

MOLNÁR LAJOS

Amit a
C64-esről
tudni érdemes

Novotrade Kiadó kft.



Molnár Lajos

Amit a C64-esről tudni érdemes...

**A C64-es felépítése és
programozása**

Novotrade Kiadó kft.

Szerző: Molnár Lajos

Lektorálta: dr. Nagy Gábor

Szerkesztő: Lukács Erzsébet

A kiadásért felel Siba László a Novotrade Kiadó kft. ügyvezető igazgatója

© Novotrade Kiadó kft., 1993

ISBN 963 583 180 4

E könyv írója a könyv és a benne levő programok készítése során a legjobb tudása szerint, nagy gondossággal járt el. Ennek ellenére hibák előfordulása mégsem kizárható. Ezért a könyv írója és kiadója semmiféle felelősséget nem vállal.

Minden jog fenntartva. Jelen könyvet, ill. annak részeit a kiadó engedélye nélkül tilos bármilyen formátumban vagy eszközzel reprodukálni, tárolni és közölni.

Készült a Kertészeti és Élelmiszeripari Egyetem Házinyomdájában
Felelős vezető: Wilpert Gábor nyomdavezető

Tartalomjegyzék

Bevezető	7
1. A C64-es hardvere	9
1.1. A processzor	12
1.1.1. A 6510 I/O portjai és belső regiszterei	13
1.1.2. A processzor felépítése és működése	14
1.1.3. A processzor nyelve	16
1.2. A videojel-generátor (VIC)	18
1.2.1. A VIC regiszterei	21
1.2.2. A VIC ábrázolási lehetőségei és a sprite-kezelés	23
1.2.3. A VIC címzési megoldásai	31
1.2.4. A VIC megszakítási technikája	33
1.2.5. A VIC programozása	36
1.3. Complex Interface Adapter (CIA)	57
1.3.1. A CIA regiszterei	58
1.3.2. A CIA1 feladatai	62
1.3.3. A CIA2 feladatai	64
1.3.4. A CIA programozása	70
1.4. A hanggenerátor (SID)	81
1.4.1. A SID regiszterei	82
1.4.2. A SID programozása	85
1.4.3. Az analóg/digitális átalakítók	92
1.5. A tárkezelő (AM)	95
1.6. A memória	97
1.6.1. Az írható-olvasható táruk: RAM	97
1.6.2. Csak olvasható memória: ROM	110
1.7. A csatlakozók	122
2. A 6510 processzor nyelve és programozása	127
2.1. A gépi nyelv	127

2.1.1.	Az assembler	128
2.1.2.	A processzor címezsmódjai	129
2.1.3.	A processzor utasításkészlete	131
2.2.	A processzor utasításai	132
2.3.	Gépi kódú programozás, assembly programozás	180
2.3.1.	A programok ellenőrzése, a MOVMON monitorprogram	182
2.3.2.	Gyakorlati programozás	184
2.3.3.	Gépi kódú programok a gyakorlatban	190
3.	A BASIC	197
3.1.	A Commodore BASIC 2.0 alapelemei	198
3.1.1.	Változók és állandók	199
3.1.2.	Kifejezések	201
3.1.3.	Értékadás	202
3.1.4.	Utasításcsoportok	203
3.2.	A BASIC 2.0 utasításai	205
3.3.	Az interpreter hibaüzenetei	224
3.4.	Tár- és változókezelés	227
3.5.	Perifériák és kezelésük	234
3.5.1.	A kazettás egység	235
3.5.2.	A lemezegység	239
	A lemezegység tárolási rendszere	240
	A tartalomjegyzék	242
	A BAM	242
	A blokkterkép kialakítása	243
	A file-bejegyzések	245
3.5.3.	File-szervezés, file-szerkezet	248
3.5.4.	A DOS kezelése	250
3.5.5.	Közvetlen lemezkezelés	258
3.5.6.	Közvetlen memóriakezelés	262
3.5.7.	File-kezelés	265
4.	A BASIC és a gépi nyelv ötvözése	271
4.1.	Hasznos rutinok	272
4.1.1.	INPUT-rutin	272
4.1.2.	Beolvasás állományból — a GET# megkerülése	281
4.1.3.	Karakterlánc-generátor	282
4.1.4.	Intarziás keresés adathalmazban	283
4.1.5.	Pseudo MID\$ — a standard utasítás módosítása	285
4.1.6.	Az USR függvény	288
4.1.7.	Univerzális menü	289

4.1.8. Magyar ékezetes karakterek	294
4.1.9. A képernyőgörgetés módosítása	298
4.1.10. A LIST utasítás módosítása	300
4.1.11. A változók kiírása	302
4.1.12. A BASIC-program visszaállítása	304
4.1.13. A képernyő tartalmának kiírása — HARDCOPY	305
4.1.14. Önálló utasítások előállítása — hardcopy parancsmódban	307
4.1.15. Tömbváltozók szelektív törlése — programba építhető BASIC-bővítés	310
4.2. BASIