,169,61,32,210,255,32,22,192,96
570 DATA 0,0,0,169,0,141,154,193,141,155
580 DATA 193,141,156,193,174,20,192,240,30,248
590 DATA 24,173,156,193,105,86,141,156,193,173
600 DATA 155,193,105,2,141,155,193,173,154,193
610 DATA 105,0,141,154,193,216,202,208,226,174
620 DATA 21,192,240,30,248,24,173,156,193,105
630 DATA 1,141,156,193,173,155,193,105,0,141
640 DATA 155,193,173,154,193,105,0,141,154,193
650 DATA 216,202,208,226,169,61,32,210,255,173
660 DATA 19,192,208,5,169,45,76,255,193,169
670 DATA 32,32,210,255,173,154,193,24,105,48
680 DATA 32,210,255,173,155,193,74,74,74,74
690 DATA 24,105,48,32,210,255,173,155,193,41
700 DATA 15,24,105,48,32,210,255,173,156,193
710 DATA 74,74,74,74,24,105,48,32,210,255
720 DATA 173,156,193,41,15,24,105,48,32,210
730 DATA 255,96,0,0,169,0,141,60,194,141
740 DATA 61,194,174,60,194,172,61,194,142,17
750 DATA 192,140,18,192,32,98,192,169,31,141
760 DATA 3,4,32,228,255,201,13,240,42,201
```

```
770 DATA 32,208,3,76,226,252,162,15,221,0
780 DATA 192,240,6,202,16,248,76,90,194,138
790 DATA 14,60,194,14,60,194,14,60,194,14
800 DATA 60,194,13,60,194,141,60,194,76,70
810 DATA 194,174,60,194,172,61,194,142,17,192
820 DATA 140,18,192,32,98,192,169,31,141,43
830 DATA 4,32,228,255,201,13,240,160,201,32
840 DATA 208,3,76,226,252,162,15,221,0,192
850 DATA 240,6,202,16,248,76,159,194,138,14
860 DATA 61,194,14,61,194,14,61,194,14,61
870 DATA 194,13,61,194,141,61,194,76,139,194
```



A teljes gépi programot a memóriába betöltő Basic programot remélhetőleg mindenki csak a szörnyűlködés kedvéért tekinti meg, mondván: „Úristen, ezt mind be kellene gépelnem, ha nem a monitort használtam volna!"

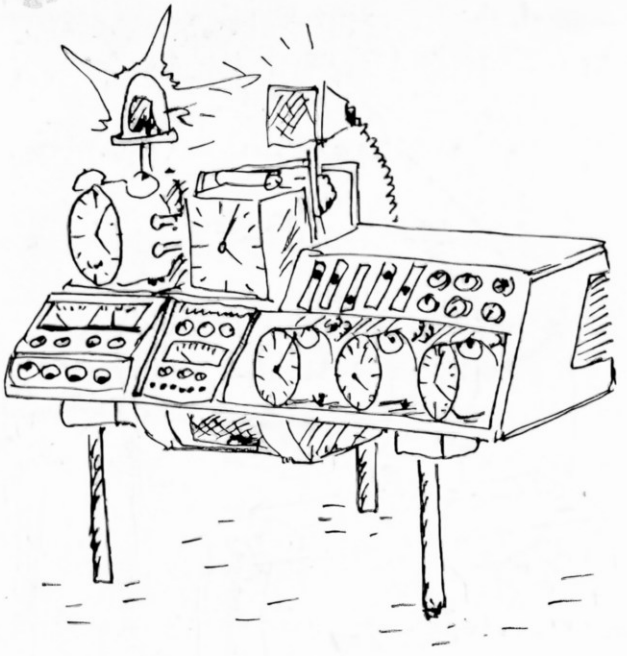
A MEGSZAKÍTÁS

Akárkivel előfordulhat, hogy bár meg kell írnia egy disszertációt, elvállalja, hogy megfőz 1000 tojást és 23 liter lekvárt a nagymamája helyett, aki éppen Miami Beachben nyaral. Mit tegyünk, ha mi kerülünk ilyen szörnyű helyzetbe? A helyzetet még az is bonyolítja, hogy a tojásokat négy percig kell főzni, a lekvárt pedig tíz percenként kell megkeverni (takaréklángon főzve). Nem állhatunk egész nap a kondér meg a tűzhely mellett, mert akkor nem lesz semmi a disszertációból, pedig egy jól sikerült disszertáció hírnevet, jó állást és sok pénzt jelent.

Furfangos, modern ember ilyen helyzetekben mindig segítségül hívja a technikát. Keressük fel elektromérnök barátunkat, és szerkesszünk egy sokcsatornás, ismétlő riasztókészüléket. Másnap már magabiztosan láthatunk neki az egyszerűvé vált feladatnak.

A sokcsatornás, ismétlő riasztószerkezet első csatornáját beállítjuk négyperces ismétlő-riasztó üzemmódra, a második csatornát pedig tízperces ismétlő-riasztó üzemmódra. A készülék kapcsolóját ON' állásba téve (azaz bekapcsolva) magára hagyjuk a lekvárt is meg a tojásokat is, és nekigyűrkőzünk a disszertációnak.

Nincs miért aggódnunk, 4 perc elteltével a készülék az egyes számú riasztódallammal jelzi a tojásriadót. Ekkor tojásokat cserélünk a vízben, és irány vissza a disszertációhoz. Újabb



4 perc múlva megismétlődik a riasztás, és mi egyre boldogabban ecseteljük magunkban ötletünk korszakalkotó jelentőségét.
A készülék bekapcsolását követő tizedik percben aztán a kettes számú csatorna is működésbe lép, és a kettes számú riasztódallammal jelzi — ezúttal a lekvárriadót. Irány a konyha, néhány evező mozdulat a fakanállal, és ismét visszatérhetünk a tudomány birodalmába. Az időzítés nemcsak egyszerűen zseniális, de még működik is, mint a karikacsapás.
Ha a készülékbe még egy füstérzékelőt is beépítünk, akkor már tűzvésztől sem kell tartanunk. Ha véletlenül meggyullad valami a konyhában, akkor a készülék rázendít egy harmadik dallamra. Ilyenkor, függetlenül attól, hogy éppen kell keverni vagy nem, először eloltjuk a tüzet, majd ha kell, keverünk. Baj nem történhet. Pár perc múlva már egy új disszertáción dolgozhatunk, amelynek a címe: *Többcsatornás ismétlő riasztókészülékek alkalmazási módszerei az élelmiszeriparban.*
Gondot csupán azok az esetek okoznak, amikor az egyes és a kettes számú riasztó dallam egyszerre szólal meg. Ám ez a probléma is inkább csak a nevében probléma, mert azt minden józan ember tudja, akinek van nagymamája, hogy a tojás várhat fél percet, de a lekvár nem, mert leég. Ilyenkor először a lekvárt keverjük meg, és utána térünk rá a tojásokra. Mire a nagymama lebarnulva hazatér Miami Beachből, mi már megírtuk a disszertációnkat, megfőztük a tojásokat és a lekvárt, de még a konyhát is kitakarítottuk, ahogyan egy tisztelettudó unokához illik.
Ilyesmi a számítástechnikában is létezik, megszakításkezelésnek hívják.
Ha például egy Basic program futtatása közben lenyomjuk a RUN/STOP billentyűt, akkor a program végrehajtása megszakad. Vagy ha beírjuk, hogy PRINT TI$, akkor a gép kiírja a bekapcsolás óta eltelt időt óra/perc/másodpercben. De honnan tudja a C—64, hogy le van-e nyomva a RUN/STOP billentyű, vagy hogy mennyi idő telt el a bekapcsolás óta? Onnan, hogy minden 1/60-ad másodpercben végrehajtja a megszakítási rutint. A megszakítási rutin olyan szubrutin, amely egyrészt felfüggeszti a Basic program futtatását, ha le van nyomva a RUN/STOP billentyű, másrészt pedig jegyzi az időt. Ezen kívül persze még rengeteg dolgot vizsgál és sok minden végez. Mindezt olyan rövid idő alatt, hogy észre sem vesszük, minden Basic programunk másodpercenként 60-szor megszakad.

A megszakítási rutin minden 1/60-ad másodpercben lefut, és megnöveli a TI változó értékét. A TI változót mi magunk is vizsgálhatjuk az alábbi rövid programmal:

```
100 PRINT CHR$(147)
110 PRINT CHR$(19);TI : GOTO 110
```

A TI$ nevű karakteres változó mindig annyi időt mutat, amennyi a TI változó alapján kiszámítható. A zökkenőmentes működéshez csak arra van szükség, hogy pontosan minden 1/60-ad másodpercben lefusson a megszakítási rutin. Ez persze nem egy szokványos szubrutin, nem programból hívja a JSR utasítás, hanem külső hatásra indul el a végrehajtása. Van a gépben egy időzítő, amely minden másodpercben hatvanszor kér egy megszakítást, és ilyenkor általában lefut a megszakítási rutin. A gépben történhet tehát bármi, futhat bármilyen program, ő minden 1/60-ad másodpercben kap egy megszakításkérelmet, amikor a következő történik:

A vezér először is befejezi az éppen végrehajtandó gépi utasítást, majd a veremben tárolja a következő végrehajtandó utasítás címét és a jelzőbiteket. Ezután elugrik egy különleges programrészre, amelyet megszakítási rutinnak hívnak. Itt elvégez a rendszer működéséhez fontos sok-sok műveletet. A végén, amikor találkozik egy RTI (ReTurn from Interrupt — visszatérés megszakításból) utasítással, kihúzza a veremből a jelzőbiteket, visszatér a megszakítási rutinból oda, ahonnan jött, és folytatja a programot.

Mikor a vezér elugrik a megszakítási rutinba, nemcsak a visszatérési címet, hanem a jelzőbitek állását is a verembe teszi. Ezért megszakításból nem lehet RTS utasítással visszatérni, hanem csak RTI utasítással. Ez olyan speciális visszatérést jelent, amelyik a jelzőbiteket visszaállítja a veremtartalom alapján. Amikor a megszakítási rutin a végén elér az RTI utasításhoz, először kihúzza a veremből a jelzőbiteket, és utána ugyanúgy jár el, mint az RTS utasítás esetében, vagyis úgy, mintha szubrutinból térne vissza.

A megszakítási kérelmek kezelése és a megszakítási rutinok futtatása tehát a programozó-

tól és az éppen futó programtól függetlenül történik. A programozó csak annyit tehet, hogy megtiltja a megszakítási kérelmek kiszolgálását egy SEI utasítással.

SEI — állítsd az I-t 1-re, vagyis ettől kezdve tagadd meg a megszakítási kérelmek kiszolgálását.
CLI — töröld az I-t, vagyis ezek után a beérkező megszakítási kérelmeket szolgáld is ki!

A dolgot képzeljük el úgy, hogy a vezérlő asztalán van egy sziréna, mellette egy IRQ felirattal. (Az IRQ az Interrupt ReQuest kifejezés rövidítése, magyarul megszakításkérés.) Ha megszólal a sziréna, akkor a vezér befejezi az éppen végrehajtás alatt levő utasítást, majd megnézi a nagy kódkönyv függelékét:

Mi a teendő, ha megszólal az IRQ sziréna?

Nézzük meg az I jelzőbitet! Ha az ég, akkor ne törődjünk semmivel, folytassuk tovább a programot a következő utasítással! Ha nem ég, akkor:

1) Tegyük a PC első két számjegyét a verembe!
2) Tegyük a PC utolsó két számjegyét a verembe!
3) Tegyük a jelzőbiteket a verembe!
4) Hozassuk be a $FFFE címen lévő számot, és írjuk le egy papírra!
5) Hozassuk be a $FFFF címen lévő számot, és írjuk az előbb behozott elé!
6) Állítsuk be a PC-t a cédulán lévő értékre!
7) Gyújtsuk ki az I jelzőbitet!
8) Folytassuk az utasítások végrehajtását a PC által meghatározott címtől!

Mivel a $FFFE és $FFFF címeken mindig ugyanaz az érték van (ez ROM—terület), ilyenkor mindig a $FF48 címen kezdődő IRQ rutin kezd el futni. Ez is ROM-ban van, így mindig ugyanaz.
Először a verembe kerülnek a regisztertartalmak.

```
 cím    kód       utasítás
 ---    ---       --------
FF48   48         PHA
FF49   8A         TXA
FF4A   48         PHA
FF4B   98         TYA
FF4C   48         PHA
```

Utána egy ügyes trükkel behozza a veremből a jelzőbiteket, amelyek közvetlenül a megszakítási rutin végrehajtása előtt lettek tárolva.

```
 cím    kód       utasítás
 ---    ---       --------
FF4D   BA         TSX
FF4E   BD 04 01   LDA $0104,X
```

Mindezek után megnézi, hogy égett-e a B jelzőbit:

```
cím     kód       utasítás
---     ---       --------
FF51  29 10       AND #$10
FF53  F0 03       BEQ $FF58
```

Ha égett a B jelzőbit, akkor végrehajt egy abszolút indirekt címzésű JMP utasítást a $0316 címen tárolt BRK vektor segítségével.

```
cím     kód       utasítás
---     ---       --------
FF55  6C 16 03    JMP ($0316)
```

Ha nem égett a B jelzőbit, akkor az abszolút indirekt JMP utasításhoz a $0314 címen levő IRQ vektort használja fel.

```
cím     kód       utasítás
---     ---       --------
FF58  6C 14 03    JMP ($0314)
```

Mivel a $0314, $0315, $0316 és $0317 címek RAM-memóriában vannak, ezeket a vektorokat megváltoztatva mi magunk is írhatunk majd saját elképzeléseink szerinti megszakítási rutinokat. Ezeket a vektorokat bekapcsoláskor a rendszer állítja be valamilyen értékre, de ezeket működés közben is módosíthatja.
Később, amikor a vezér elérkezik a megszakítási rutin végét jelző RTI utasításhoz, már a kódkönyvben keresgél.

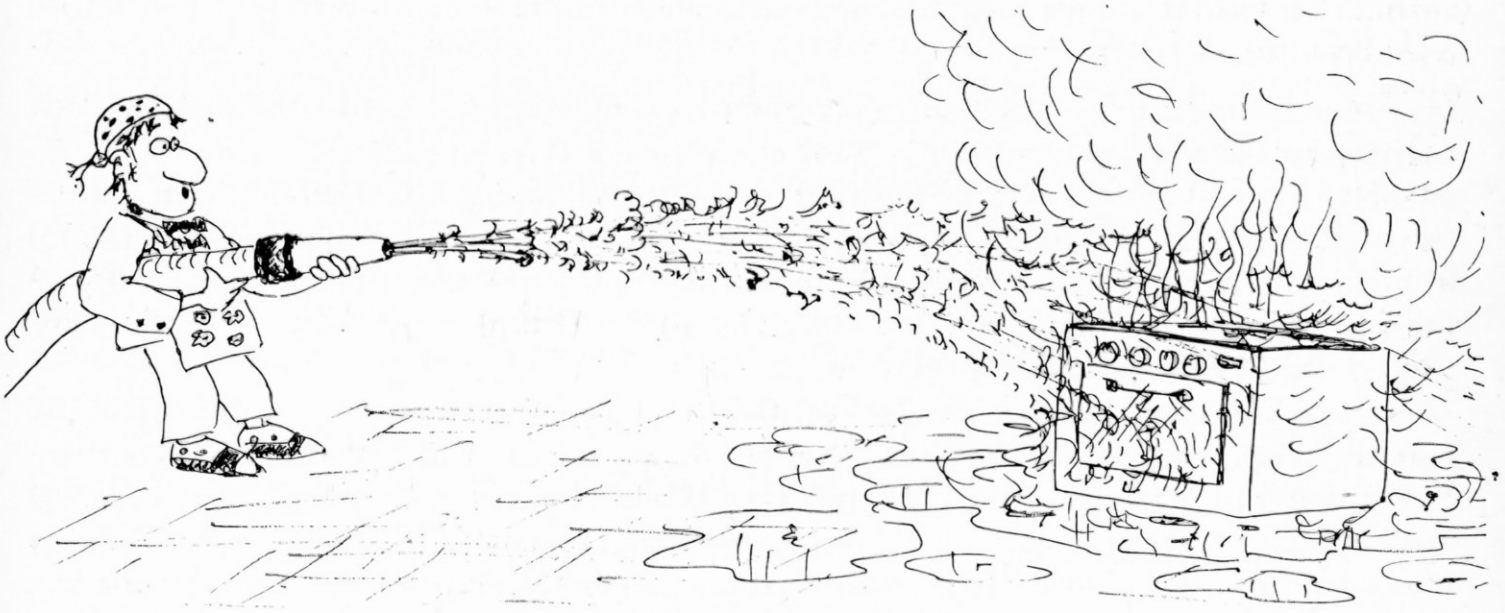
Leírás

1) Emeljünk ki a veremből egy számot, és állitsuk be annak megfelelően a jelzőbiteket!
2) Emeljünk ki a veremből két számot, és állitsuk be az igy meghatározott címre a PC-t!

Jelzőbitek kezelése

A jelzőbiteket már beállitottuk az 1) művelettel, igy azokkal többet foglalkozni nem kell.

A megszakításkezelés tehát abban is eltér a szubrutinhívástól, hogy megszakításkérelem kiszolgálásakor a soron következő utasítás címe kerül a verembe, szubrutinhíváskor pedig a következő utasítás első bájtja előtti cím.
Ha a számítógép már elkezdte egy megszakítás kiszolgálását, komoly problémákat okozhatna, ha még a megszakításból való visszatérés előtt kapna egy újabb megszakítási kérelmet. Ezért még a megszakítás kiszolgálása előtt 1-re állítódik az I jelzőbit, így a megszakítás kiszolgálása alatt újabb megszakítási kérelmet a vezérlő már nem szolgál ki. Visszatéréskor a veremből kihúzza a jelzőbiteket, így az I automatikusan újból nulla lesz. (Ha nem lett volna 1, létre sem jöhetett volna a megszakítás. Ez tehát a letiltható megszakítások működése, vagy ahogyan azt a komoly emberek nevezik, a maszkolható megszakítás.
Vannak azonban olyan megszakítási kérelmek is, amelyeknek a kiszolgálását nem lehet megtiltani. Gondoljunk vissza a befőzés esetére! Ha akkor jelez a füstjelző, amikor éppen lekvárt kellene kavarni, akkor mérlegelnünk kell, hogy melyik értékesebb. Egy edény lekvár vagy a konyha? Tehát, ha jelez a füstjelző, akkor bármit is csináltunk addig, azt abba kell hagyni, és el kell oltani a tüzet.



Hasonló okokból a vezér asztalán van egy NMI jelzésű sziréna is. Az NMI annak a Non Maskable Interrupt kifejezésnek a rövidítése, ami magyarul a nem maszkolható (nem tiltható) megszakítást jelenti. Vagyis vannak úgynevezett nem maszkolható megszakítások is. Ha érkezik egy nem maszkolható megszakítási kérelem, akkor akár tiltva vannak a megszakítások, akár nem, a vezér mindenképpen kiszolgálja a megszakítást, és csak a megszakítási rutinból való visszatérés után folytatja a programot. Így tehát elképzelhető az is, hogy éppen kiszolgál egy sima megszakítási kérelmet, amikor jelez az NMI sziréna. Ekkor a vezér befejezi az éppen megkezdett utasítást, kiszolgálja a nem maszkolható megszakítási kérelmet, majd ha onnan már visszatért, folytatja az eredeti megszakítási rutint.
Az NMI sziréna esete két dologban is különbözik az IRQ-tól. Az NMI esetén nem vizsgálja az I jelzőbitet, mindenképpen kiszolgálja a megszakítási kérelmet, ha törik, ha szakad. Abban is különbözik, hogy NMI megszakításkérés esetében más megszakítási rutint hajt végre a számítógép. NMI esetén ugyanis nem a $FFFE és $FFFF címekről hozza be a

kezdőcímet, hanem a $FFFA és $FFFB címekről. Mivel ezek a címek minden esetben a $FE43 értéket tárolják, így NMI esetén mindig a $FE43 címen kezdődő NMI rutin lesz végrehajtva.

```
cím     kód        utasítás
---     ---        --------
FE43  78           SEI
FE44  6C 18 03     JMP ($0318)
```

Itt tehát nem tárolja a regisztereket, és a B jelzőbitet sem vizsgálja.
Tisztáznunk kell még, hogy ki szólaltatja meg a szirénákat. A memóriafal másik oldalán szaladgáló alkalmazottak közül sokan nyomhatják meg a szirénákat megszólaltató gombokat. Van például két Complex Interface Adapter (CIA) nevű beosztott, az egyik az IRQ gombot, a másik az NMI gombot nyomhatja meg. Alapállapotban az egyes számú CIA minden 1/60-ad másodpercben megnyomja az IRQ gombot, ezáltal válik lehetővé az időváltozó (TI) megnövelése. Az is beprogramozható, hogy a CIA sűrűbben vagy esetleg ritkábban nyomogassa az IRQ gombot.
Az IRQ gombot megnyomhatja a VIC is. Utasítható a VIC, hogy nyomja meg az IRQ gombot, amikor például a képernyő közepénél tart. (Ezt szokták Raster Interruptnak nevezni.) És természetesen még sok egységet lehet utasítani, hogy bizonyos helyzetben nyomja le valamelyik gombot.

Megszakításhoz hasonló jelenséget a programozó is előidézhet a BRK utasítással. A BRK utasítás hatására ugyanaz történik, mint egy megszakításkérés hatására, csak még a B jelzőbit is kigyullad. A BRK megszakítás nem tiltható, mindig kiszolgálódik. Ekkor azonban a gép az IRQ rutint hajtja végre, ahol a B jelzőbit kigyulladása miatt a JMP ($0316) utasítás lesz végrehajtva. A BRK utasításnak hasonló a szerepe, mint Basicben a STOP utasításnak. Általában programjaink kipróbálásakor szoktunk beiktatni egy-egy BRK utasítást, hogy megvizsgálhassuk a részeredményeket.
Mire tudjuk használni a megszakításokat? Olyan feladatokra, amelyeket rendszeresen meg kell ismételni. Ilyen lehet például az, hogy a program futásától függetlenül, meghatározott időközönként megváltozzon a képernyőkeret színe. Legyen a meghatározott idő kb. egy másodperc! Ez megoldható lenne, ha kibővítenénk a meglevő megszakítási rutint egy olyan programrésszel, amely pl. minden hatvanadik megszakítás alkalmával megváltoztatja a keret színét. Hogy a képernyő keretének éppen milyen színe van, jegyezzük a $C803 címen. A $C802 címet használjuk számlálónak, ez a számláló jelzi, hogy hány megszakítás van még hátra a következő színváltásig. A $C800 és $C801 címeken legyen az eredeti megszakítási rutin címe!
Normál megszakításkor (IRQ) egy JMP ($0314) utasítás következtében adódik át a vezérlés a megszakítási rutinra, amely a $EA31 címen kezdődik. Mi ezt a rutint ki akarjuk bővíteni, amit úgy is megtehetünk, hogy megszabjuk: a JMP ($0314) hatására ne a $EA31 címre ugorjon a program, hanem, mondjuk a $C827 címre, és csak akkor ugorjon át a $EA31 címre, ha előtte már végrehajtott egy „adalék" rutint. Ezt a legkönnyebben úgy tudjuk megvalósítani, ha átírjuk a $0314 címen levő IRQ vektort. Mielőtt azonban a $C314 vektor tartalmát megváltoztatnánk, átírjuk a $C800-ba. A bővítésünk végén pedig egy JMP ($C800) utasítással adjuk át a vezérlést erre az eredeti kezdőcímre.
A kibővített megszakítást bekapcsoló program a következő:

```
cím     kód        utasítás
---     ---        --------
C804  AE 14 03     LDX $0314
C807  AC 15 03     LDY $0315
C80A  8E 00 C8     STX $C800
C80D  8C 01 C8     STY $C801
```

A számlálót beállítjuk $3C-re.

```
cím     kód         utasítás
---     ---         --------
C810  A9 3C         LDA #$3C
C812  8D 02 C8      STA $C802
```

A keret színét jegyző címre beírunk 15-öt.

```
cím     kód         utasítás
---     ---         --------
C815  A9 0F         LDA #$0F
C817  8D 03 C8      STA $C803
```

a $0314 címre pedig elhelyezzük a $C827 vektort.

```
cím     kód         utasítás
---     ---         --------
C81A  A2 27         LDX #$27
C81C  A0 C8         LDY #$C8
C81E  78            SEI
C81F  8E 14 03      STX $0314
C822  8C 15 03      STY $0315
C825  58            CLI
C826  60            RTS
```

Természetesen az átírás idejére le kell tiltani a megszakításokat, mert egy esetleges, közben létrejövő megszakítás tönkretehet mindent; utána viszont újból engedélyezni kell a megszakítások kiszolgálását (SEI, illetve CLI).

Ezek után megszakításkor a JMP ($0314) hatására mindig a $C827 címre adódik át a vezérlés, oda, ahol a programunk van. A programot egy JMP ($C800) utasítással fejezzük be, így minden megszakítás alkalmával nemcsak a mi programunkat hajtja végre a processzor, hanem utána az eredeti megszakítási rutint is. Természetesen, még mielőtt bekapcsolnánk az új megszakítási rendszert, el kell helyezni a memóriában a $C827 címtől kezdődően a bővítményt.

Mivel csak minden 60. megszakításkor kell valamit csinálni, először addig csökkentjük a számláló értékét, amíg el nem éri a nullát, egyébként pedig rögtön továbbmegyünk az eredeti megszakítási rutinra.

```
cím     kód         utasítás
---     ---         --------
C827  CE 02 C8      DEC $C802
C82A  D0 15         BNE $C841
```

Ha a számláló értéke elérte a nullát, akkor újból feltöltjük 60-nal, hogy csak újabb 60 megszakítás elteltével adódhasson ugyanilyen helyzet.

```
cím     kód         utasítás
---     ---         --------
C82C  A9 3C         LDA #$3C
C82E  8D 02 C8      STA $C802
```

Ezek után a keretszínt tartalmazó $C803 cím tartalmának megfelelően beállítjuk a keretszínt.

```
cím     kód         utasítás
---     ---         --------
C831  AD 03 C8      LDA $C803
C834  8D 20 D0      STA $D020
```

Természetesen a következő alkalommal már eggyel kisebb számmal fogunk dolgozni, így most csökkentjük a $C803 cím tartalmát.

```
cím     kód        utasítás
---     ---        --------
C837  CE 03 C8     DEC $C803
```

Igen ám, de ha az már a nulla volt, akkor nekünk nem $FF kell, hanem újból $0F. Tehát:

```
cím     kód        utasítás
---     ---        --------
C83A  10 05        BPL $C841
C83C  A9 0F        LDA #$0F
C83E  8D 03 C8     STA $C803
```

A vezérlő már rátérhet az eredeti megszakítási rutinra:

```
cím     kód        utasítás
---     ---        --------
C841  6C 00 C8     JMP ($C800)
```

Ez az a pont, amikor bekapcsolhatjuk a kibővített megszakítási rendszert, egy SYS 51204 utasítással ($C804=51204). A hatás lenyűgöző! A keret színe kb. másodpercenként változik, és közben mi bármit csinálhatunk, akár még Basic vagy gépi kódú programot is futtathatunk. Ahhoz, hogy megszüntessük az állandó villogást, és visszatérjünk az eredeti megszakítási rendszerre, megfelelő kikapcsoló programot kell írnunk:

```
cím     kód        utasítás
---     ---        --------
C844  AE 00 C8     LDX $C800
C847  AC 01 C8     LDY $C801
C84A  78           SEI
C84B  8E 14 03     STX $0314
C84E  8C 15 03     STY $0315
C851  58           CLI
C852  60           RTS
```

Ezt a kikapcsoló programot egy SYS 51270 utasítással hívhatjuk meg. A Basic nyelvű betöltő:

```
100 S=0
110 FOR I=0 TO 78
120 READ A
130 POKE 51204+I,A
140 S=S+A
150 NEXT I
160 IF S<>7982 THEN PRINT "GEPELESI HIBA VOLT !"
170 DATA 174,20,3,172,21,3,142,0,200,140
180 DATA 1,200,169,60,141,2,200,169,15,141
190 DATA 3,200,162,39,160,200,120,142,20,3
200 DATA 140,21,3,88,96,206,2,200,208,21
210 DATA 169,60,141,2,200,173,3,200,141,32
220 DATA 208,206,3,200,16,5,169,15,141,3
230 DATA 200,108,0,200,174,0,200,172,1,200
240 DATA 120,142,20,3,140,21,3,88,96
```

Következő bővítésünk is a keret színét változtatja, de nem folyamatosan, hanem az f1 billentyű lenyomására. Az f1 billentyű leütésével megváltoztathatjuk (akár egy program futása közben is) a keret színét. Ha azonban minden megszakítás alkalmával engedélyeznénk a keretszín változtatását, akkor nem tudnánk olyan gyorsan leütni és elengedni az f1 billentyűt, hogy a keret színe csak egyszer változzon! Éppen ezért a változtatás után csak 16 megszakítással (kb. 0,25 sec) váljon lehetővé az újabb változtatás.
A program a $C904 címtől kezdődik. A $C900 címen van az eredeti megszakítási vektor, a $C902 címen a keretszín, a $C903 címen pedig a számláló, amely a megszakításokat számolja. Nézzük meg a számlálót!

```
cím     kód         utasítás
---     ---         --------
C904   AD 03 C9     LDA $C903
```

Ha nulla, továbbmehetünk, mert szabad változtatni.

```
cím     kód         utasítás
---     ---         --------
C907   F0 06        BEQ $C90F
```

Ha még nem nulla, akkor csökkentjük eggyel.

```
cím     kód         utasítás
---     ---         --------
C909   CE 03 C9     DEC $C903
```

Elugrunk az eredeti megszakítási rutinra.

```
cím     kód         utasítás
---     ---         --------
C90C   6C 00 C9     JMP ($C900)
```

Ha szabad változtatni, akkor nézzük meg, hogy az f1 billentyű le van-e nyomva. (Ezt a programrészletet akkor értjük meg, ha megtekintjük a függelékben a billentyűzetről szóló részt.)

```
cím     kód         utasítás
---     ---         --------
C90F   A2 FF        LDX #$FF
C911   A0 00        LDY #$00
C913   8E 02 DC     STX $DC02
C916   8C 03 DC     STY $DC03
C919   A9 FE        LDA #$FE
C91B   8D 00 DC     STA $DC00
C91E   AD 01 DC     LDA $DC01
C921   29 10        AND #$10
```

Ha az f1 billentyű le van nyomva, akkor ezek után a Z értékének egynek kell lennie. Ebben az esetben kell csak továbbhaladni.

```
cím     kód         utasítás
---     ---         --------
C923   F0 03        BEQ $C928
```

Egyébként pedig ugorjunk az eredeti megszakítási rutinra!

```
cím     kód         utasítás
---     ---         --------
C925   6C 00 C9     JMP ($C900)
```

Ha az előbbi feltétel nem teljesült, vagyis a futást mégiscsak itt kell folytatnunk, akkor nézzük meg, hogy milyen a keret színe!

```
cím     kód         utasítás
---     ---         --------
C928   AD 02 C9     LDA $C902
```

Ha ez nem nulla, akkor csökkentsük eggyel, ha pedig nulla, akkor legyen 16, és azt csökkentsük eggyel, hogy 15 legyen!

```
cím     kód         utasítás
---     ---         --------
C92B   D0 05        BNE $C932
C92D   A9 10        LDA #$10
C92F   8D 02 C9     STA $C902
C932   CE 02 C9     DEC $C902
```

Ezt az értéket helyezzük el a keretszínt meghatározó mágikus fiókba!

```
cím     kód         utasítás
---     ---         --------
C935  AD 02 C9      LDA $C902
C938  8D 20 D0      STA $D020
```

A megszakításszámláló értékét a biztonság érdekében állítsuk be 16-ra:

```
cím     kód         utasítás
---     ---         --------
C93B  A9 10         LDA #$10
C93D  8D 03 C9      STA $C903
```

Végül ugorjunk el az eredeti megszakítási rutinra!

```
cím     kód         utasítás
---     ---         --------
C940  6C 00 C9      JMP ($C900)
```

A működéshez persze megint az kell, hogy egy bekapcsoló programmal kibővítsük az eredeti megszakítási rutint úgy, hogy utána már a mi rutinunkat is tartalmazza. Legyen tehát a számláló és a keretszín értéke is kezdetben nulla!

```
cím     kód         utasítás
---     ---         --------
C943  A9 00         LDA #$00
C945  8D 02 C9      STA $C902
C948  8D 03 C9      STA $C903
```

Másoljuk át az eredeti IRQ vektort a $C900 címre!

```
cím     kód         utasítás
---     ---         --------
C94B  AE 14 03      LDX $0314
C94E  AC 15 03      LDY $0315
C951  8E 00 C9      STX $C900
C954  8C 01 C9      STY $C901
```

Végül állítsuk be a $0314 címre a $C904 vektort, ami a bővítő rutinra mutat!

```
cím     kód         utasítás
---     ---         --------
C957  A2 04         LDX #$04
C959  A0 C9         LDY #$C9
C95B  78            SEI
C95C  8E 14 03      STX $0314
C95F  8C 15 03      STY $0315
C962  58            CLI
C963  60            RTS
```

A SYS 51523 hatására bekapcsolódik a kibővített megszakítási rutin, és lehetővé válik a keret színének változtatása még program futása alatt is.

Természetesen kikapcsolásra is módot kell adni. Ez a SYS 51556 utasítással érhető el, de előtte a memóriába kell tölteni az alábbi programot (ez csupán az eredeti megszakításvektort, az IRQ-t tölti vissza eredeti helyére):

```
cím     kód         utasítás
---     ---         --------
C964  AE 00 C9      LDX $C900
C967  AC 01 C9      LDY $C901
C96A  78            SEI
C96B  8E 14 03      STX $0314
C96E  8C 15 03      STY $0315
C971  58            CLI
C972  60            RTS
```

A program, valamint a ki- és bekapcsoló programok betöltője:

```
100 S=0
110 FOR I=0 TO 110
120 READ A
130 POKE 51460+I,A
140 S=S+A
150 NEXT I
160 IF S<>12034 THEN PRINT "GEPELESI HIBA VOLT !"
170 DATA 173,3,201,240,6,206,3,201,108,0
180 DATA 201,162,255,160,0,142,2,220,140,3
190 DATA 220,169,254,141,0,220,173,1,220,41
200 DATA 16,240,3,108,0,201,173,2,201,208
210 DATA 5,169,16,141,2,201,206,2,201,173
220 DATA 2,201,141,32,208,169,16,141,3,201
230 DATA 108,0,201,169,0,141,2,201,141,3
240 DATA 201,174,20,3,172,21,3,142,0,201
250 DATA 140,1,201,162,4,160,201,120,142,20
260 DATA 3,140,21,3,88,96,174,0,201,172
270 DATA 1,201,120,142,20,3,140,21,3,88
280 DATA 96
```

Mind a két közölt program betölthető bármely, a könyvben közölt program „mellé".
Ez a két példa is jól mutatja, hogy a megszakítás igen hasznos lehet olyan esetben, amikor rendszeresen kell bizonyos dolgokat elvégezni úgy, hogy ez a program futását ne zavarja. Ugyanezen sémára tetszőleges műveleteket végeztethetünk a géppel akár meghatározott időközönként, akár előre megadott időpont(ok)ban, akár külső hatásra, pl. meghatározott billentyű lenyomására, és közben más memóriaterületen zavartalanul futhatnak egyéb programjaink.
A profik persze még ennél virtuózabb dolgokat is tudnak művelni a megszakítással. Például olyat, hogy a képernyő kerete csíkos legyen. Ez azonban már a rendszer komolyabb ismeretét igényli, így itt nem ismertetjük. A gyakorlott programozó persze akkor is bűvöli a megszakításokat, ha erre nincs szüksége, hogy publikációit látva a szaktársak alázatos fővetéssel legyenek kénytelenek konstatálni szakmai fölényét. A megszakítások kezelésével tulajdonképpen mi is megtettük az első komolyabb lépést a profivá válás felé.

A KERNAL

A KERNAL egy szubrutingyűjtemény. Azokat a szubrutinokat tartalmazza, amelyeket a rendszer a leggyakrabban használ, és amelyekre a gépi kódban programozónak is sűrűn szüksége lehet.
Tegyük fel például, hogy a képernyőn a következő szabad képernyő-pozícióra ki akarunk írni egy Q betűt. Ehhez azt kell tudnunk, hogy hova került az utoljára kiírt karakter. Sőt, azt is meg kell vizsgálnunk, van-e még hely a képernyőn, azaz kell-e sort emelni. Így elég bonyolult programot írhatunk, amely a képernyővel kapcsolatos adatokat tartja nyilván, és segíti a képernyő kezelését. De miért írnánk egy ilyen programot, ha már létezik, és ráadásul szubrutin formájában? Mindössze annyit kell tennünk, hogy a nyomtatni kívánt karakter ASCII-kódját betöltjük az A-ba (LDA #$51), és meghívjuk a szubrutint egy JSR $FFD2 utasítással. Sokkal egyszerűbb és jobban is használható, mivel nemcsak a C–64 gépen futtatható. Hogy miért? Mert minden Commodore gép tartalmazza a KERNAL-t, és a KERNAL szubrutinjainak ugrótáblázata minden Commodore gépen ugyanott van: a $FF00–$FFFF területen, ami a C–64 esetében a memória utolsó lapja. Így tehát a

```
LDA #$51
JSR $FFD2
```

utasítássorozat minden Commodore gépen azt eredményezi, hogy a következő szabad képernyő-pozícióra kerül egy Q betű. Lehet persze, hogy a megfelelő programok a különböző gépeken mások, máshol vannak vagy másként működnek, de használatuk és hívásuk azonos. Az ugrótáblázat minden gépen ugyanott van, és az egyes KERNAL rutinok mutatóinak helye sem változik a táblázaton belül. Ezáltal olyan programokat írhatunk, amelyeket nemcsak egyfajta gépen lehet futtatni. Ha ugyanis a programunkban minden beviteli és kiviteli eszközzel a KERNAL segítségével beszélgetünk, akkor az egy másik Commodore gépen is futtatható lesz, és ráadásul még megírni is könnyebb a programot – csupán a megfelelő szubrutinok hívásával kell foglalkozni.
Egyes KERNAL rutinoknak szükségük van bemenő adatokra. Ezeket az adatokat a szubrutin meghívása előtt általában az A, X és Y regiszterekbe kell tölteni. Vannak olyan rutinok, amelyek adatokat adnak át visszatéréskor. Ezek az adatok is általában az A, X és Y regiszterekben találhatók. A KERNAL használatának egy másik előnye az, hogy a KERNAL a hibakeresést is elvégzi, és a keletkezett hibát visszatéréskor jelzi is.

A KERNAL RUTINOK

Minden KERNAL rutinnak van neve, hogy ne a hívási címmel kelljen rájuk hivatkozni, hanem egy olyan szóval, ami némileg a szerepükre is utal. Ezek a nevek általában a rutin működését angol nyelven leíró mondatok rövidítései.

CHKIN – $FFC6=65478
Megnyit egy csatornát adatbevitelre.

CHKOUT – $FFC9=65481
Megnyit egy csatornát adatkivitelre.

CHRIN – $FFCF=65487
Karakterbevitel egy csatornáról.

CHROUT – $FFD2=65490
Karakterkivitel egy csatornára.

CLALL – $FFE7=65511
Lezár minden csatornát és állományt.

CLOSE – $FFC3=65475
Lezár egy megadott logikai állományt.

CLRCHN – $FFCC=65484
Lezárja az I/O csatornákat.

GETIN – $FFE4=65508
Behoz egy karaktert a billentyűzetpufferből.

IOINIT – $FF84=65412
Inicializálja az I/O eszközöket.

LOAD – $FFD5=65493
Lemezről vagy kazettáról memóriába tölt.

OPEN – $FFC0=65472
Megnyit egy logikai állományt.

PLOT – $FFF0=65520
Beállítja vagy olvassa a kurzor pozícióját.

RDTIM – $FFDE=65502
Olvassa a valósidőórát.

READST – $FFB7=65463
Olvassa az I/O állapotát (hibakeresésre használható).

SAVE – $FFD8=65496
Memória kimentése lemezre vagy kazettára.

SCNKEY – $FF9F=65439
Vizsgálja a billentyűzetet.

SETLFS – $FFBA=65466
Beállítja az elsődleges és másodlagos logikai címeket.

SETNAM – $FFBD=65469
Beállítja az állomány nevét.

SETTIM – $FFDB=65499
Beállítja a valósidőórát.

STOP – $FFE1=65505
STOP billentyű vizsgálata.

UDTIM – $FFEA=65514
Növeli a valósidőóra értékét.

NÉV: **CHKIN**

RENDELTETÉS: Megnyit egy csatornát adatbevitelre.
HÍVÁSI CÍM: $FFC6=65478
ADATÁTVITELI REGISZTEREK: X
ÉRINTETT REGISZTEREK: A, X
ELŐKÉSZÍTŐ RUTINOK: (OPEN)
HIBALEHETŐSÉGEK: 0,3,5,6
LEÍRÁS: Ezzel a rutinnal bármely megnyitott logikai állományt adatbeviteli csatornaként definiálhatunk, ha előzőleg megnyitottuk a KERNAL OPEN rutinnal. Természetesen a használt egységnek beviteli egységnek kell lennie, másként ugyanis hiba lép fel, és a rutin megszakad. Ha nem a billentyűzetről kívánunk adatokat behozni, akkor ezt a rutint mindenképpen meg kell hívnunk, mielőtt a CHRIN vagy GETIN rutinokkal adatokat hozhatnánk be. Ha a billentyűzetről akarunk adatot behozni, és nincs egyéb adatbeviteli csatorna megnyitva, akkor sem ezt, sem az OPEN rutint nem szükséges meghívni.

HOGYAN HASZNÁLJUK:

1. Az OPEN rutin segítségével nyissuk meg az állományt (ha szükséges)!
2. Töltsük az X regiszterbe a használni kívánt logikai állomány számát!
3. Hívjuk meg a rutint (JSR utasítással)!

A lehetséges hibák:

#3: Az állomány nincs megnyitva (FILE NOT OPEN).
#5: Az egység nincs jelen (DEVICE NOT PRESENT).
#6: Nem beviteli állomány (FILE NOT AN INPUT FILE).

NÉV: **CHKOUT**

RENDELTETÉS: Megnyit egy csatornát adatkivitelre.
HÍVÁSI CÍM: $FFC9=65481
ADATÁTVITELI REGISZTEREK: X
ÉRINTETT REGISZTEREK: A, X
ELŐKÉSZÍTŐ RUTINOK: (OPEN)
HIBALEHETŐSÉGEK: 0,3,5,7 (lásd READST)
LEÍRÁS: A KERNAL OPEN rutinnal létrehozott bármelyik állomány adatkiviteli csatornaként definiálható. Természetesen az egységnek adatkiviteli egységnek kell lennie, egyébként hiba lép fel, és a rutin megszakad.
Ezt a rutint mindenféle adatkivitel előtt meg kell hívni, kivéve azt az esetet, amikor az egység a képernyő. (Ha a képernyőre akarunk adatot kivinni, és nincs egyéb adatkiviteli állomány megnyitva, akkor sem ezt a rutint, sem az OPEN rutint nem szükséges meghívni.)

Az adatokat a *Commodore 64 Programmer's Reference Guide* alapján közlöm.

HOGYAN HASZNÁLJUK:

1. Az OPEN rutin használatával hozzuk létre az állományt (ha szükséges)!
2. Töltsük az X regiszterbe a használt állomány számát!
3. Hívjuk meg a rutint (JSR utasítással)!

A lehetséges hibák:

#3: Az állomány nincs megnyitva (FILE NOT OPEN).
#5: Az egység nincs jelen (DEVICE NOT PRESENT).
#7: Nem kiviteli állomány (FILE NOT AN OUTPUT FILE).

NÉV: **CHRIN**

RENDELTETÉS: Behoz egy karaktert az adatbeviteli csatornáról.
HÍVÁSI CÍM: $FFCF=65487
ADATÁTVITELI REGISZTEREK: A
ÉRINTETT REGISZTEREK: A, X
ELŐKÉSZÍTŐ RUTINOK: (OPEN, CHKIN)
HIBALEHETŐSÉGEK: 0 (lásd READST)
LEÍRÁS: A KERNAL CHKIN rutin által adatbeviteli csatornaként definiált csatornáról hoz be egy bájtnyi adatot. Ha a CHKIN nem volt meghíva, akkor minden adatot a billentyűzetről vár. Az adatbájt az „A" regiszterbe kerül. A csatorna a hívás után megnyitva marad.
A billentyűzetről való bevitel módja speciális: először bekapcsolódik a kurzor, és egészen addig villog, míg egy RETURN karakter nem érkezik a billentyűzetről. A sorban mindegyik karakter (max. 88) a Basic input pufferbe kerül. Ezeket a karaktereket feldolgozni egyenként, ennek a rutinnak a hívásával lehet. A RETURN karakter behozatala után az egész sor fel van dolgozva. A rutin legközelebbi hívásakor az egész művelet elölről kezdődik a kurzor villogtatásával.

HOGYAN HASZNÁLJUK:

A BILLENTYŰZETRŐL:

1. Hozzunk be egy adatbájtot ezzel a rutinnal!
2. Tároljuk ezt a bájtot!
3. Nézzük meg, hogy nem az utolsó-e (RETURN)!
4. Ha nem, akkor kezdjük elölről!

MÁS EGYSÉGEKRŐL:

1. Hívjuk meg a KERNAL OPEN és CHKIN rutinokat!
2. Hívjuk meg ezt a rutint (JSR utasítással)!
3. Tároljuk vagy dolgozzuk fel az adatot!

NÉV: **CHROUT**

RENDELTETÉS: Kiküld egy karaktert.
HÍVÁSI CÍM: $FFD2=65490
ADATÁTVITELI REGISZTEREK: A
ÉRINTETT REGISZTEREK: A
ELŐKÉSZÍTŐ RUTINOK: (CHKOUT, OPEN)
HIBALEHETŐSÉGEK: 0 (lásd READST)
LEÍRÁS: Kiküld egy karaktert egy megnyitott csatornára. A rutin hívása előtt a KERNAL OPEN és CHKOUT rutinok hívásával létre kell hozni az adatkiviteli csatornát. Ha ez elmarad, akkor az adat a képernyőre kerül. Az adatbájtot az „A" regiszterbe kell tölteni, és utána kell a rutint meghívni. Ezek után az adat a kívánt egységhez kerül. A csatorna a hívás után nyitva marad.
A képernyő jobb alsó sarkába ezzel a rutinnal nem lehet soremelés nélkül írni. Oda csak a képernyő-memóriába való adattárolással írhatunk.

HOGYAN HASZNÁLJUK:

1. Hívjuk meg a KERNAL CHKOUT rutint (ha szükséges)!
2. A kiküldeni kívánt adatot töltsük az A regiszterbe!
3. Hívjuk meg a rutint!

NÉV: **CLALL**

RENDELTETÉS: Lezár minden állományt.
HÍVÁSI CÍM: $FFE7=65511
ADATÁTVITELI REGISZTEREK: nincsenek
ÉRINTETT REGISZTEREK: A, X
ELŐKÉSZÍTŐ RUTINOK: nincsenek
HIBALEHETŐSÉGEK: nincsenek
LEÍRÁS: Ez a rutin lezár minden állományt. A CLRCHN rutint is automatikusan meghívja, így visszaállítja az I/O csatornákat.

HOGYAN HASZNÁLJUK:

Hívjuk meg ezt a rutint!

NÉV: **CLOSE**

RENDELTETÉS: Lezár egy logikai állományt.
HÍVÁSI CÍM: $FFC3=65475
ADATÁTVITELI REGISZTEREK: A

ÉRINTETT REGISZTEREK: A, X, Y
ELŐKÉSZÍTŐ RUTINOK: nincsenek
HIBALEHETŐSÉGEK: 0, 240 (Lásd READST)
LEÍRÁS: Lezár egy logikai állományt, ha már nincs rá szükség. Az A regiszterbe kell tölteni a lezárni kívánt állomány számát, majd meghívni a rutint.

HOGYAN HASZNÁLJUK:

1. Töltsük az A regiszterbe a lezárni kívánt logikai állomány számát!
2. Hívjuk meg a rutint!

NÉV: **CLRCHN**

RENDELTETÉS: Törli az I/O csatornákat
HÍVÁSI CÍM: $FFCC=65484
ADATÁTVITELI REGISZTEREK: nincsenek
ÉRINTETT REGISZTEREK: A, X
ELŐKÉSZÍTŐ RUTINOK: nincsenek
HIBALEHETŐSÉGEK: nincsenek
LEÍRÁS: Törli és a kezdeti értékre állítja az I/O csatornákat. Erre akkor lehet szükség, — ha befejeztünk egy lemezes vagy kazettás műveletet, és vissza akarunk térni az eredeti állapothoz. A kezdeti adatbeviteli egység száma 0 (billentyűzet), a kezdeti adatkiviteli egység száma pedig a 3 (képernyő).
A rutin automatikus hívódik, ha meghívjuk a KERNAL CLALL rutint.

HOGYAN HASZNÁLJUK:

Hívjuk meg a rutint!

NÉV: **GETIN**

RENDELTETÉS: Behoz egy karaktert.
HÍVÁSI CÍM: $FFE4=65508
ADATÁTVITELI REGISZTEREK: A
ÉRINTETT REGISZTEREK: A, (X, Y)
ELŐKÉSZÍTŐ RUTINOK: CHKIN, OPEN
HIBALEHETŐSÉGEK: lásd READST
LEÍRÁS: Ha a csatorna a billentyűzet, akkor ez a rutin a billentyűzetpufferből vesz ki egy karaktert, és helyezi az A regiszterbe. Ha a billentyűzetpuffer üres, akkor az érték nulla lesz. A karakterek a pufferbe automatikusan kerülnek, mivel

a megszakítási rutin minden egyes megszakítás alkalmával meghívja a SCNKEY rutint. A billentyűzetpuffer maximum 10 karaktert tartalmazhat. Ha megtelt, akkor a továbbiakban érkező karaktereket egészen addig figyelmen kívül hagyja, amíg legalább egyet el nem távolítunk.

HOGYAN HASZNÁLJUK:

1. Hívjuk meg a rutint!
2. Nézzük meg, hogy nem nulla-e az A regiszter tartalma (üres puffer)!
3. Dolgozzuk fel az adatot.

NÉV: **IOINIT**

RENDELTETÉS: Inicializálja az I/O egységeket.
HÍVÁSI CÍM: $FF84=65412
ADATÁTVITELI REGISZTEREK: nincsenek
ÉRINTETT REGISZTEREK: A, X, Y
ELŐKÉSZÍTŐ RUTINOK: nincsenek
HIBALEHETŐSÉGEK: nincsenek
LEÍRÁS: Alapállapotba helyez minden I/O eszközt és rutint. Ha a gépen olyan programot futtatunk, amelyik cartridge-áramkörbe égetve áll rendelkezésünkre, és a működtetéséhez ezt az áramkört kell a gép hátuljához csatlakoztatnunk, az ebben levő program általában ezzel a szubrutinhívással kezdődik.

HOGYAN HASZNÁLJUK:

Hívjuk meg a rutint!

NÉV: **LOAD**

RENDELTETÉS: Memória betöltése.
HÍVÁSI CÍM: $FFD5=65493
ADATÁTVITELI REGISZTEREK: A, X, Y
ÉRINTETT REGISZTEREK: A, X, Y
ELŐKÉSZÍTŐ RUTINOK: SETLFS, SETNAM
HIBALEHETŐSÉGEK: 0,4,5,8,9,READST
LEÍRÁS: Adatbájtokat tölt be egyenesen a Commodore—64 memóriájába. Használható még összehasonlításra is (VERIFY), hogy meggyőződjünk róla, azonos-e a memória tartalma az egységen levő adatokkal. Természetesen ez utóbbi esetben a memória tartalma nem változik.
LOAD esetén hívás előtt az A regiszterbe töltsünk nullát (VERIFY esetén

1-et), mielőtt meghívjuk a rutint! Amennyiben az egység 0 másodlagos címmel lett megnyitva, a fejléc-információ figyelmen kívül marad. Ebben az esetben a LOAD kezdőcímét az X és Y regiszterekben kell tárolni. Ha az egység 1 másodlagos címmel lett megnyitva, akkor a betöltést a fejlécben meghatározott kezdőcímtől kezdi. Visszatéréskor az X és Y regiszterek a legmagasabb betöltött memóriacím címét adják.

HOGYAN HASZNÁLJUK:

1. Hívjuk meg a SETLFS és SETNAM rutinokat! Ha nem a fejléc szerinti kezdőcímtől kívánjuk betölteni, akkor a SETLFS rutinnál a másodlagos cím legyen 0!
2. Állítsuk be az A regiszter értékét 0-ra LOAD esetén, 1-re VERIFY esetén!
3. Ha nem a fejléc szerinti kezdőcímtől akarunk tölteni, akkor az X és Y regisztereket be kell állítani a kívánt kezdőcím értékre!
4. Hívjuk meg a rutint!

NÉV: **OPEN**

RENDELTETÉS: Megnyit egy logikai állományt.
HÍVÁSI CÍM: $FFC0=65472
ADATÁTVITELI REGISZTEREK: nincsenek
ÉRINTETT REGISZTEREK: A, X, Y
ELŐKÉSZÍTŐ RUTINOK: SETLFS, SETNAM
HIBALEHETŐSÉGEK: 1,2,4,5,6,240,READST
LEÍRÁS: Ezzel a rutinnal nyithatunk meg egy logikai állományt. Ha már létrehoztuk, a logikai állomány felhasználható adatbeviteli és adatkiviteli műveletekre.
A legtöbb KERNAL rutin is ezt a rutint hívja meg, hogy létrehozza az állományt, amellyel dolgozni akar. A rutin nem igényel bemenő adatokat, de a SETLFS és a SETNAM rutinokat előzőleg meg kell hívni.

HOGYAN HASZNÁLJUK:

1. Hívjuk meg a SETLFS rutint!
2. Hívjuk meg a SETNAM rutint!
3. Hívjuk meg az OPEN rutint!

NÉV: **PLOT**

RENDELTETÉS: Beállítja vagy kiolvassa a kurzorpozíciót.
HÍVÁSI CÍM: $FFF0=65520
ADATÁTVITELI REGISZTEREK: A, X, Y
ÉRINTETT REGISZTEREK: A, X, Y

ELŐKÉSZÍTŐ RUTINOK: nincsenek
HIBALEHETŐSÉGEK: nincsenek
LEÍRÁS: Amennyiben úgy hívjuk meg a rutint, hogy a C jelzőbit értéke 1, a kurzor pozícióját jelző értékek az X és Y regiszterekbe íródnak. Ha a C jelzőbit értéke nulla, az X és Y regiszterek által meghatározott helyre kerül a kurzor.
X lehetséges értékei: 0—24
Y lehetséges értékei: 0—39

HOGYAN HASZNÁLJUK:

A KURZOR POZÍCIÓJÁNAK OLVASÁSÁRA

1. Állítsuk 1-re a C jelzőbitet!
2. Hívjuk meg a rutint!
3. Az X regiszterben található a sor, az Y regiszterben az oszlop sorszáma.

A KURZOR POZÍCIÓJÁNAK BEÁLLÍTÁSÁRA

1. Állítsuk 0-ra a C jelzőbitet!
2. Állítsuk az X és Y regisztereket a kívánt értékre!
3. Hívjuk meg a rutint!

NÉV: **RDTIM**

RENDELTETÉS: Olvassa a rendszerórát.
HÍVÁSI CÍM: $FFDE=65502
ADATÁTVITELI REGISZTEREK: A, X, Y
ÉRINTETT REGISZTEREK: A, X, Y
ELŐKÉSZÍTŐ RUTINOK: nincsenek
HIBALEHETŐSÉGEK: nincsenek
LEÍRÁS: Ezzel a rutinnal olvasható a rendszeróra. Az óra felbontóképessége 1/60 másodperc. A rutin három bájt adattal tér vissza. Az Y regiszter tartalmazza a legmagasabb helyiértékű bájtot, X tartalmazza a középsőt, és A a legalacsonyabb bájtot. A bájtokat Y—X—A sorrendben összeolvasva tehát a bekapcsolás óta eltelt hatvanad-másodpercek számát kapjuk, kivéve, ha időközben a SETTIM szubrutinnal másképp állítottuk be az órát.

HOGYAN HASZNÁLJUK:

Hívjuk meg a rutint!

NÉV: **READST**

RENDELTETÉS: Olvassa az állapotjelző szót.
HÍVÁSI CÍM: $FFB7=65463
ADATÁTVITELI REGISZTEREK: A
ÉRINTETT REGISZTEREK: A
ELŐKÉSZÍTŐ RUTINOK: nincsenek
HIBALEHETŐSÉGEK: nincsenek
LEÍRÁS: Az I/O egységek jelenlegi állapotát jelzi az akkumulátorban; általában új I/O műveletek után hívjuk meg. A rutin az egység állapotáról ad információt, vagy a felmerült hibákról, mégpedig úgy, hogy az állapotkódot (hibakódot) az 'A regiszterbe teszi. (Ennek a kódnak az értéke azonos a Basic ST változójával.)

Az A regiszter egyes bitjei az alábbi információt tartalmazzák:

BITPOZÍCIÓ	NUM. ÉRTÉK	Jelentése		
		KAZETTA-OLVASÁS esetén	SOROS OLVASÁS ÍRÁS esetén	SZALAG VERIFY +LOAD esetén
0	1		Írásidőzítési hiba	
1	2		Olvasásidőzítési hiba	
2	4	Rövid blokk		Rövid blokk
3	8	Hosszú blokk		Hosszú blokk
4	16	Felismerhetetlen olvasás hibája		Bármilyen hiba
5	32	Az ellenőrző összeg hibás		Az ellenőrző összeg hibás
6	64	Állomány vége	EOI	
7	−128	Szalag vége	Az egység nincs jelen	A szalag vége

NÉV: **SAVE**

RENDELTETÉS: Memória mentése lemezre vagy kazettára.
HÍVÁSI CÍM: $FFD8=65496
ADATÁTVITELI REGISZTEREK: A, X, Y
ÉRINTETT REGISZTEREK: A, X, Y
ELŐKÉSZÍTŐ RUTINOK: SETLFS, SETNAM
HIBALEHETŐSÉGEK: 4,8,9,READST

LEÍRÁS: A memória egy részét menti ki, az A regiszterrel jelzett, nulladik lapos, indirekt címtől kezdődően, az X és Y regiszterekben tárolt címig. A SETLFS és SETNAM rutinokat meg kell hívni, mielőtt a rutint meghívnánk. A Datasette-re való mentésnél nem szükséges az állománynév (lásd a SETNAM leírásánál). Állománynév nélkül a lemezre mentés hibát eredményez.

HOGYAN HASZNÁLJUK:

1. Hívjuk meg a SETLFS és SETNAM rutinokat!
2. Két egymást követő nulladik lapos címre helyezzük el az első kimenteni kívánt bájt címét (az alacsonyabb címre az alsó, magasabb címre a felső bájtot)!
3. Töltsük az A regiszterbe ennek a nulladik lapos vektornak a címét!
4. Töltsük az X és Y regiszterekbe az utolsó kimenteni kívánt bájt címét (az X regiszterbe az alsó, az Y regiszterbe a felső bájtot)!
5. Hívjuk meg a rutint!

NÉV: **SCNKEY**

RENDELTETÉS: Vizsgálja a billentyűzetet.
HÍVÁSI CÍM: $FF9F=65439
ADATÁTVITELI REGISZTEREK: nincsenek
ÉRINTETT REGISZTEREK: A, X, Y
ELŐKÉSZÍTŐ RUTINOK: (IOINIT)
HIBALEHETŐSÉGEK: nincsenek
LEÍRÁS: Vizsgálja a billentyűzetet, lenyomott billentyűket keres. Ha egy billentyűt leütünk, és a billentyűzetpufferben van még hely, akkor a leütött billentyű ASCII-kódja a billentyűzetpufferbe kerül. Csak a normál IRQ megszakítási rutin hívja meg, alapállapotban másodpercenként kb. hatvanszor.

HOGYAN HASZNÁLJUK:

Hívjuk meg a rutint!

NÉV: **SETLFS**

RENDELTETÉS: Létrehoz egy logikai állományt.
HÍVÁSI CÍM: $FFBA=65466
ADATÁTVITELI REGISZTEREK: A, X, Y
ÉRINTETT REGISZTEREK: nincsenek
ELŐKÉSZÍTŐ RUTINOK: nincsenek
HIBALEHETŐSÉGEK: nincsenek

LEÍRÁS: Beállítja a logikai állomány számát, az egységszámot és a másodlagos címet (parancs) a többi KERNAL rutin számára.
A logikai állományszámot a rendszer az OPEN rutinok által megnyitott állományok táblázatában való eligazodásra használja. Az egységszám nullától 31-ig terjedhet. Az egyes CBM (Commodore Business Machines) egységek kódja:

kód	EGYSÉG
0	Billentyűzet
1	Datasette
2	RS–232C egység
3	CRT kijelző (képernyő)
4	Nyomtató a soros buszon
8	Lemezmegható egység a soros buszon

A 4-nél nagyobb egységszámok automatikusan a soros buszon levő egységre utalnak. A soros buszra parancsot másodlagos címként küldhetünk. Ha nem kívánunk másodlagos címet kiküldeni, akkor a rutin hívása előtt az Y regisztert töltsük fel $FF-re!

HOGYAN HASZNÁLJUK:

1. Töltsük az A regiszterbe a logikai állomány számát!
2. Töltsük az X regiszterbe az egységszámot!
3. Töltsük az Y regiszterbe a másodlagos címet!
4. Hívjuk meg a rutint!

NÉV: **SETNAM**

RENDELTETÉS: Beállítja az állomány nevét.
HÍVÁSI CÍM: $FFBD=65469
ADATÁTVITELI REGISZTEREK: A, X, Y
ÉRINTETT REGISZTEREK: A, X, Y
ELŐKÉSZÍTŐ RUTINOK: nincsenek
HIBALEHETŐSÉGEK: nincsenek
LEÍRÁS: Beállítható az OPEN, LOAD és SAVE rutinoknál használt állományok neve. Az A regiszterbe kell tölteni az állománynév hosszát. Az állomány nevét a memóriában kell valahol tárolni a megfelelő karakterek ASCII kódjaival, majd az X és Y regiszterbe kell tölteni az állománynév első karakterének címét (az X regiszterbe a cím alsó bájtját, az Y regiszterbe a cím felső bájtját). Ha nem akarunk állománynevet meghatározni (pl. ha Datasette-egységen név nélküli

állományt akarunk tárolni, vagy ha nyomtatóra akarunk egy állományt küldeni), akkor az A regiszterbe töltsünk nullát, és ekkor az X és Y regiszterekbe bármilyen érték kerülhet.

HOGYAN HASZNÁLJUK:

1. Tároljuk az állománynevet a memória egy alkalmas területén a leírt módon!
2. Töltsük az A regiszterbe az állománynév hosszát!
3. Töltsük az X regiszterbe a név címének alsó bájtját!
4. Töltsük az Y regiszterbe a név címének felső bájtját!
5. Hívjuk meg a rutint!

NÉV: **SETTIM**

RENDELTETÉS: Beállítja a rendszerórát.
HÍVÁSI CÍM: $FFDB=65499
ADATÁTVITELI REGISZTEREK: A, X, Y
ÉRINTETT REGISZTEREK: nincsenek
ELŐKÉSZÍTŐ RUTINOK: nincsenek
HIBALEHETŐSÉGEK: nincsenek
LEÍRÁS: A rendszeróra értékét egy megszakítási rutin másodpercenként 60-szor megnöveli. Az óra három bájt hosszú, ez lehetővé teszi, hogy értéke akár 5184000 legyen (24 óra). Ekkor azonban az óra újból nullára áll. A rutin hívása előtt az Y regiszternek kell tartalmaznia a beállítani kívánt érték legmagasabb bájtját, az X regiszternek a középső, és az A regiszternek a legalacsonyabb bájtot.

HOGYAN HASZNÁLJUK:

1. Töltsük az Y regiszterbe a legmagasabb helyiértékű bájtot!
2. Töltsük az X regiszterbe a középső bájtot!
3. Töltsük az A regiszterbe a legkisebb helyiértékű bájtot!
4. Hívjuk meg a rutint!

NÉV: **STOP**

RENDELTETÉS: Ellenőrzi, hogy a STOP billentyű le van-e nyomva.
HÍVÁSI CÍM: $FFE1=65505
ADATÁTVITELI REGISZTEREK: A
ÉRINTETT REGISZTEREK: A, X
ELŐKÉSZÍTŐ RUTINOK: nincsenek
HIBALEHETŐSÉGEK: nincsenek

LEÍRÁS: Ha az UDTIM hívása során a STOP billentyű le volt nyomva, akkor e rutin meghívása után a Z jelzőbit értéke 1 lesz, és a csatornák visszaállnak a kezdeti értékekre. Minden más jelzőbit változatlan marad.

HOGYAN HASZNÁLJUK:

1. Hívjuk meg az UDTIM rutint!
2. Hívjuk meg a STOP rutint!
3. Vizsgáljuk a Z jelzőbitet!

NÉV: **UDTIM**

RENDELTETÉS: Növeli a rendszerórát.
HÍVÁSI CÍM: $FFEA=65514
ADATÁTVITELI REGISZTEREK: nincsenek
ÉRINTETT REGISZTEREK: A, X
ELŐKÉSZÍTŐ RUTINOK: nincsenek
HIBALEHETŐSÉGEK: nincsenek
LEÍRÁS: Ez a rutin a rendszerórát növeli. Normál állapotban ezt a rutint az IRQ megszakítási rutin hívja meg másodpercenként 60-szor. Ha a felhasználó saját megszakítási rutint használ, nem szabad megfeledkeznie erről a rutinról, hogy a rendszeróra továbbra is helyesen működjön. Ha a STOP billentyű funkcióját is meg akarjuk őrizni, akkor a rutin hívása után a STOP rutint is meg kell hívni.

HOGYAN HASZNÁLJUK:

Hívjuk meg a rutint!

HIBAKÓDOK

Hiba esetén a KERNAL rutinból való visszatéréskor a C jelzőbit értéke 1, és az A regiszter tartalmazza a hiba kódját. Mindazonáltal egyes rutinok nem ezeket a hibakódokat használják, hanem a hiba a READST rutin hívásával tárható fel.

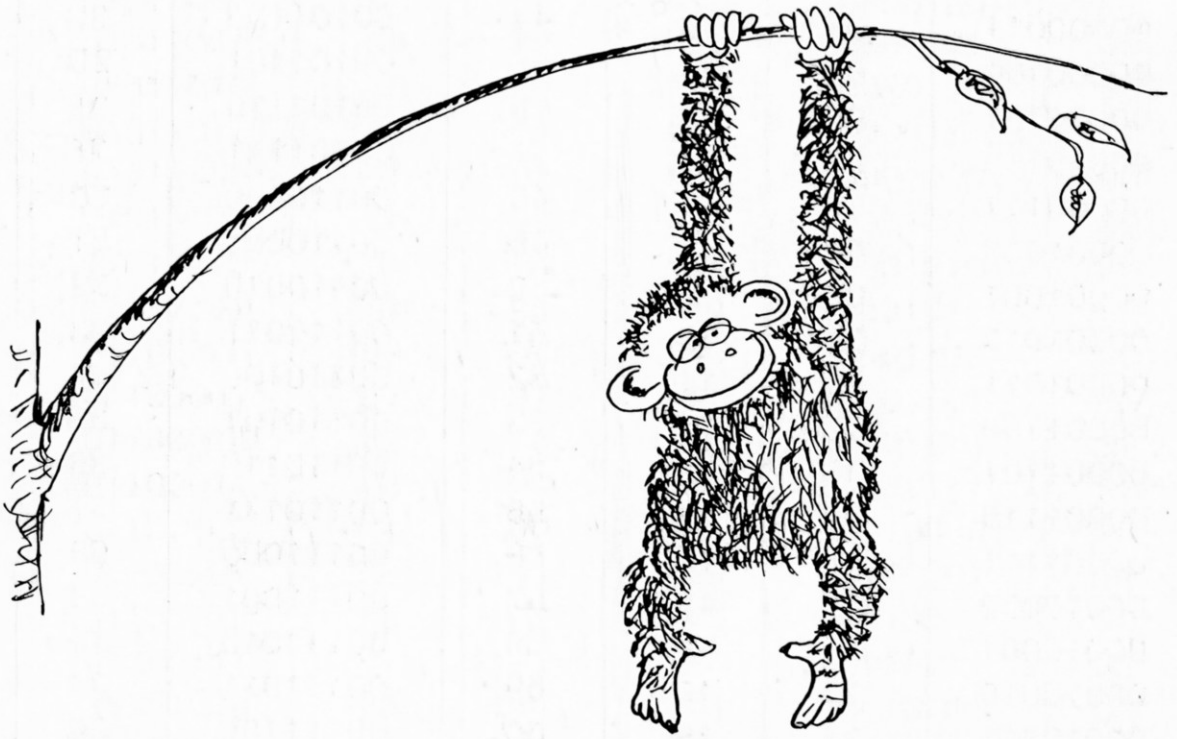
SZÁM	JELENTÉS
0	A rutin a STOP billentyűvel lett megszakítva.
1	Túl sok az egyidejűleg nyitott állomány.
2	Az állomány már meg van nyitva.
3	Az állomány még nincs megnyitva.
4	Az állomány nem található.

5 Az egység nincs jelen (pl. nincs bekapcsolva).
6 Az állomány nem beviteli állomány.
7 Az állomány nem kiviteli állomány.
8 Hiányzik az állománynév.
9 Érvénytelen az egységszám.

240 Memória teteje.

A Kernal rutinok hívására néhány példát a *Függelékben* közlünk a 189. *oldaltól.*

FÜGGELÉK



A függelék néhány hasznos tudnivalót tartalmaz. A közölt táblázatokat nem kell megtanulni. Azért vannak, hogy bármikor fellapozhassuk őket. Természetesen ez a könyv nem tartalmazhat minden a C—64-re vonatkozó információt. Éppen ezért az itt nem talált adatokat máshonnan, egyéb szakirodalomban kell megkeresni.

NYOLCBITES SZÁMOK

decimális, bináris, hexadecimális és kettes komplemens alakban

DEC.	BIN.	HEX.	K.K.	DEC.	BIN.	HEX.	K.K.
0	00000000	00	0	41	00101001	29	41
1	00000001	01	1	42	00101010	2A	42
2	00000010	02	2	43	00101011	2B	43
3	00000011	03	3	44	00101100	2C	44
4	00000100	04	4	45	00101101	2D	45
5	00000101	05	5	46	00101110	2E	46
6	00000110	06	6	47	00101111	2F	47
7	00000111	07	7	48	00110000	30	48
8	00001000	08	8	49	00110001	31	49
9	00001001	09	9	50	00110010	32	50
10	00001010	0A	10	51	00110011	33	51
11	00001011	0B	11	52	00110100	34	52
12	00001100	0C	12	53	00110101	35	53
13	00001101	0D	13	54	00110110	36	54
14	00001110	0E	14	55	00110111	37	55
15	00001111	0F	15	56	00111000	38	56
16	00010000	10	16	57	00111001	39	57
17	00010001	11	17	58	00111010	3A	58
18	00010010	12	18	59	00111011	3B	59
19	00010011	13	19	60	00111100	3C	60
20	00010100	14	20	61	00111101	3D	61
21	00010101	15	21	62	00111110	3E	62
22	00010110	16	22	63	00111111	3F	63
23	00010111	17	23	64	01000000	40	64
24	00011000	18	24	65	01000001	41	65
25	00011001	19	25	66	01000010	42	66
26	00011010	1A	26	67	01000011	43	67
27	00011011	1B	27	68	01000100	44	68
28	00011100	1C	28	69	01000101	45	69
29	00011101	1D	29	70	01000110	46	70
30	00011110	1E	30	71	01000111	47	71
31	00011111	1F	31	72	01001000	48	72
32	00100000	20	32	73	01001001	49	73
33	00100001	21	33	74	01001010	4A	74
34	00100010	22	34	75	01001011	4B	75
35	00100011	23	35	76	01001100	4C	76
36	00100100	24	36	77	01001101	4D	77
37	00100101	25	37	78	01001110	4E	78
38	00100110	26	38	79	01001111	4F	79
39	00100111	27	39	80	01010000	50	80
40	00101000	28	40	81	01010001	51	81

DEC.	BIN.	HEX.	K.K.	DEC.	BIN.	HEX.	K.K.
82	01010010	82	52	126	01111110	7E	126
83	01010011	83	53	127	01111111	7F	127
84	01010100	84	54	128	10000000	80	−128
85	01010101	85	55	129	10000001	81	−127
86	01010110	86	56	130	10000010	82	−126
87	01010111	87	57	131	10000011	83	−125
88	01011000	88	58	132	10000100	84	−124
89	01011001	59	89	133	10000101	85	−123
90	01011010	5A	90	134	10000110	86	−122
91	01011011	5B	91	135	10000111	87	−121
92	01011100	5C	92	136	10001000	88	−120
93	01011101	5D	93	137	10001001	89	−119
94	01011110	5E	94	138	10001010	8A	−118
95	01011111	5F	95	139	10001011	8B	−117
96	01100000	60	96	140	10001100	8C	−116
97	01100001	61	97	141	10001101	8D	−115
98	01100010	62	98	142	10001110	8E	−114
99	01100011	63	99	143	10001111	8F	−113
100	01100100	64	100	144	10010000	90	−112
101	01100101	65	101	145	10010001	91	−111
102	01100110	66	102	146	10010010	92	−110
103	01100111	67	103	147	10010011	93	−109
104	01101000	68	104	148	10010100	94	−108
105	01101001	69	105	149	10010101	95	−107
106	01101010	6A	106	150	10010110	96	−106
107	01101011	6B	107	151	10010111	97	−105
108	01101100	6C	108	152	10011000	98	−104
109	01101101	6D	109	153	10011001	99	−103
110	01101110	6E	110	165	10011010	9A	−102
111	01101111	6F	111	155	10011011	9B	−101
112	01110000	70	112	156	10011100	9C	−100
113	01110001	71	113	157	10011101	9D	− 99
114	01110010	72	114	158	10011110	9E	− 98
115	01110011	73	115	159	10011111	9F	− 97
116	01110100	74	116	160	10100000	A0	− 96
117	01110101	75	117	161	10100001	A1	− 95
118	01110110	76	118	162	10100010	A2	− 94
119	01110111	77	119	163	10100011	A3	− 93
120	01111000	78	120	164	10100100	A4	− 92
121	01111001	79	121	165	10100101	A5	− 91
122	01111010	7A	122	166	10100110	A6	− 90
123	01111011	7B	123	167	10100111	A7	− 89
124	01111100	7C	124	168	10101000	A8	− 88
125	01111101	7D	125	169	10101001	A9	− 87

DEC.	BIN.	HEX.	K.K.	DEC.	BIN.	HEX.	K.K.
170	10101010	AA	– 86	213	11010101	D5	– 43
171	10101011	AB	– 85	214	11010110	D6	– 42
172	10101100	AC	– 84	215	11010111	D7	– 41
173	10101101	AD	– 83	216	11011000	D8	– 40
174	10101110	AE	– 82	217	11011001	D9	– 39
175	10101111	AF	– 81	218	11011010	DA	– 38
176	10110000	B0	– 80	219	11011011	DB	– 37
177	10110001	B1	– 79	220	11011100	DC	– 36
178	10110010	B2	– 78	221	11011101	DD	– 35
179	10110011	B3	– 77	222	11011110	DE	– 34
180	10110100	B4	– 76	223	11011111	DF	– 33
181	10110101	B5	– 75	224	11100000	E0	– 32
182	10110110	B6	– 74	225	11100001	E1	– 31
183	10110111	B7	– 73	226	11100010	E2	– 30
184	10111000	B8	– 72	227	11100011	E3	– 29
185	10111001	B9	– 71	228	11100100	E4	– 28
186	10111010	BA	– 70	229	11100101	E5	– 27
187	10111011	BB	– 69	230	11100110	E6	– 26
188	10111100	BC	– 68	231	11100111	E7	– 25
189	10111101	BD	– 67	232	11101000	E8	– 24
190	10111110	BE	– 66	233	11101001	E9	– 23
191	10111111	BF	– 65	234	11101010	EA	– 22
192	11000000	C0	– 64	235	11101011	EB	– 21
193	11000001	C1	– 63	236	11101100	EC	– 20
194	11000010	C2	– 62	237	11101101	ED	– 19
195	11000011	C3	– 61	238	11101110	EE	– 18
196	11000100	C4	– 60	239	11101111	EF	– 17
197	11000101	C5	– 59	240	11110000	F0	– 16
198	11000110	C6	– 58	241	11110001	F1	– 15
199	11000111	C7	– 57	242	11110010	F2	– 14
200	11001000	C8	– 56	243	11110011	F3	– 13
201	11001001	C9	– 55	244	11110100	F4	– 12
202	11001010	CA	– 54	245	11110101	F5	– 11
203	11001011	CB	– 53	246	11110110	F6	– 10
204	11001100	CC	– 52	247	11110111	F7	– 9
205	11001101	CD	– 51	248	11111000	F8	– 8
206	11001110	CE	– 50	249	11111001	F9	– 7
207	11001111	CF	– 49	250	11111010	FA	– 6
208	11010000	D0	– 48	251	11111011	FB	– 5
209	11010001	D1	– 47	252	11111100	FC	– 4
210	11010010	D2	– 46	253	11111101	FD	– 3
211	11010011	D3	– 45	254	11111110	FE	– 2
212	11010100	D4	– 44	255	11111111	FF	– 1

ÁTVÁLTÁS SZÁMRENDSZEREK KÖZÖTT

Gépeljük be a következő programot! (Ügyeljünk az O és a 0 közötti különbségre! Mindegyik sort a RETURN gombokkal fejezzük be! Ha egy sor begépelését elrontottuk, a RETURN után egyszerűen gépeljük be az egész sort még egyszer: mindig a legutóbbi verzió az érvényes. Nagyon vigyázzunk a betű szerinti pontosságra! A sorokban levő szóközök nyugodtan elhagyhatók.)

```
100 PRINT : PRINT
110 PRINT CHR$(18);"K";CHR$(146);"ETTES"
120 PRINT CHR$(18);"T";CHR$(146);"IZES"
130 PRINT CHR$(18);"H";CHR$(146);"EXA"
140 PRINT
150 GET G$
160 IF G$="K" THEN 200
170 IF G$="T" THEN 290
180 IF G$="H" THEN 320
190 GOTO 150
200 INPUT "KETTES=";K$
210 IF LEN(K$)<1 THEN 200
220 IF LEN(K$)>16 THEN 200
230 N=0
240 FOR I=1 TO LEN(K$)
250 IF MID$(K$,I,1)<>"0" THEN IF MID$(K$,I,1)<>"1" THEN 200
260 N=(N*2)+VAL(MID$(K$,I,1))
270 NEXT I
280 GOTO 420
290 INPUT "TIZES=";N
300 IF N=INT(N) THEN IF N>=0 THEN IF N<=65536 THEN 420
310 GOTO 290
320 INPUT "HEXA=";H$
330 IF LEN(H$)<1 THEN 320
340 IF LEN(H$)>4 THEN 320
350 N=0
360 FOR I=1 TO LEN(H$)
370 A=ASC(MID$(H$,I,1))+((MID$(H$,I,1)>"9")*7)-48
380 IF A<0 THEN 320
390 IF A>15 THEN 320
400 N=(N*16)+A
410 NEXT I
420 PRINT : K=N
430 PRINT "KETTES: ";
440 FOR I=15 TO 0 STEP -1
450 IF K>=2↑I THEN K=K-(2↑I) : PRINT "1"; : GOTO 470
460 PRINT "0";
470 NEXT I
480 PRINT
490 PRINT " TIZES:";N
500 H=N
510 PRINT "  HEXA: ";
520 FOR I=3 TO 0 STEP -1
530 A=INT(H/(16↑I)) : H=H-(A*(16↑I))
540 PRINT CHR$(A+48-((A>9)*7));
550 NEXT I
560 PRINT
570 RUN
```

GÉPI KÓDÚ PROGRAMOK BETÖLTÉSE

A következő Basic programmal könnyen betölthetjük gépi kódú programjainkat. A 320. sorban kell megadnunk a kezdőcímet tizenhatos számrendszerben. Ha ez nem négyjegyű vagy nem tizenhatos számrendszerbeli szám, akkor a program hibajelzéssel leáll. A 360. sortól kezdődően kell megadnunk a gépi kódú programunknak megfelelő számsorozatot. Minden számot kétjegyű, hexadecimális számként kell megadni, ellenkező esetben a program hibaüzenettel leáll. Amennyiben minden adat helyesen van megadva, a program a következő üzenettel áll le:

```
?OUT OF DATA ERROR IN 190
READY.
```

Ez jelen esetben nem hibát jelent, hanem azt, hogy a program kifutott az adatokból, azaz már minden megadott számot beírt a memóriába.

```
100 READ KC$
110 IF LEN(KC$)<>4 THEN PRINT "HIBAS A KEZDOCIM !" : STOP
120 PC=0
130 FOR I=1 TO 4
140 A=ASC(MID$(KC$,I,1))+((MID$(KC$,I,1)>"9")*7)-48
150 IF A<0 THEN PRINT "HIBAS A KEZDOCIM !" : STOP
160 IF A>15 THEN PRINT "HIBAS A KEZDOCIM !" : STOP
170 PC=(PC*16)+A
180 NEXT I
190 READ A$
200 IF LEN(A$)<>2 THEN PRINT "HIBAS ADAT !" : STOP
210 H=ASC(LEFT$(A$,1))+((LEFT$(A$,1)>"9")*7)-48
220 L=ASC(RIGHT$(A$,1))+((RIGHT$(A$,1)>"9")*7)-48
230 IF H>=0 THEN IF H<=15 THEN IF L>=0 THEN IF L<=15 THEN 250
240 PRINT "HIBAS ADAT !" : STOP
250 D=H*16+L
260 POKE PC,D
270 PC=PC+1
280 GOTO 190
290 REM ----
300 REM ---- IDE KERUL A KEZDOCIM ----
310 REM ----
320 DATA C000
330 REM ----
340 REM ---- IDE KERUL A PROGRAM -----
350 REM ----
360 DATA A9,00,8D,00,05,60
```

A közölt adatokkal a program az alábbi gépi kódú programot tölti be:

cím	kód	utasítás
C000	A9 00	LDA #$00
C002	8D 00 05	STA $0500
C005	60	RTS

AZ UTASÍTÁSOK ÉS HATÁSUK A JELZŐBITEKRE

Jelmagyarázat

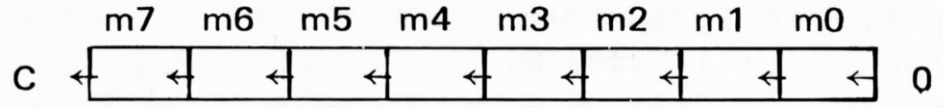
M — Valamely memóriacím tartalma
A — Akkumulátor (A regiszter)
X — X indexregiszter
Y — Y indexregiszter

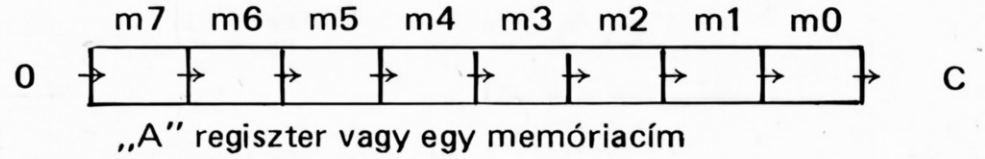
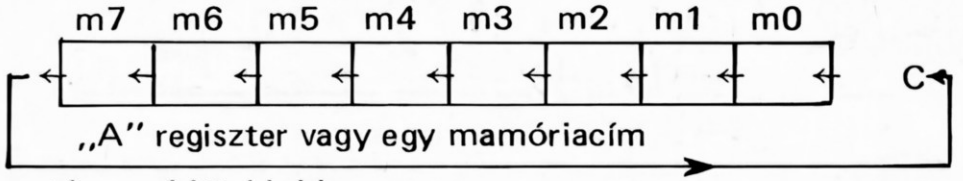
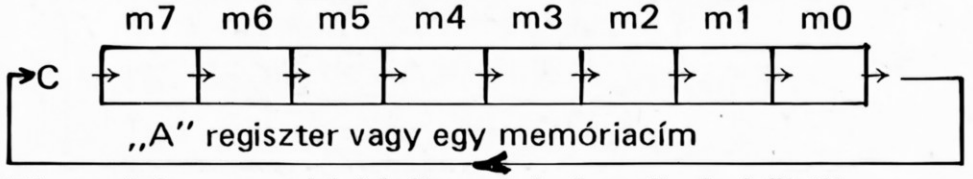
V — Logikai VAGY művelet
$\wedge$ — Logikai ÉS művelet
$\oplus$ — Logikai KIZÁRÓLAGOS VAGY művelet

N — „negatív" jelzőbit
V — „túlcsordulás" jelzőbit
B — "BREAK" jelzőbit
D — „decimális mód" jelzőbit
I — „interrupt" jelzőbit (megszakítások tiltása és engedélyezése)
Z — „zéró" jelzőbit
C — „átvitel" jelzőbit
m7 — a memóriacím hetedik bitje
m6 — a memóriacím hatodik bitje
PC — utasításszámláló, 16 bites regiszter (mindig azt jelzi, hogy a programban mely címnél tart a vezérlő)
PCh — az utasításszámláló felső bájtja
PCl — az utasításszámláló alsó bájtja
SP — veremmutató
ST — a jelzőbitek regisztere

7	6	5	4	3	2	1	0
N	V	.	B	D	I	Z	C

0 : A jelzőbit az utasítás végrehajtásakor törlődik.
1 : A jelzőbit az utasítás végrehajtásakor 1-re állítódik.
$*$: A jelzőbit az utasítás eredményétől függően törlődik vagy állítódik be.
— : A jelzőbit változatlan marad.
7 : A jelzőbitbe a memóriacím 7. bitjének megfelelő érték kerül. (Csak BIT utasításnál.)
6 : A jelzőbitbe a memóriacím 6. bitjének megfelelő érték kerül. (Csak BIT utasításnál.)

Utasítás	Magyarázat, értelmezés	NV.BDIZC
ADC	A+memória+C → C-be és A-ba	××.———××
AND	A∧memória → A-ba	×—.———×—
ASL	Csúsztatás balra.	×—.———××
	 „A" regiszter vagy egy memóriacím	
BCC	Relatív vezérlésátadás, ha C=0	——.——————
BCS	Relatív vezérlésátadás, ha C=1	——.——————
BEQ	Relatív vezérlésátadás, ha Z=1	——.——————
BIT	m7 → N m6 → V Ha A ∧ M = 0, akkor Z = 1 Ha A ∧ M ≠ 0, akkor Z = 0 A művelet alatt A és M értéke nem változik.	76.———×—
BMI	Relatív vezérlésátadás, ha N=1	——.——————
BNE	Relatív vezérlésátadás, ha Z=0	——.——————
BPL	Relatív vezérlésátadás, ha N=0	——.——————
BRK	BREAK megszakítás PCh → verem PCl → verem ST → verem Ugrás ($FFFE)-re. Alaphelyzetben megfelel egy JMP $FF48 utasításnak.	——.1—1——
BVC	Relatív vezérlésátadás, ha V=0	——.——————
BVS	Relatív vezérlésátadás, ha V=1	——.——————
CLC	0 → C	—— . ————0
CLD	0 → D Decimális mód törlése.	——.—0———
CLI	0 → I megszakítások engedélyezése.	——.——0——
CLV	0 → V	—0.——————
CMP	$100+A—M művelet végrehajtása. Ha az eredmény nagyobb, mint $FF, akkor C=1. Ha az eredmény nem nagyobb, mint $FF, akkor C=0. Z jelzőbit és N jelzőbit az eredménytől függően alakul. A művelet után az A és az M tartalma ugyanaz marad, mint ami előtte volt.	×—.———××
CPX	$100+X—M művelet. (Továbbiakban lásd CMP.)	×—.———××
CPY	$100+Y—M művelet. (Továbbiakban lásd CMP.)	×—.———××
DEC	Ha M=0, akkor $FF → M. Ha M≠0, akkor M—1 → M.	×—.———×—
DEX	Ha X=0, akkor $FF → X. Ha X≠0, akkor X—1 → X.	×—.———×—
DEY	Ha Y=0, akkor $FF → Y. Ha Y≠0, akkor Y—1 → Y.	×—.———×—

Utasítás	Magyarázat, értelmezés	NV.BDIZC
EOR	A ⊕ M → A.	✕—.———✕—
INC	Ha M=$FF, akkor $00 → M. Ha M≠$FF, akkor M+1 → M.	✕—.———✕—
INX	Ha X=$FF, akkor $00 → X. Ha X≠$FF, akkor X+1 → X.	✕—.———✕—
INY	Ha Y=$FF, akkor $00 → Y. Ha Y≠$FF, akkor Y+1 → Y.	✕—.———✕—
JMP	Feltétel nélküli vezérlésátadás.	——.——————
JSR	Ugrás szubrutinra, ami a következő 4 részműveletből áll: PC+2 → PC PCh → verem PCl → verem Ugrás a megadott címre.	——.——————
LDA	M → A	✕—.———✕—
LDX	M → X	✕—.———✕—
LDY	M → Y	✕—.———✕—
LSR	Csúsztatás jobbra.  „A" regiszter vagy egy memóriacím	0—.———✕✕
NOP	Üres utasítás.	——.——————
ORA	AVM → A	✕—.———✕—
PHA	A → verem	——.——————
PHP	ST → verem	——.——————
PLA	Verem → A	✕—.———✕—
PLP	Verem → ST	Veremből
ROL	Forgatás egy bittel balra.  „A" regiszter vagy egy mamóriacím	✕—.———✕✕
ROR	Forgatás egy bittel jobbra.  „A" regiszter vagy egy memóriacím	✕—.———✕✕
RTI	Visszatérés megszakításból, ami 4 részműveletből áll: Verem → ST Verem → PCl Verem → PCh Ugrás PC-re.	Veremből

Utasítás	Magyarázat, értelmezés	NV.BDIZC
RTS	Visszatérés szubrutinból, ami 4 részműveletből áll: Verem → PCl Verem → PCh PC+1 → PC Ugrás PC-re.	——.——————
SBC	$100+A—M—¬C → C és A (¬C: a C negáltja)	✕✕.———✕✕
SEC	1 → C	——.————1
SED	1 → D Decimális módra állítás.	——.—1———
SEI	1 → I Maszkolható megszakítások letiltása.	——.——1——
STA	A → M	——.——————
STX	X → M	——.——————
STY	Y → M	——.——————
TAX	A → X	✕—.———✕—
TAY	A → Y	✕—.———✕—
TSX	SP → X	✕—.———x—
TXA	X → A	✕—.———✕—
TXS	X → SP	——.——————
TYA	Y → A	✕—.———✕—

AZ UTASÍTÁSOK KÓDJAI A KÜLÖNBÖZŐ CÍMZÉSI MÓDOKBAN

	Akkumulátor	Közvetlen	Nulladik lapos	Nulladik,X	Nulladik,Y	Abszolút	Abszolút,X	Abszolút,Y	Burkolt	Relatív	(indirekt,X)	(indirekt),Y	Abszolút indirekt
hossz	1	2	2	2	2	3	3	3	1	2	2	2	3
ADC	.	69	65	75	.	6D	7D	79	.	.	61	71	.
AND	.	29	25	35	.	2D	3D	39	.	.	21	31	.
ASL	0A	.	06	16	.	0E	1E	.	.	.	.	.	.
BCC	.	.	.	.	.	.	.	.	.	90	.	.	.
BCS	.	.	.	.	.	.	.	.	.	B0	.	.	.
BEQ	.	.	.	.	.	.	.	.	.	F0	.	.	.
BIT	.	.	24	.	.	2C	.	.	.	.	.	.	.
BMI	.	.	.	.	.	.	.	.	.	30	.	.	.
BNE	.	.	.	.	.	.	.	.	.	D0	.	.	.
BPL	.	.	.	.	.	.	.	.	.	10	.	.	.
BRK	.	.	.	.	.	.	.	.	00	.	.	.	.
BVC	.	.	.	.	.	.	.	.	.	50	.	.	.
BVS	.	.	.	.	.	.	.	.	.	70	.	.	.
CLC	.	.	.	.	.	.	.	.	18	.	.	.	.
CLD	.	.	.	.	.	.	.	.	D8	.	.	.	.
CLI	.	.	.	.	.	.	.	.	58	.	.	.	.
CLV	.	.	.	.	.	.	.	.	B8	.	.	.	.
CMP	.	C9	C5	D5	.	CD	DD	D9	.	.	C1	D1	.
CPX	.	E0	E4	.	.	EC	.	.	.	.	.	.	.
CPY	.	C0	C4	.	.	CC	.	.	.	.	.	.	.
DEC	.	.	C6	D6	.	CE	DE	.	.	.	.	.	.
DEX	.	.	.	.	.	.	.	.	CA	.	.	.	.
DEY	.	.	.	.	.	.	.	.	88	.	.	.	.
EOR	.	49	45	55	.	4D	5D	59	.	.	41	51	.
INC	.	.	E6	F6	.	EE	FE	.	.	.	.	.	.
INX	.	.	.	.	.	.	.	.	E8	.	.	.	.
INY	.	.	.	.	.	.	.	.	C8	.	.	.	.
JMP	.	.	.	.	.	4C	.	.	.	.	.	.	6C

	Akkumulátor	Közvetlen	Nulladik lapos	Nulladik,X	Nulladik,Y	Abszolút	Abszolút,X	Abszolút,Y	Burkolt	Relatív	(indirekt,X)	(indirekt),Y	Abszolút indirekt
hossz	1	2	2	2	2	3	3	3	1	2	2	2	3
JSR	.	.	.	.	.	20	.	.	.	.	.	.	.
LDA	.	A9	A5	B5	.	AD	BD	B9	.	.	A1	B1	.
LDX	.	A2	A6	.	B6	AE	.	BE	.	.	.	.	.
LDY	.	A0	A4	B4	.	AC	BC	.	.	.	.	.	.
LSR	4A	.	46	56	.	4E	5E	.	.	.	.	.	.
NOP	.	.	.	.	.	.	.	.	EA	.	.	.	.
ORA	.	09	05	15	.	0D	1D	19	.	.	01	11	.
PHA	.	.	.	.	.	.	.	.	48	.	.	.	.
PHP	.	.	.	.	.	.	.	.	08	.	.	.	.
PLA	.	.	.	.	.	.	.	.	68	.	.	.	.
PLP	.	.	.	.	.	.	.	.	28	.	.	.	.
ROL	2A	.	26	36	.	2E	3E	.	.	.	.	.	.
ROR	6A	.	66	76	.	6E	7E	.	.	.	.	.	.
RTI	.	.	.	.	.	.	.	.	40	.	.	.	.
RTS	.	.	.	.	.	.	.	.	60	.	.	.	.
SBC	.	E9	E5	F5	.	ED	FD	F9	.	.	E1	F1	.
SEC	.	.	.	.	.	.	.	.	38	.	.	.	.
SED	.	.	.	.	.	.	.	.	F8	.	.	.	.
SEI	.	.	.	.	.	.	.	.	78	.	.	.	.
STA	.	.	85	95	.	8D	9D	99	.	.	81	91	.
STX	.	.	86	.	96	8E	.	.	.	.	.	.	.
STY	.	.	84	94	.	8C	.	.	.	.	.	.	.
TAX	.	.	.	.	.	.	.	.	AA	.	.	.	.
TAY	.	.	.	.	.	.	.	.	A8	.	.	.	.
TSX	.	.	.	.	.	.	.	.	BA	.	.	.	.
TXA	.	.	.	.	.	.	.	.	8A	.	.	.	.
TXS	.	.	.	.	.	.	.	.	9A	.	.	.	.
TYA	.	.	.	.	.	.	.	.	98	.	.	.	.

UTASÍTÁSKÓDOK

A három kérdőjel azokat a kódokat jelöli, amelyekhez a leírások szerint nem tartozik utasítás.

0—$00 BRK
1—$01 ORA (indirekt,X)
2—$02 ???
3—$03 ???
4—$04 ???
5—$05 ORA nulladik
6—$06 ASL nulladik
7—$07 ???
8—$08 PHP
9—$09 ORA közvetlen
10—$0A ASL akkumulátor
11—$0B ???
12—$0C ???
13—$0D ORA abszolút
14—$0E ASL abszolút
15—$0F ???
16—$10 BPL relatív
17—$11 ORA (indirekt),Y
18—$12 ???
19—$13 ???
20—$14 ???
21—$15 ORA nulladik,X
22—$16 ASL nulladik,X
23—$17 ???
24—$18 CLC
25—$19 ORA abszolút,Y
26—$1A ???
27—$1B ???
28—$1C ???
29—$1D ORA abszolút,X
30—$1E ASL abszolút,X
31—$1F ???
32—$20 JSR abszolút
33—$21 AND (indirekt,X)
34—$22 ???
35—$23 ???
36—$24 BIT nulladik
37—$25 AND nulladik
38—$26 ROL nulladik
39—$27 ???
40—$28 PLP
41—$29 AND közvetlen
42—$2A ROL akkumulátor
43—$2B ???
44—$2C BIT abszolút
45—$2D AND abszolút
46—$2E ROL abszolút
47—$2F ???
48—$30 BMI relatív
49—$31 AND (indirekt),Y
50—$32 ???
51—$33 ???
52—$34 ???
53—$35 AND nulladik,X
54—$36 ROL nulladik,X
55—$37 ???
56—$38 SEC
57—$39 AND abszolút,Y
58—$3A ???
59—$3B ???
60—$3C ???
61—$3D AND abszolút,X
62—$3E ROL abszolút,X
63—$3F ???
64—$40 RTI
65—$41 EOR (indirekt,X)
66—$42 ???
67—$43 ???
68—$44 ???
69—$45 EOR nulladik
70—$46 LSR nulladik
71—$47 ???
72—$48 PHA
73—$49 EOR közvetlen
74—$4A LSR akkumulátor
75—$4B ???
76—$4C JMP abszolút
77—$4D EOR abszolút
78—$4E LSR abszolút
79—$4F ???
80—$50 BVC relatív
81—$51 EOR (indirekt),Y
82—$52 ???

83–$53 ???
84–$54 ???
85–$55 EOR nulladik,X
86–$56 LSR nulladik,X
87–$57 ???
88–$58 CLI
89–$59 EOR abszolút,Y
90–$5A ???
91–$5B ???
92–$5C ???
93–$5D EOR abszolút,X
94–$SE LSR abszolút,X
95–$5F ???
96–$60 RTS
97–$61 ADC (indirekt,X)
98–$62 ???
99–$63 ???
100–$64 ???
101–$65 ADC nulladik
102–$66 ROR nulladik
103–$67 ???
104–$68 PLA
105–$69 ADC közvetlen
106–$6A ROR akkumulátor
107–$6B ???
108–$6C JMP abszolút indirekt
109–$6D ADC abszolút
110–$6E ROR abszolút
111–$6F ???
112–$70 BVS relatív
113–$71 ADC (indirekt),Y
114–$72 ???
115–$73 ???
116–$74 ???
117–$75 ADC nulladik,X
118–$76 ROR nulladik,X
119–$77 ???
120–$78 SEI
121–$79 ADC abszolút,Y
122–$7A ???
123–$7B ???
124–$7C ???
125–$7D ADC abszolút,X
126–$7E ROR abszolút,X
127–$7F ???
128–$80 ???
129–$81 STA (indirekt,X)
130–$82 ???
131–$83 ???
132–$84 STY nulladik
133–$85 STA nulladik
134–$86 STX nulladik
135–$87 ???
136–$88 DEY
137–$89 ???
138–$8A TXA
139–$8B ???
140–$8C STY abszolút
141–$8D STA abszolút
142–$8E STX abszolút
143–$8F ???
144–$90 BCC relatív
145–$91 STA (indirekt),Y
146–$92 ???
147–$93 ???
148–$94 STY nulladik,X
149–$95 STA nulladik,X
150–$96 STX nulladik, Y
151–$97 ???
152–$98 TYA
153–$99 STA abszolút,Y
154–$9A TXS
155–$9B ???
156–$9C ???
157–$9D STA abszolút,X
158–$9E ???
159–$9F ???
160–$A0 LDY közvetlen
161–$A1 LDA (indirekt,X)
162–$A2 LDX közvetlen
163–$A3 ???
164–$A4 LDY nulladik
165–$A5 LDA nulladik
166–$A6 LDX nulladik
167–$A7 ???
168–$A8 TAY
169–$A9 LDA közvetlen
170–$AA TAX
171–$AB ???
172–$AC LDY abszolút
173–$AD LDA abszolút
174–$AE LDX abszolút
175–$AF ???
176–$B0 BCS relatív

177–$B1 LDA (indirekt),Y
178–$B2 ???
179–$B3 ???
180–$B4 LDY nulladik,X
181–$B5 LDA nulladik,X
182–$B6 LDX nulladik,Y
183–$B7 ???
184–$B8 CLV
185–$B9 LDA abszolút,Y
186–$BA TSX
187–$BB ???
188–$BC LDY abszolút,X
189–$BD LDA abszolút,X
190–$BE LDX abszolút,Y
191–$BF ???
192–$C0 CPY közvetlen
193–$C1 CMP (indirekt,X)
194–$C2 ???
195–$C3 ???
196–$C4 CPY nulladik
197–$C5 CMP nulladik
198–$C6 DEC nulladik
199–$C7 ???
200–$C8 INY
201–$C9 CMP közvetlen
202–$CA DEX
203–$CB ???
204–$CC CPY abszolút
205–$CD CMP abszolút
206–$CE DEC abszolút
207–$CF ???
208–$D0 BNE relatív
209–$D1 CMP (indirekt),Y
210–$D2 ???
211–$D3 ???
212–$D4 ???
213–$D5 CMP nulladik,X
214–$D6 DEC nulladik,X
215–$D7 ???
216–$D8 CLD
217–$D9 CMP abszolút,Y
218–$DA ???
219–$DB ???
220–$DC ???
221–$DD CMP abszolút,X
222–$DE DEC abszolút,X
223–$DF ???
224–$E0 CPX közvetlen
225–$E1 SBC (indirekt,X)
226–$E2 ???
227–$E3 ???
228–$E4 CPX nulladik
229–$E5 SBC nulladik
230–$E6 INC nulladik
231–$E7 ???
232–$E8 INX
233–$E9 SBC közvetlen
234–$EA NOP
235–$EB ???
236–$EC CPX abszolút
237–$ED SBC abszolút
238–$EE INC abszolút
239–$EF ???
240–$F0 BEQ relatív
241–$F1 SBC (indirekt),Y
242–$F2 ???
243–$F3 ???
244–$F4 ???
245–$F5 SBC nulladik,X
246–$F6 INC nulladik,X
247–$F7 ???
248–$F8 SED
249–$F9 SBC abszolút,Y
250–$FA ???
251–$FB ???
252–$FC ???
253–$FD SBC abszolút,X
254–$FE INC abszolút,X
255–$FF ???

AZ UTASÍTÁSOK VÉGREHAJTÁSI SEBESSÉGE

A Commodore–64 számítógépben a gépi kódú utasításokat egy mikroprocesszor hajtja végre. Ez egy soklábú áramkör, a könyv példáit tekintve magában foglalja a vezérlőt, az ALU-t és a memóriakezelőt.
A MOS 6510 mikroprocesszor az utasításokat folytonos egymás utánban hajtja végre, feltéve, hogy vezérlésátadó utasítások másként nem rendelkeznek. Amint végzett egy utasítással, azonnal elkezdi végrehajtani a következőt, vagyis soha nem áll, mindig csinál valamit. A C–64 bekapcsoláskor a tevékenységét a $FFFC címen tárolt RESET vektor által jelölt címen ($FCE2) kezdi. Természetesen egy-egy utasítás elvégzéséhez időre van szükség, ám ezt az időt nem másodpercekben mérjük, hanem órajel ciklusokban, mert a processzor nem időre dolgozik, hanem ritmusra. Gyorsasága tehát a ritmus gyorsaságától függ. A ritmust pedig kívülről kapja. Van ugyanis a C–64-ben egy belső óra, ami másodpercenként kb. egymillió ütemet „ver a fülébe", és így a másodpercenkénti 1000000 ütemhez kell igazítania a gyorsaságát. Ennek következtében az egyes utasítások végrehajtásához szükséges idők így alakulnak:

2 ciklusos utasítás – kb. 0.000002 sec
3 ciklusos utasítás – kb. 0.000003 sec
4 ciklusos utasítás – kb. 0.000004 sec
5 ciklusos utasítás – kb. 0.000005 sec
6 ciklusos utasítás – kb. 0.000006 sec
7 ciklusos utasítás – kb. 0.000007 sec

Láthatjuk, hogy ezek az értékek igen kicsik. Az egyes utasítások rendkívül gyorsan végrehajtódnak. Így a gépi kódban való programozással, a feladattól függően tíz-, száz- vagy akár ezerszeres sebességet is elérhetünk a Basic-kel szemben.

AZ UTASÍTÁSOK VÉGREHAJTÁSÁNAK IDEJE ÓRAJEL CIKLUSOKBAN

	Akkumulátor	Közvetlen	Nulladik lapos	Nulladik,X	Nulladik,Y	Abszolút	Abszolút,X	Abszolút,Y	Burkolt	Relatív	(indirekt,X)	(indirekt),Y	Abszolút indirekt
ADC	.	2	3	4	.	4	4*	4*	.	.	6	5*	.
AND	.	2	3	4	.	4	4*	4*	.	.	6	5*	.
ASL	2	.	5	6	.	6	7	.	.	.	.	.	.
BCC	.	.	.	.	.	.	.	.	.	2**	.	.	.
BCS	.	.	.	.	.	.	.	.	.	2**	.	.	.
BEQ	.	.	.	.	.	.	.	.	.	2**	.	.	.
BIT	.	.	3	.	.	4	.	.	.	.	.	.	.
BMI	.	.	.	.	.	.	.	.	.	2**	.	.	.
BNE	.	.	.	.	.	.	.	.	.	2**	.	.	.
BPL	.	.	.	.	.	.	.	.	.	2**	.	.	.
BRK	.	.	.	.	.	.	.	.	7	.	.	.	.
BVC	.	.	.	.	.	.	.	.	.	2**	.	.	.
BVS	.	.	.	.	.	.	.	.	.	2**	.	.	.
CLC	.	.	.	.	.	.	.	.	2	.	.	.	.
CLD	.	.	.	.	.	.	.	.	2	.	.	.	.
CLI	.	.	.	.	.	.	.	.	2	.	.	.	.
CLV	.	.	.	.	.	.	.	.	2	.	.	.	.
CMP	.	2	3	4	.	4	4*	4*	.	.	6	5*	.
CPX	.	2	3	.	.	4	.	.	.	.	.	.	.
CPY	.	2	3	.	.	4	.	.	.	.	.	.	.
DEC	.	.	5	6	.	6	7	.	.	.	.	.	.
DEX	.	.	.	.	.	.	.	.	2	.	.	.	.
DEY	.	.	.	.	.	.	.	.	2	.	.	.	.
EOR	.	2	3	4	.	4	4*	4*	.	.	6	5*	.
INC	.	.	5	6	.	6	7	.	.	.	.	.	.
INX	.	.	.	.	.	.	.	.	2	.	.	.	.
INY	.	.	.	.	.	.	.	.	2	.	.	.	.
JMP	.	.	.	.	.	3	.	.	.	.	.	.	5

	Akkumulátor	Közvetlen	Nulladik lapos	Nulladik,X	Nulladik,Y	Abszolút	Abszolút,X	Abszolút,Y	Burkolt	Relatív	(indirekt,X)	(indirekt),Y	Abszolút indirekt
JSR	.	.	.	.	.	6	.	.	.	.	.	.	.
LDA	.	2	3	4	.	4	4*	4*	.	.	6	5*	.
LDX	.	2	3	.	4	4	.	4*	.	.	.	.	.
LDY	.	2	3	4	.	4	4*	.	.	.	.	.	.
LSR	2	.	5	6	.	6	7	.	.	.	.	.	.
NOP	.	.	.	.	.	.	.	.	2	.	.	.	.
ORA	.	2	3	4	.	4	4*	4*	.	.	6	5*	.
PHA	.	.	.	.	.	.	.	.	3	.	.	.	.
PHP	.	.	.	.	.	.	.	.	3	.	.	.	.
PLA	.	.	.	.	.	.	.	.	4	.	.	.	.
PLP	.	.	.	.	.	.	.	.	4	.	.	.	.
ROL	2	.	5	6	.	6	7	.	.	.	.	.	.
ROR	2	.	5	6	.	6	7	.	.	.	.	.	.
RTI	.	.	.	.	.	.	.	.	6	.	.	.	.
RTS	.	.	.	.	.	.	.	.	6	.	.	.	.
SBC	.	2	3	4	.	4	4*	4*	.	.	6	5*	.
SEC	.	.	.	.	.	.	.	.	2	.	.	.	.
SED	.	.	.	.	.	.	.	.	2	.	.	.	.
SEI	.	.	.	.	.	.	.	.	2	.	.	.	.
STA	.	.	3	4	.	4	5	5	.	.	6	6	.
STX	.	.	3	.	4	4	.	.	.	.	.	.	.
STY	.	.	3	4	.	4	.	.	.	.	.	.	.
TAX	.	.	.	.	.	.	.	.	2	.	.	.	.
TAY	.	.	.	.	.	.	.	.	2	.	.	.	.
TSX	.	.	.	.	.	.	.	.	2	.	.	.	.
TXA	.	.	.	.	.	.	.	.	2	.	.	.	.
TXS	.	.	.	.	.	.	.	.	2	.	.	.	.
TYA	.	.	.	.	.	.	.	.	2	.	.	.	.

* Egy ciklussal több, ha az indexelés lapátfedést okoz.

** Egy ciklussal több, ha megvalósul az elágazás. Még egy ciklussal több, ha az elágazás másik lapra irányul.

A SUPER 64—MON MONITORPROGRAM HASZNÁLATA

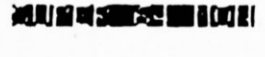
Gépi kódú programot Basic programmal lehet a memóriába tölteni. Néha ugyan hosszú DATA-sorokat kell írni, és előtte kódok után is kell keresgélni, de megoldható. Ha azonban már kész, esetleg mások által írt programokat kell visszafejtenünk, akkor megáll a tudomány. Egyenként kell kiíratnunk a memóriarekeszek tartalmát, majd a táblázatokban kell kikeresni, hogy az egyes kódokhoz milyen utasítások tartoznak.
Próbálkozzon meg a visszafejtéssel az Olvasó is! Fejtse vissza például a rendszer egy rövid részét! Mondjuk a $F69B címtől kezdődően 22 utasítást! Aki, akár csak egyszer is, próbált már kész gépi kódú programot táblázatok alapján visszafejteni, nem lesz tanácstalan, ha majd valamelyik haragosának akar valami igazán gonoszat kívánni.

A SUPER 64—MON monitorprogram egyszerű és aránylag elterjedt, így könnyen hozzáférhető. Aki nem tud hozzájutni, választhat a piacon levő sok egyéb monitorprogram közül, csak ne felejtsen el leírást is szerezni hozzá.
A programok engedély nélküli másolása szerzői jogi törvényekbe ütköző tevékenység, mely büntetést vonhat maga után. A mániákus programmásolók ezért lehetőleg csak kulcsra zárt ajtók mögött, a legnagyobb körültekintéssel és titoktartással másolnak, leginkább az éjféli bagolyhuhogás és a hajnali kakasszó közötti időszakban, amikor a szerzői

jogvédő hivatal munkatársai talán maguk is másolt játékprogramokkal játszva hódolnak számítógépes szenvedélyüknek.
Programot másolni kizárólag saját, egyéni, nem üzleti célra: szabad, legális. Másolt programmal üzleti tevékenységet folytatni, akár közvetve is: szigorúan tiltott. Mindezek után tekintsük át a SUPER 64—MON monitorprogram kezelését!
A program betöltése lemezről: LOAD "SUPER 64—MON",8

Kazettáról pedig: LOAD "SUPER 64—MON"

Ha más néven van tárolva, akkor természetesen azt a nevet kell a LOAD után az idézőjelek közé írni. A programot mindkét esetben a RUN utasítással indíthatjuk. Indítás után a képernyőn a következő jelenik meg:



```
   .. JIM BUTTERFIELD
B*
     PC   SR  AC  XR   YR   SP
.;   97FE 31  40  E6   00   F6
.■
```

A programot Jim Butterfield írta.
A programban használt betűkombinációk az alábbiakat jelentik:

PC — utasításszámláló
SR — státuszregiszter, azaz a jelzőbitek
AC — akkumulátor — A regiszter
XR — X indexregiszter
YR — Y indexregiszter
SP — veremmutató

Ezeket a regisztertartalmakat a program tárolja, és ha a monitorprogramból indítunk egy gépi kódú programot, akkor ezekkel az értékekkel indulnak a regiszterek. Ezeket az értékeket persze felülírhatjuk és a RETURN billentyű leütésével véglegesíthetjük.

A képernyőn minden, amit a program ír ki, fekete színben jelenik meg, a programozó által begépelt szövegek pedig fehérben.
A monitorprogram a sor elejére írt ponttal jelzi, hogy begépelhetjük az utasításunkat.
A monitorprogramban egybetűs utasításokat adhatunk. Ez a betű általában az angol nyelvű utasítás első betűje.
Nézzük meg először, hogy hogyan lehet a monitorprogrammal visszafejteni! Erre a D utasítás szolgál. A D utasítás mellé meg kell még adni, hogy melyik memóriacímtől kezdődően akarunk visszafejteni. Írjuk be például az alábbi utasítást:

.D F69B

Ez most azt jelenti, hogy a $F69B címtől akarunk visszafejteni 22 utasítást. Azért 22 utasítást, mert egyszerre csak ennyi fér el a képernyőn. Az utasítás hatására a képernyőn a következők jelennek meg:

```
. , F69B A2 00      LDX #$00
. , F69D E6 A2      INC $A2
. , F69F D0 06      BNE $F6A7
. , F6A1 E6 A1      INC $A1
. , F6A3 D0 02      BNE $F6A7
. , F6A5 E6 A0      INC $A0
. , F6A7 38         SEC
. , F6A8 A5 A2      LDA $A2
. , F6AA E9 01      SBC #$01
. , F6AC A5 A1      LDA $A1
. , F6AE E9 1A      SBC #$1A
. , F6B9 A5 A0      LDA $A0
. , F6B2 E9 4F      SBC #$4F
. , F6B4 90 06      BCC $F6BC
. , F6B6 86 A0      STX $A0
. , F6B8 86 A1      STX $A1
. , F6BA 86 A2      STX $A2
. , F6BC AD 01 DC   LDA $DC01
. , F6BF CD 01 DC   CMP $DC01
. , F6C2 D0 F8      BNE $F6BC
. , F6C4 AA         TAX
. ■ F6C5 30 13      BMI $F6DA
```

A program villanásnyi idő alatt visszafejtett 22 utasítást. Ha most nem mozgatjuk sehova a kurzort, és lenyomjuk a D billentyűt, majd utána a RETURN-t, akkor újból visszafejt 22 utasítást, a legalul levő utasítástól kezdődően.

```
. , F6C5 30 13      BMI $F6DA
. , F6C7 A2 BD      LDX #$BD
. , F6C9 8E 00 DC   STX $DC00
. , F6CC AE 01 DC   LDX $DC01
. , F6CF EC 01 DC   CPX $DC01
. , F6D2 D0 F8      BNE $F6CC
. , F6D4 8D 00 DC   STA $DC00
. , F6D7 E8         INX
. , F6D8 D0 02      BNE $F6DC
. , F6DA 85 91      STA $91
. , F6DC 60         RTS
. , F6DD 78         SEI
. , F6DE A5 A2      LDA $A2
. , F6E0 A6 A1      LDX $A1
. , F6E2 A4 A0      LDX $A0
. , F6E4 78         SEI
. , F6E5 85 A2      STA $A2
. , F6E7 86 A1      STX $A1
. , F6E9 84 A0      STX $A0
. , F6EB 58         CLI
. , F6EC 60         RTS
. ■ F6ED A5 91      LDA $91
```

Ha megunjuk a visszafejtést, a SHIFT és a CLR/HOME billentyűk egyidejű lenyomásával töröljük a képernyőt. A program ekkor újabb utasításra vár majd.
Írjunk egy rövid programot a monitorprogram segítségével, a $C000 címtől kezdődően! Először az A betűvel jelezzük, hogy programot akarunk írni, majd megadjuk a kezdőcímet és az első utasítást.

```
.A C000 LDA #$00■
```

Ha leütjük a RETURN billentyűt, akkor a következő választ kapjuk:

```
.A C000 LDA #$00
.A C002 ■
```

A program jelezte, hogy a következő utasítás már a $C002 címen lesz, és természetesen a LDA #$00 utasítást beírta a memóriába a $C000 címtől kezdve. Nem kellett kódokat keresgélnünk, nem kellett POKE-olni, csak az utasítást kellett megadnunk — minden további tevékenységet elvégzett a monitorprogram. Nekünk csak annyi a feladatunk, hogy begépeljük a következő utasítást:

```
.A C000 LDA #$00
.A C002 STA $D020■
```

A RETURN billentyű leütése után most sem késik a válasz.

```
.A C000 LDA  #$00
.A C002 STA  $D020
.A C005 ■
```

A program tárolta a $C002 címen a STA $D020 utasítást, és most jelzi, hogy a következő utasítás a $C005 címre fog kerülni. Írjuk hát be!

```
.A C000 LDA  #$00
.A C002 STA  $D020
.A C005 BRK■
```

A RETURN billentyű leütése után:

```
.A C000 LDA  #$00
.A C002 STA  $D020
.A C005 BRK
.A C006 ■
```

Ha begépelnénk egy új utasítást, akkor az a $C006 címre kerülne. De mi üssük le egyszerűen csak a RETURN billentyűt!

```
.A C000 LDA  #$00
.A C002 STA  $D020
.A C005 BRK
.A C006
.■
```

A monitorprogram kilépett a programszerkesztő üzemmódból, és várja újabb parancsainkat.
Ennyire könnyedén írhatunk programot a monitorprogram segítségével. Egyszerűen csak be kell gépelnünk az egyes utasításokat. A kódokkal nem kell törődnünk, minden kényelmetlen teendőt magára vállal a monitorprogram.
Programunk a memóriában van. Győződjünk meg erről a P utasítás segítségével! A P utasítással egy adott memóriatartományt fejthetünk vissza. Meg kell adnunk az első és az utolsó visszafejteni kívánt utasítás címét is. Fejtsük most vissza saját programunkat!

```
.P C000 C005
```

A RETURN billentyű leütésére válaszként a képernyőn megjelenik programunk fordítási listája.

```
.A  C000 LDA  #$00
.A  C002 STA  $D020
.A  C005 BRK
.A  C006
.P  C000 C005
C000   A9 00        LDA  #$00
C002   8D 20 D0     STA  $D020
C005   00           BRK
.■
```

Programunk tehát valóban a memóriában van. Erről az M utasítással is meggyőződhetünk. Ezzel az utasítással a memória tartalmát vizsgálhatjuk. Ha a $C000-tól $C007-ig terjedő memóriaterület tartalmára vagyunk kíváncsiak, gépeljük be a következőt:

```
.M C000 C007
```

A RETURN billentyű leütésére az alábbi választ kapjuk:

```
.A C000  LDA  #$00
.A C002  STA  $D020
.A C005  BRK
.A C006
.P C000  C005
C000 A9  00        LDA  #$00
C002 8D  20  D0    STA  $D020
C005 00            BRK
.M C000  C007
.: C000 A9  00  8D  20  D0  00 CF  58
.■
```

Most már teljesen nyugodtak lehetünk. Programunk a memóriában van a $C000 címtől a $C005 címig. Futtatására szolgál a G utasítás.

.G C000

Ha ezután leütjük a RETURN billentyűt is, akkor hirtelen fekete lesz a keret színe, majd újból bejelentkezik a monitorprogram.

Programot futtatni tehát a G utasítással lehet, amelyet a kezdőcímnek kell követnie. Meglevő programot módosítani is egyszerű, ha a monitorprogramot használjuk hozzá. Változtassuk meg az első utasítást LDA #$00-ról LDA #$01-re! (Ez azt fogja jelenteni, hogy a keretszín kódja ne 0, hanem 1 legyen.) Ehhez először listázzuk ki a programot!

.D C000

A képernyőn most az első három utasítás a mi programunk, a többi pedig lehet bármi, attól függően, hogy mi volt előzőleg a memóriában. Észrevehetjük, hogy ha visszafejtés során a monitorprogram visszafejthetetlen kódot talál, akkor három kérdőjelet ír ki.
Vigyük fel a kurzort az első utasításhoz, és a LDA #$00 utasítás második bájtját (nem az utasítást, hanem a kódját) írjuk át 00-ról 01-re! Üssük le a RETURN billentyűt, és az utasítás átváltozik LDA #$01 utasítássá! Ha most töröljük a képernyőt, a P utasítással is kilistázhatjuk a programot:

.P C000 C005

Lefuttathatjuk ezt a programot is. Lehet azonban úgy is futtatni, hogy csak a G betűt ütjük le, és nem adunk meg kezdőcímet. Ilyenkor mindig a tárolt PC értéktől indul a program. Csakhogy ez most nem $C000, hanem $C005.
Ha újból kiíratjuk a regisztertartalmakat, akkor megváltoztathatjuk őket. A regisztertartalmak kijelzését az R utasítással lehet előidézni. Írjuk hát be:

.R

Vigyük a kurzort a kurzormozgató gombok segítségével a PC felirat alatti számértékhez, és egyszerűen írjuk át $C000-ra, majd üssük le a RETURN-t! Mivel a tárolt PC érték most már $C000, így nyugodtan lefuttathatjuk a programot pusztán a G begépelésével:

.G

Ha mindent pontosan csináltunk, akkor most fehér lett a keretszín. Gépi kódú programunk sikeresen futott.

Egy jól működő programot érdemes tárolni, hogy esetleg később is futtatni tudjuk. Erre szolgál az S utasítás.

.S "programnév", tárolóegység száma, kezdőcím, végcím+1

Ha lemezen akarjuk tárolni, akkor az alábbiakat kell begépelnünk:

.S "FEHER",08,C000,C006

Ha kazettára tárolunk, akkor a tároló utasítás módosul:

.S "FEHER",01,C000,C006

A nevet követően kell megadni az egységszámot. Ez lemez esetén 08, kazettás egység esetén pedig 01.
Az egységszámot követően jön a kezdőcím, majd az első, már nem tárolandó bájt címe. Programunk a $C000 címen kezdődik, és a $C005 címen végződik, így az első nem tárolandó cím a $C006.
Ha sikeresen tároltuk a programot, akkor akár törölhetjük is a memóriát. Bármekkora memóriaterületet feltölthetünk bármivel az F utasítás segítségével. Nézzünk egy példát:

.F C000 CFFF EA

Ennek eredményeként $C000-tól kezdődően egészen $CFFF-ig minden egyes címre a $EA érték kerül, a NOP utasítás kódja. Ellenőrizzük!

.P C000 CFFF

A RETURN billentyű leütését követően hosszú ideig csak csupa NOP utasítást láthatunk a képernyőn. Amikor meguntuk, üssük le a RUN/STOP billentyűt! A listázás nyomban befejeződik. A memória tartalmát persze most is vizsgálhatjuk az M utasítással.

.M C000 CFFF

Ezt a kijelzést is megszakíthatjuk a RUN/STOP billentyűvel.
A memóriatartalom számszerű kijelzése szintén megváltoztatható felülírással. Listázzuk ki a memória egy részét a következő utasítással!

.M C800 C80F

Az utasítás hatására a képernyő a következő lesz:

```
. M C800  C80F
. :C800  EA EA EA EA EA EA EA EA
. :C808  EA EA EA EA EA EA EA EA
. ■
```

Amelyik számra mozgatjuk a kurzort, azt át is írhatjuk. Írjuk át az egyes számokat a következőképpen (természetesen minden egyes sor átírása után üssük le a RETURN billentyűt!):

```
. M C800  C80F
. :C800  A9 00  A2 28  9D 4F 04  EA
. :C800  D0 FA 00  EA EA EA EA EA
.■
```

Ha most újra kilistázzuk a memória tartalmát, akkor láthatóvá válik a változás:

```
. M C800  C80F
. :C800  A9 00  A2 28  9D 4F 04  CA
. :C800  D0 FA 00  EA EA EA EA EA
. M C800  C80F
. :C800  A9 00  A2 28  9D 4F 04  CA
. :C808  D0 FA 00  EA EA EA EA EA
```

A memória egyes címeit tehát a kilistázott értékek felülírásával lehet megváltoztatni. Jelen esetben egy programot töltöttünk be így. Fejtsük vissza:

```
. M C800 C80F
. :C800 A9 00  A2 28  9D 4F 04  CA
. :C808 D0 FA 00  EA EA EA EA EA
. M C800 C80F
. :C800 A9 00  A2 28  9D 4F 04  CA
. :C808 D0 FA 00  EA EA EA EA EA
. P C800 C80A
C800  A9 00      LDA #$00
C802  A2 28      LDX #$28
C804  9D 4F 04   STA $044F,X
C807  CA         DEX
C808  D0 FA      BNE $C804
C80A  00         BRK
.■
```

A programot a T utasítással helyezhetjük át egy másik területre. Nézzük meg először, hogy mi van a $C200 címtől kezdődően! Töröljük a képernyőt, és írjuk be:

.D C200

Mivel előzőleg a $C000-tól $CFFF-ig terjedő területet feltöltöttük $EA-val, így a kijelzett lista az alábbi lesz:

```
. , C200  EA      NOP
. , C201  EA      NOP
. , C202  EA      NOP
. , C203  EA      NOP
. , C204  EA      NOP
. , C205  EA      NOP
. , C206  EA      NOP
. , C207  EA      NOP
. , C208  EA      NOP
. , C209  EA      NOP
. , C20A  EA      NOP
. , C20B  EA      NOP
. , C20C  EA      NOP
. , C20D  EA      NOP
. , C20E  EA      NOP
. , C20F  EA      NOP
. , C210  EA      NOP
. , C211  EA      NOP
. , C212  EA      NOP
. , C213  EA      NOP
. , C214  EA      NOP
. ■ C215  EA      NOP
```

Most újból töröljük a képernyőt, és helyezzük át a $C800 címen kezdődő programunkat a $C200 címre (az utolsó áthelyezendő cím a $C80A)!

.TC800 C80A C200

Most is ellenőrizzünk! Programunk nemcsak át lett helyezve, de még a relatív címzések is helyesen működnek az új területen. A relatív címzés előnye az, hogy az ilyen címzésű uta-

sításokat használó program változtatás nélkül áthelyezhető a memória bármely másik területére.
Ismerjük meg a betöltésre szolgáló utasítást is! Töltsük be előzőleg tárolt FEHER nevű programunkat! Lemezről a következő utasítással tölthetjük be: .L "FEHER",08

Kazettáról pedig: .L "FEHER",01

Listázással meggyőződhetünk a betöltés elvégzéséről:

```
.T C800  C80A  C200
.P C200  C20A
C200  A9  00        LDA  #$00
C202  A2  28        LDX  #$28
C204  9D  4F 04     STA  $044F,X
C207  CA            DEX
C208  D0  FA        BNE  $C204
C20A  00            BRK
.L "FEHER",08
SEARCHING FOR FEHER
LOADING
.P C000  C005
C000  A9  01        LDA  #$01
C002  8D  20 D0     STA  $D020
C005  00            BRK
.■
```

A memória bármely területén kereshetünk egy adott bájtsorozatot, anélkül, hogy egyenként végig kellene néznünk a memóriacímeket. Erre szolgál a H utasítás, amit a memóriaterület kezdő- és végcímének, valamint a keresett bájtsorozatnak kell követnie. Ezzel az utasítással maximálisan 32 bájt hosszú bájtsorozatot kereshetünk. Keressük meg, hogy jelenleg a memóriában a $C000 és $CFFF címek között hol helyezkedik el az A9,00 kódsorozat!

```
.H C000  CFFF  A9 00
C200  C800
.■
```

Ezzel az utasítással ASCII-kódokkal tárolt szöveget is kereshetünk. Nézzünk erre is egy példát!

```
.H A000  BFFF  CBMBASIC
A004
.■
```

A programozást befejezve, a monitorprogramból az X utasítással térhetünk vissza a Basicbe.
Jól jegyezzük meg, hogy ha a SUPER 64—MON monitorprogramból akarjuk gépi kódú programunkat indítani, akkor az BRK utasítással kell befejezni! (Az RTS utasítással lezárt programok a Basicbe térnek vissza.)
Ha a monitorprogram üzeneteit (például egy P utasítás eredményét) a nyomtatóra szeretnénk kiküldeni, akkor a monitorprogramot ne RUN utasítással indítsuk, hanem a következő utasítássorozattal:

OPEN 4,4 : CMD 4 : SYS 2176

Ezek után a monitorprogram minden válasza és üzenete a nyomtatóra kerül. Tekintsük át az utasításokat ABC-sorrendben!

A (Assembler — programbeírás)
Csak a program első utasításánál kell az A betűt és a címet megadni. A további

utasítások előtt a monitorprogram automatikusan kijelzi az A betűt is, és a címet is.
Példa: .A C000 LDA #$01

D (Disassembler — visszafejtés)
22 utasítást fejthetünk vissza a megadott címtől kezdődően. A visszafejtett utasításkódok átírásával módosíthatjuk a programot.
Példa: .D C000

F (Fill — feltöltés)
A memória bármely területét feltölthetjük egy bizonyos értékre.
Példa: .F C000 CFFF EA
(a $C000-tól $CFFF-ig terjedő memóriaterület minden címére beírja a $EA értéket)

G (Go — programindítás)
Elindít egy megírt programot. Szerepe ugyanaz, mint a Basic SYS utasításának. Ha nem adunk meg kezdőcímet, akkor a PC tárolt értékétől kezdve indítja a programot.
Példa: .G C000 (a $C000 címtől indítja a programot)
.G (a tárolt PC értéktől kezdve indítja a programot)

H (keresés)
Egy bájtsorozatot vagy ASCII-kódokkal tárolt szöveget kereshetünk a memóriában.
Példa: .H C000 CFFF A9 00
(az A9,00 bájtsorozatot keresi a $C000-tól $CFFF-ig terjedő területen)
.H E000 FFFF 'CBM80
(az ASCII-kódokkal tárolt 'CBM80 szöveget keresi a $E000-tól $FFFF-ig terjedő területen).

L (Load — programbetöltés)
Egy tárolt programot tölthetünk be a memóriába. A programok mindig arra a területre töltődnek, ahonnan ki lettek mentve.
Példa: .L "FEHER",08
(lemezről tölti be a "FEHER" nevű programot)
.L "FEHER",01
(kazettáról tölti be a "FEHER" néven tárolt programot)

M (Memory — memóriakijelzés)
Megvizsgálja a memória tartalmát. A kijelzett értékek felülírással, majd a RETURN billentyű leütésével módosíthatók.
Példa: .M C000 C01F
(kiírja a $C000-tól $C01F-ig terjedő memóriaterületet)

P (Print — programlistázás)
Kilistázunk egy megadott tartományt. Abban az esetben használjuk, amikor nem 22 utasítást akarunk visszafejteni, hanem megadott számút egy megadott memóriaterületről.
Példa: .P C000 C005
(a $C000 címen kezdődő és a $C005 címen végződő programot listázza ki)

R (Registers — regiszterek kijelzése)
Kijelzi azokat a tárolt regisztertartalmakat, amelyeket felülírhatunk. A G utasítás hatására az itt tárolt PC címtől kezdve indul a program, és a regiszterek is az itt tárolt kezdeti értékeket veszik fel.
Példa: .R

S (Save — programtárolás)
Egy programot ment ki lemezre vagy kazettára.
Példa: .S "FEHER",08,C000,C006
(lemezre menti "FEHER" névvel a $C000-tól $C005-ig terjedő memóriaterületet — a $C006 címet már nem)
.S "FEHER",01,C000,C006 (kazettás egységre tárol)

T (Transfer — adott memóriaterület átírása máshová)
Példa: .T C000 C005 C100
(a $C000 címtől a $C005 címig terjedő memóriatartományt átírja a $C100 címtől kezdődő területre)

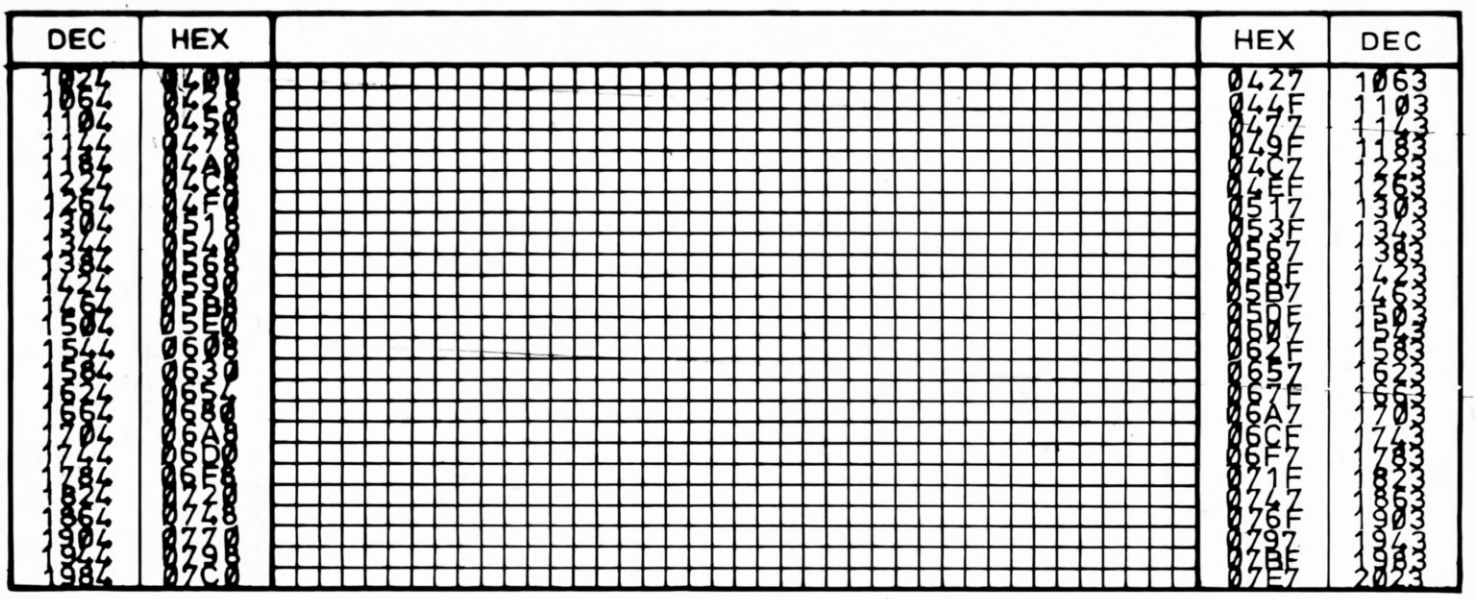
X (eXit — kilépés a monitorprogramból)
Kilépés a monitorprogramból, és visszatérés a Basicbe.
Példa: .X

A KÉPERNYŐ

A Commodore—64 képernyőjéről és különböző grafikus üzemmódjairól köteteket lehetne írni, ám ez messze meghaladná e könyv kereteit. A bekapcsolás utáni állapotban a képernyőn 25 sorban és 40 oszlopban összesen 1000 karakter helyezhető el. Mindegyik képernyő-pozíciónak megvan a memóriában a maga bájtja, ami meghatározza, hogy az adott képernyő-pozíción milyen karakter foglaljon helyet. Így a képernyőn egyszerre maximálisan 256 különböző fajta karakter lehet, mert egy bájton összesen 256-féle szám tárolható. Minden számhoz hozzá van rendelve egy karakter.
A VIC állandóan figyeli a $0400-tól $07E7-ig terjedő memóriaterületet (1024—2023), és attól függően, hogy az egyes bájtokon milyen számot talál, minden karakterpozícióra odahelyezi a számnak megfelelő karaktert. Alapállapotban tehát van 1000 képernyőcímünk, ahova adott számokat elhelyezve, adott karakterek jelennek meg.
A Commodore—64 használatakor 16 különböző színből válogathatunk, és ezekhez a színekhez számokat rendelünk:

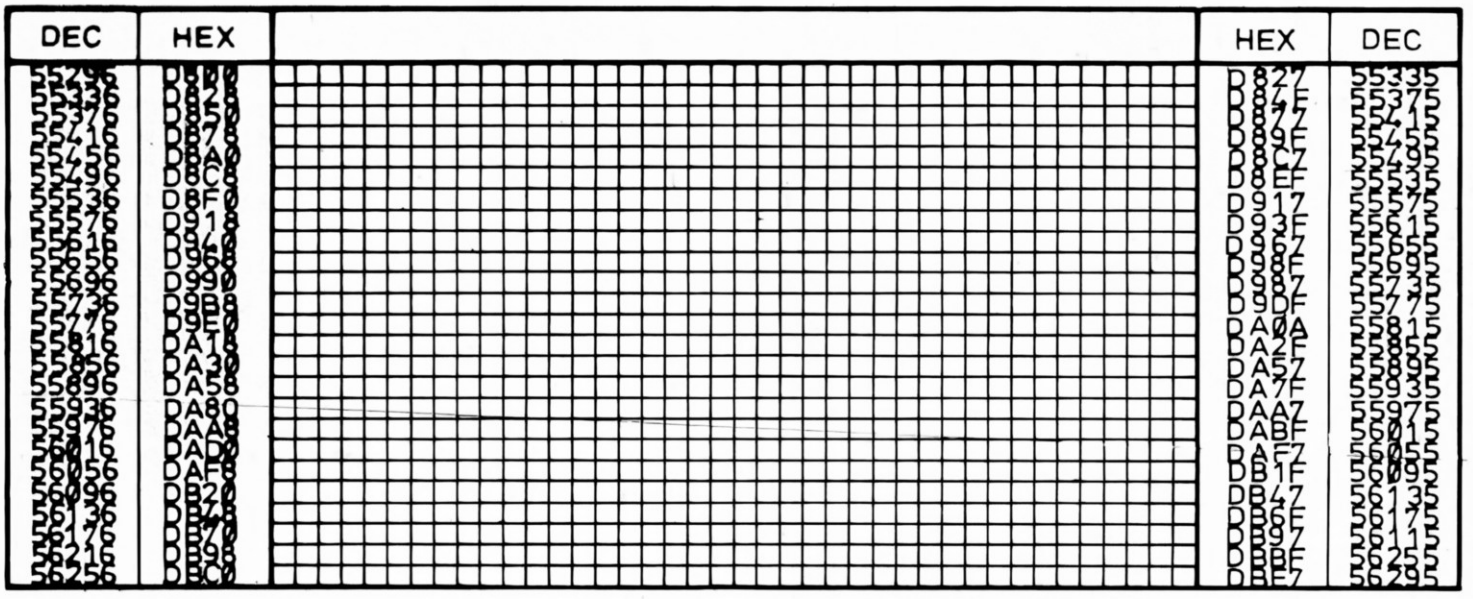
0 — fekete
1 — fehér
2 — piros
3 — cián
4 — bíbor
5 — zöld
6 — kék
7 — citromsárga
8 — narancssárga
9 — barna
A — halvány piros
B — sötétszürke
C — középszürke
D — világoszöld
E — világoskék
F — halvány szürke

Egy adott karakterhely színét úgy tudjuk meghatározni, hogy a hozzá tartozó színmemóriába beírjuk a szín kódját. Az egyes képernyő-pozíciókhoz ugyanis nemcsak a karakter kódját tartalmazó bájt tartozik, hanem egy olyan bájt is, ami a karakter színének kódját tartalmazza. Ez azt jelenti, hogy minden egyes pozícióhoz két bájt tartozik. A karakterkódokat tartalmazó bájtok $0400-tól $07E7-ig vannak (1024—2023).



DEC	HEX	HEX	DEC
1024	0400	0427	1063
1064	0428	044F	1103
1104	0450	0477	1143
1144	0478	049F	1183
1184	04A0	04C7	1223
1224	04C8	04EF	1263
1264	04F0	0517	1303
1304	0518	053F	1343
1344	0540	0567	1383
1384	0568	058F	1423
1424	0590	05B7	1463
1464	05B8	05DF	1503
1504	05E0	0607	1543
1544	0608	062F	1583
1584	0630	0657	1623
1624	0658	067F	1663
1664	0680	06A7	1703
1704	06A8	06CF	1743
1744	06D0	06F7	1783
1784	06F8	071F	1823
1824	0720	0747	1863
1864	0748	076F	1903
1904	0770	0797	1943
1944	0798	07BF	1983
1984	07C0	07E7	2023

A színkódokat tartalmazó bájtok $D800-tól $DBE7-ig (55296—56295).



DEC	HEX	HEX	DEC
55296	D800	D827	55335
55336	D828	D84F	55375
55376	D850	D877	55415
55416	D878	D89F	55455
55456	D8A0	D8C7	55495
55496	D8C8	D8EF	55535
55536	D8F0	D917	55575
55576	D918	D93F	55615
55616	D940	D967	55655
55656	D968	D98F	55695
55696	D990	D9B7	55735
55736	D9B8	D9DF	55775
55776	D9E0	DA07	55815
55816	DA08	DA2F	55855
55856	DA30	DA57	55895
55896	DA58	DA7F	55935
55936	DA80	DAA7	55975
55976	DAA8	DABF	56015
56016	DAD0	DAF7	56055
56056	DAF8	DB1F	56095
56096	DB20	DB47	56135
56136	DB48	DB6F	56175
56176	DB70	DB97	56215
56216	DB98	DBBF	56255
56256	DBC0	DBE7	56295

A háttér színét a $D021 (53281) címen, a keret színét pedig a $D020 (53280) címen levő szám határozza meg. Ezek után már gyerekjáték bármilyen színű szöveget kiírni a képernyőre, tetszőleges keret- és háttérszín-beállítással.

Van persze a képernyő kezelésének egy másik módja is. Ez a KERNAL ($FFD2 címen levő) CHROUT szubrutinja, ami úgy működik, hogy az A regiszterben tárolt szám CHR$-jét nyomtatja ki a képernyőre. Vigyázzunk azonban, mert ezzel a szubrutinnal nem tudunk a jobb alsó sarokba nyomtatni anélkül, hogy sort ne emelne. A jobb alsó sarokba való nyomtatáshoz továbbra is a képmemóriába kell írnunk, ha nem akarunk sort emelni.

A képernyőkódok távolról sem azonosak a CHR$-ekkel, nem ajánlatos a kettőt összekeverni! A karakterkódokhoz tartozó karaktereket a két különböző karakterkészletben a következő táblázat foglalja össze. (A $80–$FF mindkét készletben a $00–$7F inverzeit tartalmazza.)

ASCII és CHR$ értékek

Karaktereket megjelenítő kódok

	32=$20	@	64=$40	─	96=$60		160=$A0
!	33=$21	A	65=$41	♠	97=$61	▌	161=$A1
"	34=$22	B	66=$42	|	98=$62	▄	162=$A2
#	35=$23	C	67=$43	─	99=$63	▔	163=$A3
$	36=$24	D	68=$44	▔	100=$64	_	164=$A4
%	37=$25	E	69=$45	▔	101=$65	▏	165=$A5
&	38=$26	F	70=$46	─	102=$66	▒	166=$A6
'	39=$27	G	71=$47	|	103=$67	▏	167=$A7
(	40=$28	H	72=$48	|	104=$68	▒	168=$A8
)	41=$29	I	73=$49	╲	105=$69	◤	169=$A9
*	42=$2A	J	74=$4A	╲	106=$6A	▏	170=$AA
+	43=$2B	K	75=$4B	╱	107=$6B	├	171=$AB
,	44=$2C	L	76=$4C	└	108=$6C	▪	172=$AC
-	45=$2D	M	77=$4D	╲	109=$6D	└	173=$AD
.	46=$2E	N	78=$4E	╱	110=$6E	┐	174=$AE
/	47=$2F	O	79=$4F	┌	111=$6F	▬	175=$AF
0	48=$30	P	80=$50	┐	112=$70	┌	176=$B0
1	49=$31	Q	81=$51	●	113=$71	┴	177=$B1
2	50=$32	R	82=$52	─	114=$72	┬	178=$B2
3	51=$33	S	83=$53	♥	115=$73	┤	179=$B3
4	52=$34	T	84=$54	|	116=$74	▏	180=$B4
5	53=$35	U	85=$55	╱	117=$75	▌	181=$B5
6	54=$36	V	86=$56	╳	118=$76	▌	182=$B6
7	55=$37	W	87=$57	○	119=$77	▔	183=$B7
8	56=$38	X	88=$58	♣	120=$78	▔	184=$B8
9	57=$39	Y	89=$59	|	121=$79	▄	185=$B9
:	58=$3A	Z	90=$5A	♦	122=$7A	┘	186=$BA
;	59=$3B	[	91=$5B	+	123=$7B	▪	187=$BB
<	60=$3C	£	92=$5C	▒	124=$7C	▪	188=$BC
=	61=$3D	]	93=$5D	|	125=$7D	┘	189=$BD
>	62=$3E	↑	94=$5E	π	126=$7E	▪	190=$BE
?	63=$3F	←	95=$5F	◥	127=$7F	▚	191=$BF

A 192-223 kódok azonosak a 96-127 kódokkal.
A 224-254 kódok azonosak a 160-190 kódokkal.
A 255 kód azonos a 126 kóddal.

ASCII ÉS CHR$ ÉRTÉKEK
VEZÉRLŐ KARAKTEREK

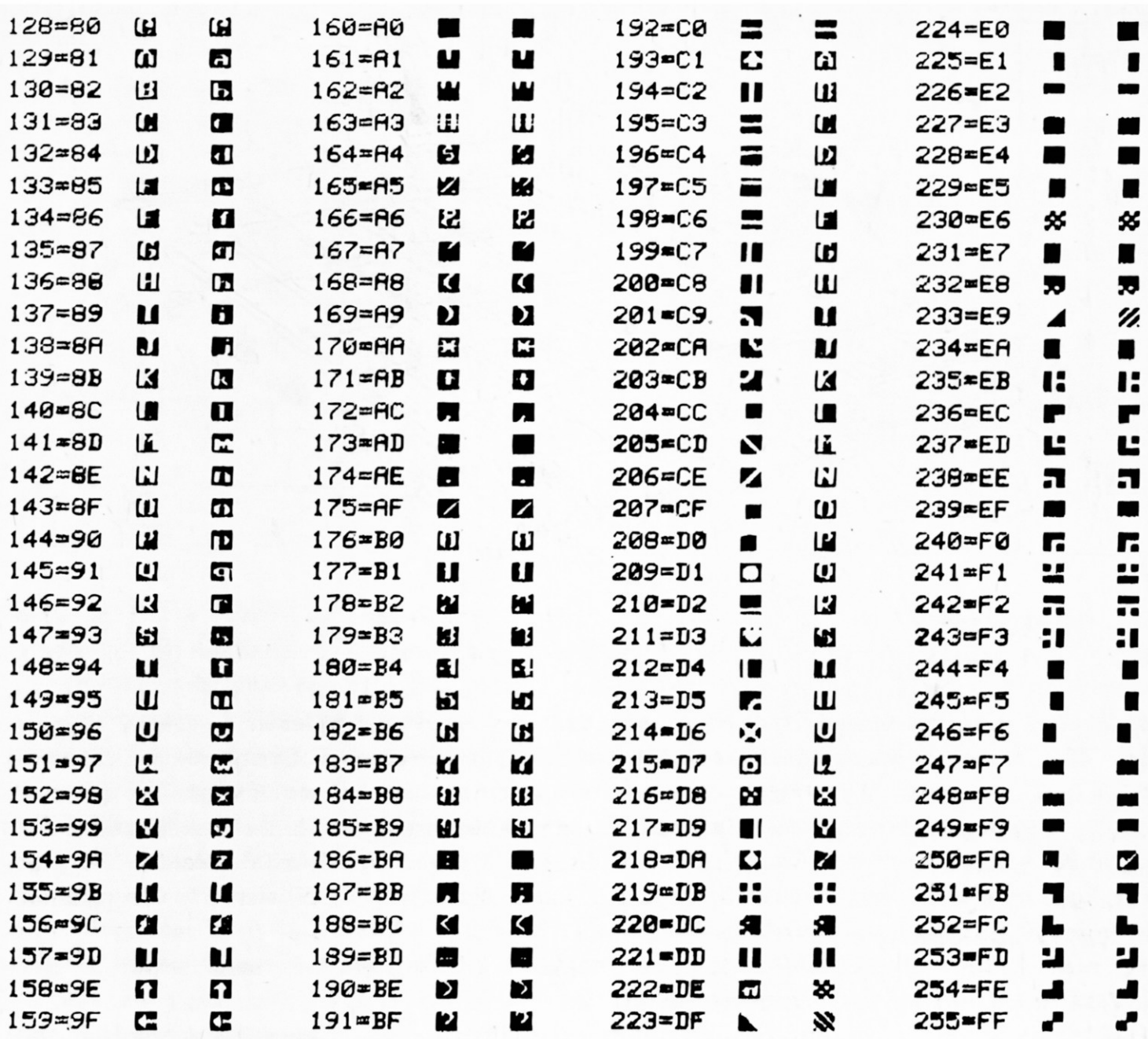
0	00	–
1	01	–
2	02	–
3	03	–
4	04	–
5	05	– fehér
6	06	–
7	07	–
8	08	– SHIFT C engedélyezése
9	09	– SHIFT C letiltása
10	0A	–
11	0B	–
12	0C	–
13	0D	– RETURN
14	0E	– átváltás kis/nagybetűre
15	0F	–
16	10	–
17	11	– CURSOR le
18	12	– RVS ON
19	13	– CLR/HOME – HOME
20	14	– INST/DEL – DEL
21	15	–
22	16	–
23	17	–
24	18	–
25	19	–
26	1A	–
27	1B	–
28	1C	– piros
29	1D	– CURSOR jobbra
30	1E	– zöld
31	1F	– kék
128	80	–
129	81	– narancssárga
130	82	–
131	83	–
132	84	–
133	85	– F1
134	86	– F3
135	87	– F5
136	88	– F7
137	89	– F2
138	8A	– F4
139	8B	– F6
140	8C	– F8
141	8D	– SHIFTELT RETURN
142	8E	– átváltás nagybetűs grafikusra
143	8F	–
144	90	– fekete
145	91	– CURSOR fel
146	92	– RVS OFF
147	93	– CLR/HOME – CLR
148	94	– INST/DEL – INST
149	95	– barna
150	96	– halványpiros

151 – 97 – szürke 1
152 – 98 – szürke 2
153 – 99 – világoszöld
154 – 9A – világoskék
155 – 9B – szürke 3
156 – 9C – bíbor
157 – 9D – CURSOR balra
158 – 9E – citromsárga
159 – 9F – cián

A KÉPERNYŐKÓDOKNAK MEGFELELŐ ÁBRÁK

Karakterkészletet a SHIFT és a Commodore billentyűk egyidejű lenyomásával válthatunk.

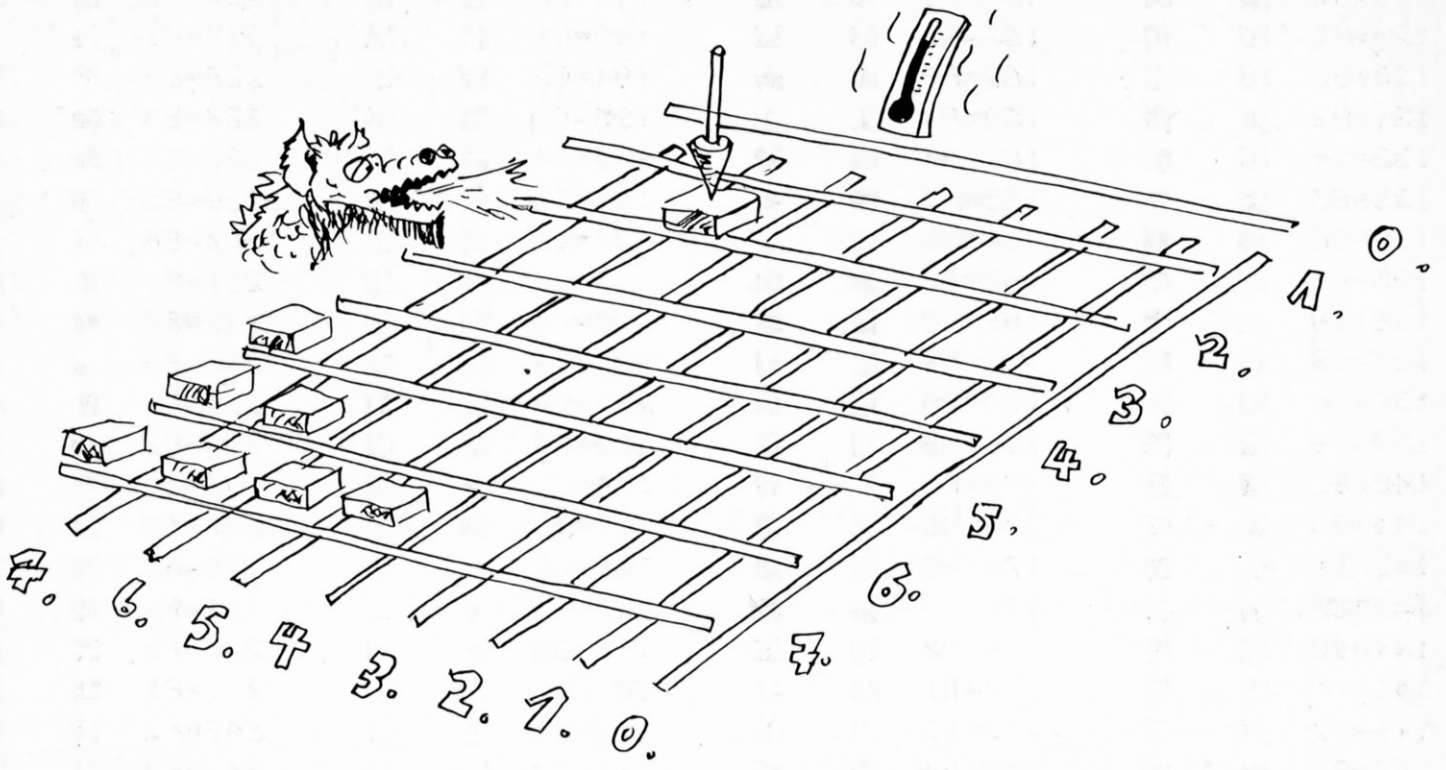
DEC=HEX	1	2	DEC=HEX	1	2	DEC=HEX	1	2	DEC=HEX	1	2
000=00	@	@	032=20			064=40	─	─	096=60		
001=01	A	a	033=21	!	!	065=41	♠	A	097=61	▌	▌
002=02	B	b	034=22	"	"	066=42	│	B	098=62	▄	▄
003=03	C	c	035=23	#	#	067=43	─	C	099=63	▔	▔
004=04	D	d	036=24	$	$	068=44	─	D	100=64	▁	▁
005=05	E	e	037=25	%	%	069=45	─	E	101=65	▏	▏
006=06	F	f	038=26	&	&	070=46	─	F	102=66	▒	▒
007=07	G	g	039=27	'	'	071=47	│	G	103=67	▕	▕
008=08	H	h	040=28	(	(	072=48	│	H	104=68	▚	▚
009=09	I	i	041=29	)	)	073=49	╮	I	105=69	◤	▨
010=0A	J	j	042=2A	*	*	074=4A	╰	J	106=6A	▕	▕
011=0B	K	k	043=2B	+	+	075=4B	╯	K	107=6B	├	├
012=0C	L	l	044=2C	,	,	076=4C	└	L	108=6C	▖	▖
013=0D	M	m	045=2D	-	-	077=4D	╲	M	109=6D	└	└
014=0E	N	n	046=2E	.	.	078=4E	╱	N	110=6E	┐	┐
015=0F	O	o	047=2F	/	/	079=4F	┌	O	111=6F	▁	▁
016=10	P	p	048=30	0	0	080=50	┐	P	112=70	┌	┌
017=11	Q	q	049=31	1	1	081=51	●	Q	113=71	┴	┴
018=12	R	r	050=32	2	2	082=52	▁	R	114=72	┬	┬
019=13	S	s	051=33	3	3	083=53	♥	S	115=73	┤	┤
020=14	T	t	052=34	4	4	084=54	│	T	116=74	▏	▏
021=15	U	u	053=35	5	5	085=55	╭	U	117=75	▌	▌
022=16	V	v	054=36	6	6	086=56	╳	V	118=76	▐	▐
023=17	W	w	055=37	7	7	087=57	○	W	119=77	▔	▔
024=18	X	x	056=38	8	8	088=58	✲	X	120=78	▔	▔
025=19	Y	y	057=39	9	9	089=59	│	Y	121=79	▁	▁
026=1A	Z	z	058=3A	:	:	090=5A	◆	Z	122=7A	▗	✓
027=1B	[	[	059=3B	;	;	091=5B	┼	┼	123=7B	▪	▪
028=1C	£	£	060=3C	<	<	092=5C	▚	▚	124=7C	▪	▪
029=1D	]	]	061=3D	=	=	093=5D	│	│	125=7D	▗	▗
030=1E	↑	↑	062=3E	>	>	094=5E	▝	▨	126=7E	▪	▪
031=1F	←	←	063=3F	?	?	095=5F	◥	▨	127=7F	▚	▚



A BILLENTYŰZET

A Commodore—64 gépen 64 olyan billentyű van, amelyek állapotát bármikor megvizsgálhatjuk. A billentyűzet ugyan 66 billentyűből áll, de a SHIFT LOCK ugyanazon a bemeneten jelentkezik, mint a bal oldali SHIFT billentyű, a RESTORE billentyűt pedig nem lehet olvasni, az egy megszakításkérő szerkezet. Csakhogy a RESTORE billentyű leütésekor egy olyan megszakítási rutin veszi át a hatalmat, ami nem csinál semmit, ha egyidejűleg a RUN/STOP billentyű nincs lenyomva. Így, ha csak egymagában a RESTORE billentyűt ütjük le, akkor elugrik ugyan a megszakításba, de mert a RUN/STOP nincs lenyomva, azonnal vissza is tér. Ha a RUN/STOP is le van nyomva, akkor folytatja a megszakítási rutint.

De térjünk vissza az olvasható billentyűkhöz. Képzeljünk el nyolc darab vízszintes és nyolc darab függőleges izzásképes drótot, amelyek nem érintkeznek egymással, csak elhaladnak egymás felett! Legyen minden kereszteződés fölött egy billentyű, melynek lenyomására a két egymás fölött elhaladó drót érintkezik.



Hogyan tudjuk megvizsgálni, hogy pl. a 2. számú vízszintes drót és az 5. számú függőleges drót kereszteződése fölött levő billentyű le van-e nyomva? Drótjaink rendkívül jó hővezetők. Kezdjük el melegíteni a 2. számú vízszintes drót bal oldali végét. Ha ez a drót egyik függőleges dróttal sem érintkezik, akkor kizárólag ő hevül. Ám ha bármely más dróthoz hozzáér, akkor már együtt hevülnek, mivel a hevített drót felmelegíti környezetét. Így, ha a vízszintes 2. és a függőleges 5. kereszteződésénél a billentyű le van nyomva, akkor a függőleges 5. is elkezd hevülni.

Ha melegítjük a vízszintes 2-t, és ennek hatására a függőleges 5 is hevül, akkor a kettő kereszteződésében le van nyomva a billentyű.

Hasonlóan, az összes billentyű állapotát megvizsgálhatjuk. Ha érdekel, hogy egy billentyű le van-e nyomva, akkor melegítjük azt a vízszintes drótot, amely elfut a billentyű alatt, és ha a billentyű alatt elfutó függőleges drót is melegszik, akkor a billentyű valószínűleg le van nyomva. Bármelyik billentyű állapotát meg tudjuk vizsgálni, csak azt kell tudnunk, hogy mely billentyűk mely kereszteződések fölött vannak.

A 187. oldalon levő táblázat megmutatja, hogy melyik billentyű melyik kereszteződési pontot jelenti a gép számára.

A drótok melegítése és vizsgálata a Commodore 64-ben az egyik CIA áramkör feladata, amellyel a $DC00–$DC03 (56320–56323) memóriacímeken keresztül társaloghatunk. Ő a memóriát nemcsak vizsgálni tudja a fal másik oldaláról, adatot is tud bele helyezni: a memóriát olvasni és írni is tudja. Társalgás előtt meg kell határoznunk, hogy mely címeken akarunk mi beszélni, és mely címek mely bitjein várjuk tőle a választ. Erre szolgálnak az adat–irány rekeszek. A CIA-val a $DC00 és $DC01 címeken keresztül beszélgethetünk, ehhez a két címhez tartozik még két cím, az adat–irány rekeszek. A $DC00-nak a $DC02 az adat–irány rekesze, a $DC01-nek pedig a $DC03.

Először tehát az adat–irány rekeszeket kell helyesen beállítani. Ha az adat–irány rekeszben valamelyik bit 1, akkor az I/O (Input/Output – bemenet/kimenet) rekeszben a neki megfelelő bit kimeneti lesz, így ezen a vonalon a billentyűzetkezelő felé áramlik az adat,

	7.	6.	5.	4.	3.	2.	1.	0.
0.	⇑ CRSR ⇓	15	13	11	17	⇐ CRSR ⇒	RETURN	INST DEL
1.	SHIFT BAL	E	S	Z	$ 4	A	W	# 3
2.	X	T	F	C	& 6	D	R	% 5
3.	V	U	H	B	< 8	G	Y	/ 7
4.	N	O	K	M	0	J	I	> 9
5.	< ,	@	[:	> .	-	L	P	+
6.	? /	↑	=	SHIFT JOBB	CLR HOME	] :	*	£
7.	RUN STOP	Q	C=	SPACE	" 2	CTRL	←	! 1

a parancs. Ha viszont az adat—irány rekeszben egy bit 0, akkor az I/O rekeszben a neki megfelelő bit bementi lesz, azaz ekkor a billentyűzetkezelőtől jön rajta adat. Nekünk jelen esetben a $DC00 rekesz teljes egészében kimeneti rekesz, így a neki megfelelő adat—irány rekeszbe, a $DC02 címre, $FF-et kell írni. A $DC01 rekeszt teljes egészében bemeneti rekesznek fogjuk használni, így a neki megfelelő adat—irány rekeszbe, a $DC03 címre, $00-t kell tenni.

A $DC00 címre kiadjuk, hogy melyik vízszintes drót(ok)at melegítse a billentyűzetkezelő, és a $DC01 címen jelenik majd meg, hogy ennek hatására melyik függőleges drótok hevülnek.

Egy vízszintes drót hevítésére úgy adunk parancsot, hogy a neki megfelelő bitet nullára állítjuk. Amennyiben nem akarjuk, hogy a drót hevüljön, úgy a neki megfelelő bitet 1-re állítjuk. Ha ezek után a $DC01 címen egy bit nulla lesz, akkor az azt jelentí, hogy hevül a neki megfelelő függőleges drót, ha pedig 1, akkor azt, hogy nem.

Gépi kódú billentyűzetolvasásnál mindig tiltsuk le a megszakításokat, mert az eredeti rendszermegszakítás is tartalmaz egy billentyűzetolvasó rutint, és ez változtathat a regiszterek tartalmán!

Nézzünk most egy példaprogramot, amely a Commodore billentyű állapotát vizsgálja.

Először tiltsuk le a megszakításokat!

```
cím    kód       utasítás
---    ---       --------
C000   78        SEI
```

A $DC00 rekesznek megfelelő adat—irány rekeszt beállítjuk úgy, hogy minden bit kimeneti legyen:

```
cím    kód       utasítás
---    ---       --------
C006   A9 00     LDA #$00
C008   8D 03 DC  STA $DC03
```

A $DC01 regiszternek megfelelő adat—irány rekeszt beállítjuk úgy, hogy minden bit bemeneti legyen:

```
cím    kód       utasítás
---    ---       --------
C001   A9 FF     LDA #$FF
C003   8D 02 DC  STA $DC02
```

Parancsot adunk a 7. számú vízszintes drót hevítésére, mivel a drót felett helyezkedik el a Commodore billentyű. A többi drótot pedig hidegen hagyjuk ($7F=b01111111).

```
cím     kód        utasítás
---     ---        --------
C00B  A9 7F        LDA #$7F
C00D  8D 00 DC     STA $DC00
```

Betöltjük A-ba a függőleges drótok állapotát jelző rekeszt:

```
cím     kód        utasítás
---     ---        --------
C010  AD 01 DC     LDA $DC01
```

Kizárólag az 5. számú függőleges drót érdekel, a többi bitet nullázzuk. Ha az 5. függőleges drót most nem izzott, akkor az 5. bit 1 volt, így az AND művelet után Z=0.

```
cím     kód        utasítás
---     ---        --------
C013  29 20        AND #$20
```

Ha a Commodore billentyű nem volt lenyomva, újra vizsgálunk:

```
cím     kód        utasítás
---     ---        --------
C015  D0 F9        BNE $C010
```

A végére tehát csak akkor juthat el a program, ha a Commodore billentyű le volt nyomva. Engedélyezzük újból a megszakításokat, és térjünk vissza a Basicbe! (SUPER 64-MON-ból való hívás esetén RTS helyett BRK-t kell írnunk!)

```
cím     kód        utasítás
---     ---        --------
C017  58           CLI
C018  60           RTS
```

Programunk tehát mindaddig vár, míg a Commodore billentyűt le nem nyomjuk, majd visszatér. Hasonló módon bármely billentyű állapotát megvizsgálhatjuk.
Programunk betöltője:

```
100 FOR I=0 TO 24
110 READ A
120 POKE 49152+I,A
130 NEXT I
140 DATA 120,169,255,141,2,220,169,0,141,3
150 DATA 220,169,127,141,0,220,173,1,220,41
160 DATA 32,208,249,88,96
```

Az eredeti rendszer is tartalmaz billentyűzetolvasó szubrutint, ám ott más történik. Ha le van nyomva egy billentyű, akkor annak ASCII kódja bekerül a billentyűzetpufferbe, ahonnan egy újabb szubrutin segítségével az A regiszterbe tehető. Így:

JSR $FF97 — vizsgálja a billentyűzetet (KERNAL SCNKEY rutin)
JSR $FFE4 — kiveszi a pufferből az A-ba (KERNAL GETIN rutin)

Ha nem volt lenyomott billentyű, akkor az A-ban $00 lesz, egyébként pedig a lenyomott billentyű ASCII kódja.
A $FF97 címen kezdődő SCNKEY rutint a rendszer minden IRQ megszakítás alkalmával, azaz másodpercenként kb. 60-szor meghívja, nekünk a billentyűzet olvasásakor csupán a

$FFE4 címen kezdődő GETIN rutinnal kell foglalkoznunk. Ha ezt a rutint meghívjuk, akkor a pufferből az A regiszterbe kerül annak a billentyűnek az ASCII kódja, amelyik le volt ütve a billentyűzetet vizsgáló szubrutin működése közben. A puffer maximálisan 10 karakter hosszú billentyűsorozatot tud tárolni, és híváskor ezeket a GETIN rutin a leütések sorrendjében tölti az A-ba. Így, ha a pufferben van 8 karakter, akkor nyolcszor kell meghívni a $FFE4 címen levő GETIN rutint, hogy a puffer újból üres legyen.
És most térjünk rá a botkormányok olvasására! A botkormányok szintén a $DC00 és a $DC01 címeket használják, ám ebben az esetben mindkét rekesz bemeneti rekesz kell hogy legyen! A botkormányok állapota a következő módon jelenik meg:

a $DC00 címen a CONTROL PORT 2-be csatlakoztatott botkormányé,
a $DC01 címen a CONTROL PORT 1-be csatlakoztatott botkormányé.

Mindkét címen csupán a 4., 3., 2., 1. és 0. biteket kell vizsgálni. Ezek jelentése:

4. bit – TŰZ
3. bit – JOBBRA
2. bit – BALRA
1. bit – LE
0. bit – FEL

Amennyiben valamelyik bit nulla, akkor a megfelelő irányban el van mozdítva a botkormány, ha pedig egy bit 1, úgy abban az irányban nem lett elmozgatva. Természetesen a rézsútos elmozgatásoknál egyszerre két irány jelződik.

PÉLDAPROGRAMOK

1. példa
Írjunk egy olyan programot, mely kiolvassa a lemez hibacsatornáját. Erre az alábbi Basic megoldás kínálkozik:

```
100 OPEN 15,8,15
110 INPUT# 15,A,B$,C,D
120 PRINT A,B$,C,D
130 CLOSE 15
140 END
```

Ezzel egyenértékű az a megoldás, amely karakterenként olvassa be az adatokat:

```
100 OPEN 15,8,15
110 GET# 15,A$
120 IF A$=CHR$(13) THEN 150
130 PRINT A$;
140 GOTO 110
150 CLOSE 15
160 END
```

Ahhoz, hogy ezt a programot gépi kódban írjuk meg, először meg kell nyitnunk a 15-ös csatornát. Ehhez kell az OPEN rutin, de előtte még meg kell hívni a SETLFS és SETNAM rutinokat is. Először hívjuk meg a SETLFS rutint!

```
cím    kód        utasítás       megjegyzés
---    ---        --------       ----------
C000   A9 0F      LDA #$0F       az állomány száma=15
C002   A2 08      LDX #$08       egységszám=8
C004   A0 0F      LDY #$0F       csatornaszám=15
C006   20 BA FF   JSR $FFBA      KERNAL SETLFS rutin
```

Mivel jelen esetben nincs állománynév, a SETNAM rutin hívása előtt csak az A regisztert kell nullázni, ezzel jelezve, hogy az állománynév hossza nulla karakter.

```
cím    kód        utasítás       megjegyzés
---    ---        --------       ----------
C009   A9 00      LDA #$00       nincs állománynév
C00B   20 BD FF   JSR $FFBD      KERNAL SETNAM rutin
```

Megnyithatjuk az állományt.

```
cím    kód        utasítás       megjegyzés
---    ---        --------       ----------
C00E   20 C0 FF   JSR $FFC0      KERNAL OPEN rutin
```

Az állományt adatbememententi állományként akarjuk használni, ezért adatbemeneteti állományként kell definiálni a CHKIN rutinnal.

```
cím    kód        utasítás       megjegyzés
---    ---        --------       ----------
C011   A2 0F      LDX #$0F       az állomány száma=15
C013   20 C6 FF   JSR $FFC6      KERNAL CHKIN rutin
```

Az állományból karaktereket hozunk be. Olvassunk be egy karaktert, és nézzük meg, hogy nem RETURN-e!

```
cím    kód        utasítás       megjegyzés
---    ---        --------       ----------
C016   20 CF FF   JSR $FFCF      KERNAL CHRIN rutin
C019   C9 0D      CMP #$0D       megnézzük, hogy 13-e.
```

Amennyiben a behozott karakter RETURN volt, legyen vége a programnak!

```
cím    kód        utasítás       megjegyzés
---    ---        --------       ----------
C01B   F0 06      BEQ $C023
```

Ha nem RETURN volt, akkor nyomtassuk ki a CHROUT rutin segítségével a képernyőre! (Ehhez nem kellett sem az OPEN, sem a többi előkészítő rutin.)

```
cím    kód        utasítás       megjegyzés
---    ---        --------       ----------
C01D   20 D2 FF   JSR $FFD2      KERNAL CHROUT rutin
```

Menjünk el a következő karakterért!

```
cím    kód        utasítás       megjegyzés
---    ---        --------       ----------
C020   4C 16 C0   JMP $C016
```

Ha a behozott karakter RETURN volt, akkor zárjuk le az állományt a CLALL rutinnal, és térjünk vissza a Basicbe!

```
cím    kód        utasítás       megjegyzés
---    ---        --------       ----------
C023   20 E7 FF   JSR $FFE7      KERNAL CHALL rutin
C026   60         RTS            visszatérés
```

A program betöltője:

```
100 FOR I=0 TO 38
110 READ A
120 POKE 49152+I,A
130 NEXT I
140 DATA 169,15,162,8,160,15,32,186,255
150 DATA 169,0,32,189,255,32,192,255,162
160 DATA 15,32,198,255,32,207,255,201,13
170 DATA 240,6,32,210,255,76,22,192,32,231,255,96
```

Ha ezek után SYS 49152-vel meghívjuk a rutint, akkor kinyomtatja a képernyőre a 8-as egységszámú lemezmeghajtó egység hibaüzenetét.

2. példa

Hozzunk létre a lemezegységen egy szekvenciális állományt, majd írjuk bele a számokat 1-től 9-ig! A megoldás Basicben:

```
100 OPEN 5,8,7,"SZAMOK,S,W"
110 FOR I=1 TO 9
120 PRINT# 5,I
130 NEXT I
140 CLOSE 5
150 END
```

Először létre kell hoznunk az állományt, hogy utána dolgozhassunk vele. Ám itt az állománynak már van neve: "SZAMOK,S,W", ezt a szöveget kell valahol a memóriában tárolnunk az ASCII kódjaival, hogy a SETNAM rutint meg tudjuk hívni. Legyen tehát ez a szöveg a memóriában a $C040 címtől kezdődően. Hozzuk létre az állományt!

cím	kód	utasítás	megjegyzés
C000	A9 05	LDA #$05	az állomány száma=5
C002	A2 08	LDX #$08	egységszám=8
C004	A0 07	LDY #$07	csatornaszám=7
C006	20 BA FF	JSR $FFBA	KERNAL SETLFS rutin

Adjunk nevet az állománynak! Mivel az állomány neve 10 karakter hosszú, ezért:

cím	kód	utasítás	megjegyzes
C009	A9 0A	LDA #$0A	az állománynév 10 karakter

Az állomány neve a memóriában megegyezésünk szerint a $C040 címtől kezdődően lesz tárolva, tehát az X és Y regisztereket ezek szerint kell beállítani:

cím	kód	utasítás	megjegyzés
C00B	A2 40	LDX #$40	alsó bájt=$40
C00D	A0 C0	LDY #$C0	felső bájt=$c0
C00F	20 BD FF	JSR $FFBD	KERNAL SETNAM rutin

Ha már neve is van az állománynak, akkor megnyithatjuk.

cím	kód	utasítás	megjegyzés
C012	20 C0 FF	JSR $FFC0	KERNAL OPEN rutin

Mivel írni akarunk az állományba, ezért adatkimeneti állományként kell definiálni.

cím	kód	utasítás	megjegyzés
C015	A2 05	LDX #$05	az állomány száma=5
C017	20 C9 FF	JSR $FFC9	KERNAL CHKOUT rutin

Kiírhatjuk az állományba a számokat 1-től 9-ig. Először vegyük az 1-et!

```
cím     kód       utasítás
---     ---       --------
C01A  A2 01       LDX #$01
```

A számokat a lemezen az ASCII kódjaikkal kell tárolni, adjunk hozzá a számhoz $30-t, hogy az ASCII kódját kapjuk. Ez persze már csak az A regiszterben történhet, másoljuk tehát át az A-ba!

```
cím     kód       utasítás      megjegyzés
---     ---       --------      ----------
C01C  8A          TXA
C01D  18          CLC
C01E  69 30       ADC #$30      ASCII érték kiszámolása
```

Most már kivihetjük a karaktert a lemezre.

```
cím     kód       utasítás      megjegyzés
---     ---       --------      ----------
C020  20 D2 FF    JSR $FFD2     KERNAL CHROUT rutin
```

Vegyük most a $13 kódú RETURN karaktert, küldjük ki azt is, jelezve, hogy vége a számnak.

```
cím     kód       utasítás      megjegyzés
---     ---       --------      ----------
C023  A9 0D       LDA #$0D      RETURN karakter
C025  20 D2 FF    JSR $FFD2     KERNAL CHROUT rutin
```

Jöhet a következő szám!

```
cím     kód       utasítás      megjegyzés
---     ---       --------      ----------
C028  E8          INX
```

Ha ez a szám még nem tíz, ezt küldjük ki, egyébként pedig folytassuk tovább a programot!

```
cím     kód       utasítás      megjegyzés
---     ---       --------      ----------
C029  E0 0A       CPX #$0A      10-e a szám?
C02B  D0 EF       BNE $C01C     ha nem 10, akkor vissza
```

Ha a szám már 10, akkor ne küldjük ki, hanem zárjuk le az állományt, és térjünk vissza.

```
cím     kód       utasítás      megjegyzés
---     ---       --------      ----------
C02D  20 CC FF    JSR $FFCC     KERNAL CLRCHN rutin
C030  A9 05       LDA #$05      az állomány száma=5
C032  20 C3 FF    JSR $FFC3     KERNAL CLOSE rutin
C035  60          RTS           visszatérés
```

Természetesen ahhoz, hogy a programunk jól működjön, az állomány nevét tárolnunk kell a $C040 címtől kezdődően.

```
.;C040 53 5A 41 4D 4F 4B 2C 53
.;C048 2C 57 00 00 00 00 00 00
```

A betöltő:

```
100 FOR I=0 TO 73
110 READ A
120 POKE 49152+I,A
130 NEXT I
140 DATA 169,5,162,8,160,7,32,186,255,169
150 DATA 10,162,64,160,192,32,189,255,32,192
160 DATA 255,162,5,32,201,255,162,1,138,24
```

```
170 DATA 105,48,32,210,255,169,13,32,210,255
180 DATA 232,224,10,208,239,32,204,255,169,5
190 DATA 32,195,255,96,0,0,0,0,0,0
200 DATA 0,0,0,0,83,9C,65,77,79,75
210 DATA 44,83,44,87
```

Ez a program is SYS 49152 utasítással indítható. Futtatás után az alábbi Basic programmal ellenőrizhetjük az eredményt:

```
100 OPEN 5,8,7,"SZAMOK,S,R"
110 FOR I=1 TO 9
120 INPUT# 5,A
130 PRINT A
140 NEXT I
150 CLOSE 5
160 END
```

3. példa

A kazettás egység kezelése. Lássuk először a Basic programot!

```
100 OPEN 2,1,2,"UZENET"
110 PRINT# 2,"EZ EGY UZENET MAGNOSOKNAK."
120 FOR I=2 TO 6
130 PRINT# 2,I
140 NEXT I
150 CLOSE 2
160 END
```

Az „UZENET" szöveget tároljuk a $C000-tól $C006-ig terjedő területen, magát az üzenetet pedig a $C008 és $C021 között. Tároljuk a kiküldendő adatokat is, mégpedig a $C028-tól $C02C-ig terjedő területen. Természetesen mindhárom szöveget ASCII kódokkal kell tárolni. Az adatok a memóriában az alábbi módon helyezkednek el:

```
.;C000 55  5A  45  4E  45  54  00  00
       (U  Z   E   N   E   T )
.;C008 45  5A  20  45  47  59  20  55
       (E  Z       E   G   Y       U
.;C010 5A  45  4E  45  54  20  4D  41
       Z   E   N   E   T       M   A
.;C018 47  4E  4F  53  4F  4B  4E  41
       G   N   O   S   O   K   N   A
.;C020 4B  2E  00  00  00  00  00  00
       K   .   )
.;C028 32  33  34  35  36  00  00  00
       (2  3   4   5   6 )
```

A program kezdődjék a $C030 címen azzal, hogy létrehozzuk és megnyitjuk az állományt!

```
cím    kód        utasítás      megjegyzés
---    ---        --------      ----------
C030   A9 02      LDA #$02      az állomány száma=2
C032   A2 01      LDX #$01      egységszám=1
C034   A0 02      LDY #$02      a másodlagos cím=2
C036   20 BA FF   JSR $FFBA     KERNAL SETLFS rutin
C039   A9 06      LDA #$06      az állomány neve 6 karakter
C03B   A2 00      LDX #$00      a név a $C000 címen
C03D   A0 C0      LDY #$C0      kezdődik
C03F   20 BD FF   JSR $FFBD     KERNAL SETNAM rutin
C042   20 C0 FF   JSR $FFC0     KERNAL OPEN rutin
```

Mivel adatokat akarunk kivinni, az állományt adatkimeneti állományként kell definiálni.

```
cím     kód        utasítás       megjegyzés
---     ---        --------       ----------
C045  A2 02        LDX #$02       az állomány száma=2
C047  20 C9 FF     JSR $FFC9      KERNAL CHKOUT rutin
```

Küldjük ki karakterenként a szöveget!

```
cím     kód        utasítás       megjegyzés
---     ---        --------       ----------
C04A  A2 00        LDX #$00
C04C  BD 08 C0     LDA $C008,X
C04F  20 D2 FF     JSR $FFD2      KERNAL CHROUT rutin
C052  E8           INX
C053  E0 1A        CPX #$1A
C055  D0 F5        BNE $C04C
```

A szöveget egy RETURN karakterrel zárjuk le!

```
cím     kód        utasítás       megjegyzés
---     ---        --------       ----------
C057  A9 0D        LDA #$0D       a RETURN ASCII kódja
C059  20 D2 FF     JSR $FFD2      KERNAL CHROUT rutin
```

Küldjük ki a számokat úgy, hogy mindegyik után kiküldünk egy RETURN karaktert is!

```
cím     kód        utasítás       megjegyzés
---     ---        --------       ----------
C05C  A2 00        LDX #$00
C05E  BD 28 C0     LDA $C028,X
C061  20 D2 FF     JSR $FFD2      KERNAL CHROUT rutin
C064  A9 0D        LDA #$0D       a RETURN kódja
C066  20 D2 FF     JSR $FFD2      KERNAL CHROUT rutin
C069  E8           INX
C06A  E0 05        CPX #$05
C06C  D0 F0        BNE $C05E
```

Mivel minden adatot kiküldtünk, zárjuk le az állományt!

```
cím     kód        utasítás       megjegyzés
---     ---        --------       ----------
C06E  20 CC FF     JSR $FFCC      KERNAL CLRCHN rutin
C071  A9 02        LDA #$02       az állomány száma=2
C073  20 C3 FF     JSR $FFC3      KERNAL CLOSE rutin
C076  60           RTS
```

A betöltő:

```
100 FOR I=0 TO 118
110 READ A
120 POKE 49152+I,A
130 NEXT I
140 DATA 85,90,69,78,69,84,0,0,69,90
150 DATA 32,69,71,89,32,85,90,69,78,69
160 DATA 84,32,77,65,71,78,79,83,79,75
170 DATA 78,65,75,46,0,0,0,0,0,0
180 DATA 50,51,52,53,54,0,0,0,169,2
190 DATA 162,1,160,2,32,186,255,169,6,162
200 DATA 0,160,192,32,189,255,32,192,255,162
210 DATA 2,32,201,255,162,0,189,8,192,32
220 DATA 210,255,232,224,26,208,245,169,13,32
230 DATA 210,255,162,0,189,40,192,32,210,255
240 DATA 169,13,32,210,255,232,224,5,208,240
250 DATA 32,204,255,169,2,32,195,255,96
```

Programunk SYS 49200-zal hívható, és futtatás után a következő Basic programmal ellenőrizhető:

```
100 OPEN 2,1,0,"UZENET"
110 INPUT# 2,A$
120 PRINT A$
130 FOR I=1 TO 5
140 INPUT# 2,A
150 PRINT A
160 NEXT I
170 CLOSE 2
180 END
```

4. példa

Nyomtassuk ki azt a szöveget, hogy „COMMODORE 64", de ne csak egyszer, hanem kilencszer, egy kicsit eltolva. Ez Basicben nem túl bonyolult:

```
100 OPEN 4,4
110 PRINT# 4,CHR$(15);
120 FOR Y=1 TO 9
130 FOR X=Y TO 1 STEP -1
140 PRINT# 4," ";
150 NEXT X
160 PRINT# 4,"COMMODORE 64"
170 NEXT Y
180 CLOSE 4
```

Az egyszerűség kedvéért a kinyomtatandó szöveget most is tároljuk a memóriában, mégpedig a $C000-tól $C00B-ig terjedő területen!

```
.;C000 43  4F  4D  4D  4F  44  4F  52
       (C  O   M   M   O   D   O   R
.;C009 45  20  36  34  00  00  00  00
       E       6   4   )
```

A program a $C010 címen kezdődik.

cím	kód	utasítás	megjegyzés
C010	A9 04	LDA #$04	az állomány száma=4
C012	A2 04	LDX #$04	egységszám=4
C014	A0 FF	LDY #$FF	nincs másodlagos cím
C016	20 BA FF	JSR $FFBA	KERNAL SETLFS rutin
C019	A9 00	LDA #$00	nincs állománynév
C01B	20 BD FF	JSR $FFBD	KERNAL SETNAM rutin
C01E	20 C0 FF	JSR $FFC0	KERNAL OPEN rutin
C021	A2 04	LDX #$04	az állomány száma=4
C023	20 C9 FF	JSR $FFC9	KERNAL CHKOUT rutin

Megnyitottuk a nyomtatót négyes egységszámmal, és mielőtt elkezdenénk nyomtatni, váltsuk standard karakteres üzemmódba. Ez Basicben egy PRINT# 4,CHR$(15), gépi kódban viszont:

cím	kód	utasítás	megjegyzés
C026	A9 0F	LDA #$0F	
C028	20 D2 FF	JSR $FFD2	KERNAL CHROUT rutin

Kezdjük el ciklusunkat Y=1 kezdeti értékkel!

cím	kód	utasítás	megjegyzés
C02B	A0 01	LDY #$01	

Nyomtassunk ki annyi szóközt, amennyi az Y értéke:

```
cím    kód        utasítás      megjegyzés
---    ---        --------      ----------
C02D   98         TYA
C02E   AA         TAX
C02F   A9 20      LDA #$20      szóköz ASCII kódja
C031   20 D2 FF   JSR $FFD2     KERNAL CHROUT rutin
C034   CA         DEX
C035   D0 FA      BNE $C031
```

és nyomtassuk ki, hogy COMMODORE 64!

```
cím    kód        utasítás      megjegyzés
---    ---        --------      ----------
C037   A2 00      LDX #$00
C039   BD 00 C0   LDA $C000,X
C03C   20 D2 FF   JSR $FFD2     KERNAL CHROUT rutin
C03F   E8         INX
C040   E0 0C      CPX #$0C
C042   D0 F5      BNE $C039
```

Küldjünk ki végül egy RETURN karaktert is!

```
cím    kód        utasítás      megjegyzés
---    ---        --------      ----------
C044   A9 0D      LDA #$0D
C046   20 D2 FF   JSR $FFD2     KERNAL CHROUT rutin
```

Növeljük meg az Y értékét, és ha még nem tíz, hajtsuk végre újból a ciklust!

```
cím    kód        utasítás      megjegyzés
---    ---        --------      ----------
C049   C8         INY
C04A   C0 0A      CPY #$0A
C04C   D0 DF      BNE $C02D
```

Ha már végigmentünk kilencszer a programrészen, zárjuk le a nyomtatót, és legyen vége a programnak!

```
cím    kód        utasítás      megjegyzés
---    ---        --------      ----------
C04E   20 CC FF   JSR $FFCC     KERNAL CLRCHN rutin
C051   A9 04      LDA #$04      az állomány száma=4
C053   20 C3 FF   JSR $FFC3     KERNAL CLOSE rutin
C056   60         RTS
```

A betöltő:

```
100 FOR I=0 TO 86
110 READ A
120 POKE 49152+I,A
130 NEXT I
140 DATA 67,79,77,77,79,68,79,82,69,32
150 DATA 54,52,0,0,0,0,169,4,162,4
160 DATA 160,255,32,186,255,169,0,32,189,255
170 DATA 32,192,255,162,4,32,201,255,169,15
180 DATA 32,210,255,160,1,152,170,169,32,32
190 DATA 210,255,202,208,250,162,0,189,0,192
200 DATA 32,210,255,232,224,12,208,245,169,13
210 DATA 32,210,255,200,192,10,208,223,32,204
220 DATA 255,169,4,32,195,255,96
```

Programunkat SYS 49168-cal futtathatjuk.

5. példa

Mozgattassunk egy csillagot botkormány segítségével a képernyőn (az átlós esetekkel nem foglalkozunk). A botkormány olvasására Basicből csak a PEEK utasítás segítségével van

lehetőség, ezért kezdjük rögtön a gépi kódú programmal. A programból a KERNAL PLOT és CHROUT rutinjait fogjuk hívni.
Először is állítsuk be a kurzort a képernyő közepére: a PLOT rutin számára adjuk meg az Y=19 és X=12 kezdeti értékeket!

```
cím    kód       utasítás      megjegyzés
---    ---       --------      ----------
C000   A2 0C     LDX #$0C      a sorszám legyen 12
C002   A0 13     LDY #$13      az oszlopszám legyen 19
```

Mielőtt továbblépnénk, töröljük a képernyőt!

```
cím    kód       utasítás      megjegyzés
---    ---       --------      ----------
C004   A9 93     LDA #$93      CHR$(147)
C006   20 D2 FF  JSR $FFD2     KERNAL CHROUT rutin
```

Hívjuk meg a PLOT rutint úgy, hogy a kurzor a meghatározott helyre kerüljön!

```
cím    kód       utasítás      megjegyzés
---    ---       --------      ----------
C009   18        CLC
C00A   20 F0 FF  JSR $FFF0     KERNAL PLOT rutin
```

Helyezzünk el a képernyőn egy csillagot oda, ahol a kurzor van!

```
cím    kód       utasítás      megjegyzés
---    ---       --------      ----------
C00D   A9 2A     LDA #$2A      a csillag ASCII kódDja
C00F   20 D2 FF  JSR $FFD2     KERNAL CHROUT rutin
```

Nézzük meg a control port 1-be illesztett botkormány állapotát! Az egyes irányokat vizsgáljuk végig, és azok szerint ágaztassunk el a megfelelő címekre. A tűz gomb lenyomása esetén térjen vissza a program!

```
cím    kód       utasítás      megjegyzés
---    ---       --------      ----------
C012   AD 01 DC  LDA $DC01
C015   4A        LSR
C016   90 0D     BCC $C025     irány=fel
C018   4A        LSR
C019   90 12     BCC $C02D     irány=le
C01B   4A        LSR
C01C   90 17     BCC $C035     irány=bal
C01E   4A        LSR
C01F   90 1C     BCC $C03D     irány=jobb
C021   4A        LSR
C022   B0 EE     BCS $C012     ha a tűz gomb nincs lenyomva
C024   60        RTS
```

Ha az irány=fel, akkor (ha lehet) csökkentsük X értékét

```
cím    kód       utasítás      megjegyzés
---    ---       --------      ----------
C025   E0 00     CPX #$00
C027   F0 E9     BEQ $C012
C029   CA        DEX
C02A   4C 42 C0  JMP $C042
```

Ha az irány=le, akkor (ha lehet) növeljük X értékét!

```
cím    kód       utasítás      megjegyzés
---    ---       --------      ----------
C02D   E0 18     CPX #$18
C02F   F0 E1     BEQ $C012
C031   E8        INX
C032   4C 42 C0  JMP $C042
```

Ha az irány=bal, akkor csökkentsük Y értékét — ha még nem nulla!

```
cím     kód         utasítás        megjegyzés
---     ---         --------        ----------
C035  C0 00         CPY #$00
C037  F0 D9         BEQ $C012
C039  88            DEY
C03A  4C 42 C0      JMP $C042
```

Ha az irány=jobb, akkor növeljük Y értékét! Ezt is csak akkor, ha még nincs a maximális értéken.

```
cím     kód         utasítás        megjegyzés
---     ---         --------        ----------
C03D  C0 27         CPY #$27
C03F  F0 D1         BEQ $C012
C041  C8            INY
```

Ha a kurzor nem a jobb alsó pozíción van, akkor hajtsuk újból végre a rutint. Töröljük a képernyőt, és írassunk ki egy csillagot!

```
cím     kód         utasítás        megjegyzés
---     ---         --------        ----------
C042  C0 27         CPY #$27
C044  D0 BE         BNE $C004
C046  E0 18         CPX #$18
C048  D0 BA         BNE $C004
```

Ha a kurzor a jobb alsó pozíción van, akkor ne a PLOT és CHROUT rutinnal nyomtassuk ki a csillagot! A CHROUT rutin (mint az őt használó Basic PRINT utasítás használatakor láthatjuk) a sorok utolsó pozíciójának kitöltése után sort emel. Ha ezzel írnánk ki az utolsó sor utolsó pozíciójára a csillagot, akkor az egész képernyő feljebb ugrana.

```
cím     kód         utasítás        megjegyzés
---     ---         --------        ----------
C04A  A9 93         LDA #$93
C04C  20 D2 FF      JSR $FFD2       KERNAL CHROUT rutin
C04F  A9 2A         LDA #$2A
C051  8D E7 07      STA $07E7
C054  4C 12 C0      JMP $C012
```

A betöltő:

```
100 FOR I=0 TO 86
110 READ A
120 POKE 49152+I,A
130 NEXT I
140 DATA 162,12,160,19,169,147,32,210,255,24
150 DATA 32,240,255,169,42,32,210,255,173,1
160 DATA 220,74,144,13,74,144,18,74,144,23
170 DATA 74,144,28,74,176,238,96,224,0,240
180 DATA 233,202,76,66,192,224,24,240,225,232
190 DATA 76,66,192,192,0,240,217,136,76,66
200 DATA 192,192,39,240,209,200,192,39,208,190
210 DATA 224,24,208,186,169,147,32,210,255,169
220 DATA 42,141,231,7,76,18,192
```

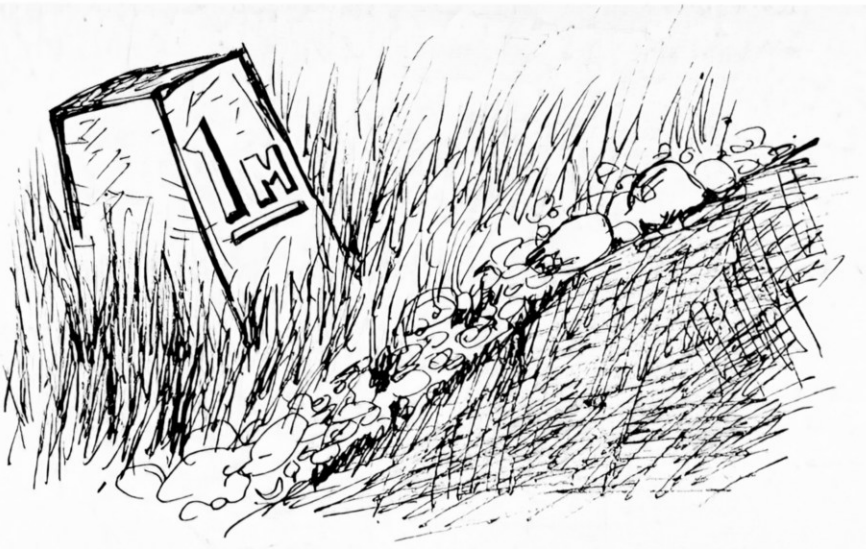
A programot SYS 49152-vel hívhatjuk.

JÁTÉKPROGRAM

Minden programozó pályafutása során komoly mérföldkövet jelent az első játékprogram megírása. Gépi kód esetén ez még inkább így van. Írjunk egy játékprogramot véges végig gépi kódban!

Programunk egy blokádjáték, amellyel sok örömet szerezhetünk családtagjainknak. Két játékos játssza, kell hozzá két botkormány. A játékosoknak leleményes módon kell egymást minél kisebb területre beszorítaniuk. Az veszít, aki előbb ütközik a falnak. Ekkor a győztes indulásának helyén megjelenik egy négyzet (jelezve, hogy ő a győztes), s lehet újból játszani. A játék akkor indul, ha mindkét játékos egyszerre lenyomja a „tűz" gombot.

A játék SYS 49160-nal ($C008) indítható, a $C000–$C007 területet pedig a program adatainak tárolására használjuk. Egy játékos helyzetét mindig az X és Y koordinátájával



jelezzük, a bal felső négyzet mindkét koordinátája nulla. Mindkét játékoshoz tartozik tehát egy X és egy Y érték, de ebből még nem tudhatjuk, hogy hova fog lépni. Ez mindig a botkormány helyzetétől függ. Így nekünk azt is tárolnunk kell, hogy mennyit kell majd az aktuális X és Y értékekhez hozzáadni egy-egy lépéskor. Ez azt jelenti, hogy mindkét játékoshoz tartozik négy adat, ez összesen nyolc. Ezeket az adatokat fogjuk tárolni a $C000-tól $C007-ig terjedő területen, az alábbi módon:

$C000 — Az első játékos helyzetének X koordinátája.
$C001 — Az első játékos X koordinátájához hozzáadandó szám.
$C002 — Az első játékos helyzetének Y koordinátája.
$C003 — Az első játékos Y koordinátájához hozzáadandó szám.
$C004 — A második játékos helyzetének X koordinátája.
$C005 — A második játékos X koordinátájához hozzáadandó szám.
$C006 — A második játékos helyzetének Y koordinátája.
$C007 — A második játékos Y koordinátájához hozzáadandó szám.

Állítsuk be a keret és a háttér színét feketére!

```
cím     kód         utasítás
---     ---         --------
C008   A9 00        LDA #$00
C00A   8D 20 D0     STA $D020
C00D   8D 21 D0     STA $D021
```

A karakterek a képernyőn sárga színűek. Ehhez ki kell küldenünk egy $9E ASCII kódot a KERNAL CHROUT rutin segítségével.

```
cím     kód         utasítás
---     ---         --------
C010   A9 9E        LDA #$9E
C012   20 D2 FF     JSR $FFD2
```

Mielőtt bármi történne, töröljük a képernyőt!

```
cím     kód         utasítás
---     ---         --------
C015   A9 93        LDA #$93
C017   20 D2 FF     JSR $FFD2
```

Rajzoljunk fel a képernyőre egy keretet. Ezen belül mozognak majd a játékosok. Felhasználjuk a $C108 címen kezdődő szubrutint, amely egy adott X és Y értékkel meghíva ki-

nyomtat egy inverz szóközt a képernyőre. Ez a rutin azonban az Y értékét tekinti X koordinátának, és az X értékét Y koordinátának, így ennek megfelelő módon kell hívni.

```
cím    kód        utasítás
---    ---        --------
C01A   A2 18      LDX #$18
C01C   A0 00      LDY #$00
C01E   20 08 C1   JSR $C108
C021   A0 27      LDY #$27
C023   20 08 C1   JSR $C108
C026   CA         DEX
C027   10 F3      BPL $C01C
C029   A0 27      LDY #$27
C02B   A2 00      LDX #$00
C02D   20 08 C1   JSR $C108
C030   A2 18      LDX #$18
C032   20 08 C1   JSR $C108
C035   88         DEY
C036   10 F3      BPL $C02B
```

Induljon az első játékos arról a pontról, amelynek mindkét koordinátája kettő!

```
cím    kód        utasítás
---    ---        --------
C038   A9 02      LDA #$02
C03A   8D 00 C0   STA $C000
C03D   8D 02 C0   STA $C002
```

A második játékos induljon arról a pontról, melynek X koordinátája 37, Y koordinátája 22!

```
cím    kód        utasítás
---    ---        --------
C040   A9 25      LDA #$25
C042   8D 04 C0   STA $C004
C045   A9 16      LDA #$16
C047   8D 06 C0   STA $C006
```

Kezdetben mindkét játékos függőlegesen mozog.

```
cím    kód        utasítás
---    ---        --------
C04A   A9 00      LDA #$00
C04C   8D 01 C0   STA $C001
C04F   8D 05 C0   STA $C005
```

Az első játékos lefelé fog elindulni.

```
cím    kód        utasítás
---    ---        --------
C052   A9 01      LDA #$01
C054   8D 03 C0   STA $C003
```

A második játékos felfelé fog elindulni, így az Y koordinátájához −1-et kell hozzáadni.

```
cím    kód        utasítás
---    ---        --------
C057   A9 FF      LDA #$FF
C059   8D 07 C0   STA $C007
```

Jelezzük ki a kezdőállapotot mindkét játékos esetében a $C108 címen kezdődő rutin segítségével!

```
cím    kód        utasítás
---    ---        --------
C05C   AE 02 C0   LDX $C002
C05F   AC 00 C0   LDY $C000
C062   20 08 C1   JSR $C108
C065   AE 06 C0   LDX $C006
C068   AC 04 C0   LDY $C004
C06B   20 08 C1   JSR $C108
```

Várjunk addig, amíg mindketten egyszerre le nem nyomják a „tűz" gombot!

```
cím     kód        utasítás
---     ---        --------
C06E  AD 00 DC     LDA $DC00
C071  0D 01 DC     ORA $DC01
C074  29 10        AND #$10
C076  D0 F6        BNE $C06E
```

A játéknak el kell indulnia, tehát foglalkozzunk először az első játékossal! Nézzük meg, milyen irányban akar haladni! Ehhez először ismernünk kell a botkormány állapotát.

```
cím     kód        utasítás
---     ---        --------
C078  AD 00 DC     LDA $DC00
```

A botkormány bizonyos állapotaihoz bizonyos irányok tartoznak. Meg kell hívnunk egy szubrutint, amely az A regiszterben tárolt értéktől függően beállítja az X és Y regisztereket. Ezeket aztán tárolhatjuk a hozzáadandó értékeket tartalmazó címekre. A mi szubrutinunk olyan, hogy ha egy irány sincs jelezve, akkor nem tesz az X és Y regiszterbe semmit. Biztonsági okokból tehát töltsük fel az X és Y regisztert a hozzáadandó értékekkel, és csak utána hívjuk meg a rutint!

```
cím     kód        utasítás
---     ---        --------
C07B  AE 01 C0     LDX $C001
C07E  AC 03 C0     LDY $C003
C081  20 EB C0     JSR $C0EB
```

Az X és Y regiszterekben most megvan, hogy mennyit kell a koordinátákhoz hozzáadni. Ha a botkormány nem jelöl egyetlen irányt sem, akkor ezek megegyeznek az előbbi értékekkel. Tároljuk ezeket az adott címeken!

```
cím     kód        utasítás
---     ---        --------
C084  8E 01 C0     STX $C001
C087  8C 03 C0     STY $C003
```

Nézzük a mozgatást! Először adjuk hozzá a koordinátákhoz, amit kell!

```
cím     kód        utasítás
---     ---        --------
C08A  8A           TXA
C08B  18           CLC
C08C  6D 00 C0     ADC $C000
C08F  8D 00 C0     STA $C000
C092  98           TYA
C093  18           CLC
C094  6D 02 C0     ADC $C002
C097  8D 02 C0     STA $C002
```

Már tudjuk tehát, hogy hová kerül a következő lépésben a játékos. Most kell megvizsgálnunk, hogy szabad-e még az a hely, vagy már foglalt. Ezt is egy szubrutinnal intézzük el, amely az A regiszterbe teszi az adott karakterpozíción levő karakter kódját. Ez a rutinunk is megfordítva kéri a koordinátákat, azaz az Y regiszterben az X koordinátát, és az X regiszterben az Y koordinátát.

```
cím     kód        utasítás
---     ---        --------
C09A  AE 02 C0     LDX $C002
C09D  AC 00 C0     LDY $C000
C0A0  20 2D C1     JSR $C12D
```

Ha az adott pozíción nem szóköz volt, akkor az első játékos veszített, mert falba ütközött. Ebben az esetben ugorjunk el arra a programrészre, mely kijelzi a mérkőzés eredményét, egyébként pedig menjünk tovább!

```
cím     kód         utasítás
---     ---         --------
C0A3  C9 20         CMP #$20
C0A5  F0 03         BEQ $C0AA
C0A7  4C 62 C1      JMP $C162
```

Ha a játékos nem ütközött falba, akkor nyomtassunk ki a pozíciójára egy inverz szóközkaraktert!

```
cím     kód         utasítás
---     ---         --------
C0AA  20 08 C1      JSR $C108
```

Mielőtt a második játékossal foglalkoznánk, meg kell hívnunk egy késleltető rutint. Erre azért van szükség, mert nélküle a program futása túlzottan gyors lenne, és élvezhetetlen lenne a játék.

```
cím     kód         utasítás
---     ---         --------
C0AD  20 50 C1      JSR $C150
```

Most pedig foglalkozzunk hasonló módon a második játékossal is! A programrész majdnem azonos az előbbivel, a különbség csak ott van, hogy ez a második játékos adataival dolgozik, és ütközéskor az első játékost jelzi ki győztesnek. Most tehát a másik botkormány helyzetétől függően adjuk meg a hozzáadandó értékeket!

```
cím     kód         utasítás
---     ---         --------
C0B0  AD 01 DC      LDA $DC01
C0B3  AE 05 C0      LDX $C005
C0B6  AC 07 C0      LDY $C007
C0B9  20 EB C0      JSR $C0EB
C0BC  8E 05 C0      STX $C005
C0BF  8C 07 C0      STY $C007
```

Adjuk hozzá a koordinátákhoz a hozzáadandó értékeket!

```
cím     kód         utasítás
---     ---         --------
C0C2  8A            TXA
C0C3  18            CLC
C0C4  6D 04 C0      ADC $C004
C0C7  8D 04 C0      STA $C004
C0CA  98            TYA
C0CB  18            CLC
C0CC  6D 06 C0      ADC $C006
C0CF  8D 06 C0      STA $C006
```

Ha az adott pozíción nem szóköz van, akkor az első játékos győz.

```
cím     kód         utasítás
---     ---         --------
C0D2  AE 06 C0      LDX $C006
C0D5  AC 04 C0      LDY $C004
C0D8  20 2D C1      JSR $C12D
C0DB  C9 20         CMP #$20
C0DD  F0 03         BEQ $C0E2
C0DF  4C 5B C1      JMP $C15B
```

Amennyiben nem volt ütközés, jelezzük ki az adott pozícióra a játékost, késleltessünk, és foglalkozzunk újból az első játékossal!

```
cím     kód        utasítás
---     ---        --------
C0E2  20 08 C1     JSR $C108
C0E5  20 50 C1     JSR $C150
C0E8  4C 78 C0     JMP $C078
```

Programunk még nincs készen, hiszen ahhoz, hogy az egyes szubrutinokat meghívhassuk, meg is kell őket írni. Jöjjön tehát a botkromány helyzetét kiértékelő rutin! A rutin nem foglalkozik az átlós irányokkal, ilyenkor mindig az adott irány vízszintes vetületét adja meg. Az A regisztert csúsztassuk bitenként jobbra, és minden egyes csúsztatás után vizsgáljuk a C jelzőbit állapotát! Ha a C jelzőbit értéke nulla, akkor az X és Y regisztereket feltöltjük az adott iránynak megfelelő értékekkel, egyébként pedig továbbhaladunk. Nézzük először a felfelé irányt!

```
cím     kód        utasítás
---     ---        --------
C0EB  4A           LSR
C0EC  B0 04        BCS $C0F2
C0EE  A2 00        LDX #$00
C0F0  A0 FF        LDY #$FF
```

Vizsgáljuk meg a lefelé irányt!

```
cím     kód        utasítás
---     ---        --------
C0F2  4A           LSR
C0F3  B0 04        BCS $C0F9
C0F5  A2 00        LDX #$00
C0F7  A0 01        LDY #$01
```

A következő irány a balra:

```
cím     kód        utasítás
---     ---        --------
C0F9  4A           LSR
C0FA  B0 04        BCS $C100
C0FC  A2 FF        LDX #$FF
C0FE  A0 00        LDY #$00
```

Az utolsó lehetséges irány a jobbra:

```
cím     kód        utasítás
---     ---        --------
C100  4A           LSR
C101  B0 04        BCS $C107
C103  A2 01        LDX #$01
C105  A0 00        LDY #$00
```

Ha már minden irányt megvizsgáltunk, akkor térjünk vissza!

```
cím     kód        utasítás
---     ---        --------
C107  60           RTS
```

A $C108 címen annak a szubrutinnak kell kezdődnie, amely egy adott pozícióra egy inverz szóközt nyomtat ki. Itt célszerű az (indirekt),Y címzési módot alkalmazni, és a $22 és $23 nulladik lapon levő címeket használni. A cím úgy alakul ki, hogy $0400-hoz hozzá kell adnunk negyvenszer az X regiszter értékét. Ezt a címet az Y regiszterrel indexelve pontosan a kívánt címet kapjuk. Az X regiszter értékét úgy adjuk hozzá a címhez negyvenszer, hogy először hozzáadjuk a nyolcszorosát, majd a harminckétszeresét is. Először tehát legyen a cím $0400:

```
cím     kód        utasítás
---     ---        --------
C108  A9 04        LDA #$04
C10A  85 23        STA $23
```

A cím alsó bájtja legyen az X regiszter nyolcszorosa. Mivel az X regiszter értéke legföljebb 24 lehet, így a nyolcszorosa is elfér egy bájton, nem kell tehát a felső bájtra is figyelnünk. Nyolccal úgy szorzunk, hogy elvégzünk három ASL utasítást.

```
 cím     kód        utasítás
 ---     ---        --------
C10C   8A          TXA
C10D   0A          ASL
C10E   0A          ASL
C10F   0A          ASL
C110   85 22       STA $22
```

A cím alsó bájtjához hozzá kell még adni az X harminckétszeresét, de itt már foglalkoznunk kell a magas bájttal is. A harminckétszeres a nyolcszorosból két ASL utasítással képezhető.

```
 cím     kód        utasítás
 ---     ---        --------
C112   0A          ASL
C113   0A          ASL
C114   18          CLC
C115   65 22       ADC $22
C117   85 22       STA $22
C119   A5 23       LDA $23
C11B   69 00       ADC #$00
C11D   85 23       STA $23
```

Az X harminckétszerese azonban nem fér el egy bájton. Az ASL utasítások következtében lemaradt a felső öt bit. Ezeket tehát hozzá kell még adni a felső bájthoz, de előtte még az A regiszter alsó öt bitjére kell tolnunk őket. Ezt könnyen elvégezhetjük három LSR utasítás segítségével.

```
 cím     kód        utasítás
 ---     ---        --------
C11F   8A          TXA
C120   4A          LSR
C121   4A          LSR
C122   4A          LSR
C123   18          CLC
C124   65 23       ADC $23
C126   85 23       STA $23
```

Megvan már a cím, amihez hozzá kellene adni az Y értékét. Ha nem adjuk hozzá, hanem inkább indexelünk az Y-nal, akkor megtakarítunk egy összeadást. Tegyünk az indexelt címre 160-at, az inverz szóköz képernyőkódját:

```
 cím     kód        utasítás
 ---     ---        --------
C128   A9 A0       LDA #$A0
C12A   91 22       STA ($22),Y
C12C   60          RTS
```

Kell lennie persze olyan szubrutinnak is, amely egy adott pozícióról behozza az adott pozíción levő karakter képernyőkódját. Ez a rutin a cím kiszámítását ugyanúgy végzi, mint az előző rutin, csak a végén nem STA utasítás van, hanem LDA.

```
 cím     kód        utasítás
 ---     ---        --------
C12D   A9 04       LDA #$04
C12F   85 23       STA $23
C131   8A          TXA
C132   0A          ASL
C133   0A          ASL
```

```
C134  0A        ASL
C135  85 22     STA $22
C137  0A        ASL
C138  0A        ASL
C139  18        CLC
C13A  65 22     ADC $22
C13C  85 22     STA $22
C13E  A5 23     LDA $23
C140  69 00     ADC #$00
C142  85 23     STA $23
C144  8A        TXA
C145  4A        LSR
C146  4A        LSR
C147  4A        LSR
C148  18        CLC
C149  65 23     ADC $23
C14B  85 23     STA $23
C14D  B1 22     LDA ($22),Y
C14F  60        RTS
```

Szükségünk van egy késleltető programra, amely a késleltetést két egymásba ágyazott ciklussal végzi. A szubrutin olyan egyszerű, hogy nem szükséges hozzá magyarázat. A késleltetés idejét Basicből egy POKE 49489,szám utasítással megváltoztathatjuk, és így vezérelhetjük a program gyorsaságát.

```
cím    kód       utasítás
---    ---       --------
C150  A9 80      LDA #$80
C152  AA         TAX
C153  A8         TAY
C154  CA         DEX
C155  D0 FD      BNE $C154
C157  88         DEY
C158  D0 FA      BNE $C154
C15A  60         RTS
```

Hátra van még az eredmény kijelzése. Ez úgy történik, hogy a győztes játékos kiindulási pozícióján megjelenik egy inverz szóköz, majd bizonyos idő után újból elindul a program. A programnak két belépési pontja van. Az első belépési pont meghívására az első játékost jelzi ki győztesnek.

```
cím    kód       utasítás
---    ---       --------
C15B  A2 02      LDX #$02
C15D  A0 02      LDY #$02
C15F  4C 66 C1   JMP $C166
```

A második belépési pont hívására a második játékost jelzi ki győztesnek.

```
cím    kód       utasítás
---    ---       --------
C162  A2 16      LDX #$16
C164  A0 25      LDY #$25
```

A rutin először törli a képernyőt, majd az adott X és Y pozícióra kinyomtat egy inverz szóközt.

```
cím    kód       utasítás
---    ---       --------
C166  A9 93      LDA #$93
C168  20 D2 FF   JSR $FFD2
C16B  20 08 C1   JSR $C108
```

Mielőtt újból elindulna az egész program, még ötször meghívjuk a késleltető rutint, hogy a győztes játékosnak legyen ideje gyönyörködni győzelme eredményében.

```
 cím     kód         utasítás
 ---     ---         --------
C16E   20 50 C1     JSR $C150
C171   20 50 C1     JSR $C150
C174   20 50 C1     JSR $C150
C177   20 50 C1     JSR $C150
C17A   20 50 C1     JSR $C150
C17D   4C 08 C0     JMP $C008
```

Ezzel készen is vagyunk. A program persze még kezdetleges, nincsenek benne hangjelzések, nem paraméterezhető, de fut. Bárki bővítheti ízlése szerint, de vigyáznia kell arra, hogy a JMP és JSR utasítások a bővítés során megváltozott címekre ágaztassanak el.
A programot jelenlegi állapotában memóriába töltő Basic program az alábbi:

```
100 S=0
110 FOR I=0 TO 375 : READ A : S=S+A
120 POKE 49160+I,A
130 NEXT I : IF S<>39570 THEN PRINT "HIBA VOLT !"
140 DATA 169,0,141,32,208,141,33,208,169,158
150 DATA 32,210,255,169,147,32,210,255,162,24
160 DATA 160,0,32,8,193,160,39,32,8,193
170 DATA 202,16,243,160,39,162,0,32,8,193
180 DATA 162,24,32,8,193,136,16,243,169,2
190 DATA 141,0,192,141,2,192,169,37,141,4
200 DATA 192,169,22,141,6,192,169,0,141,1
210 DATA 192,141,5,192,169,1,141,3,192,169
220 DATA 255,141,7,192,174,2,192,172,0,192
230 DATA 32,8,193,174,6,192,172,4,192,32
240 DATA 8,193,173,0,220,13,1,220,41,16
250 DATA 208,246,173,0,220,174,1,192,172,3
260 DATA 192,32,235,192,142,1,192,140,3,192
270 DATA 138,24,109,0,192,141,0,192,152,24
280 DATA 109,2,192,141,2,192,174,2,192,172
290 DATA 0,192,32,45,193,201,32,240,3,76
300 DATA 98,193,32,8,193,32,80,193,173,1
310 DATA 220,174,5,192,172,7,192,32,235,192
320 DATA 142,5,192,140,7,192,138,24,109,4
330 DATA 192,141,4,192,152,24,109,6,192,141
340 DATA 6,192,174,6,192,172,4,192,32,45
350 DATA 193,201,32,240,3,76,91,193,32,8
360 DATA 193,32,80,193,76,120,192,74,176,4
370 DATA 162,0,160,255,74,176,4,162,0,160
380 DATA 1,74,176,4,162,255,160,0,74,176
390 DATA 4,162,1,160,0,96,169,4,133,35
400 DATA 138,10,10,10,133,34,10,10,24,101
410 DATA 34,133,34,165,35,105,0,133,35,138
420 DATA 74,74,74,24,101,35,133,35,169,160
430 DATA 145,34,96,169,4,133,35,138,10,10
440 DATA 10,133,34,10,10,24,101,34,133,34
450 DATA 165,35,105,0,133,35,138,74,74,74
460 DATA 24,101,35,133,35,177,34,96,169,128
470 DATA 170,168,202,208,253,136,208,250,96,162
480 DATA 2,160,2,76,102,193,162,22,160,37
490 DATA 169,147,32,210,255,32,8,193,32,80
500 DATA 193,32,80,193,32,80,193,32,80,193
510 DATA 32,80,193,76,8,192
```

A betöltő lefuttatása után a programot SYS 49160 utasítással futtathatjuk. Jó szórakozást!

TÁRGYMUTATÓ

A sorozat eddig megjelent kötetei:

1. Bodor T.–Gerő P.: Alapismeretek	148,–
2. Bodor Tibor: Soros lemezállományok	133,–
3. Bodor Tibor: Közvetlen elérésű lemezállományok	137,–

Egyéb könyvajánlatunk:

Bodor Tibor: A Basic programozás technikája	60,–
dr. Kocsis András: TV Basic	120,–
TV–Informatika	144,–
Csépai J.–Quittner P.: Bit–les	125,–
Allaga Gyula: Bűbájt (Mikromese)	120,–
Seymour Papert: Észrengés	160,–

Előkészületben:

Angster E.–Béres E.: Turbo Pascal
Gordon–Körtvélyesi–Sós–Székely: Pascal programozási nyelv
Ada-Winter Péter: Simon's Basic példatár
Hosszú F.–Lukács O.: Matek szigorlat (feladatgyűjtemény)

Könyveink megvásárolhatóak az ÁKV, a Művelt Nép, a Könyvért könyvesboltjaiban és a SZÜV C-M szakboltjaiban. Postai utánvéttel megrendelhetők a Számalk kiadói főosztályán: 1502 Bp. 112 Pf. 146., valamint a Műszaki Könyváruházban: 1061 Bp., Liszt F. tér 9.

Keresse ingyenes könyvismertetőinket;
olvassa szakkönyveinket!

Ára: 198.- Ft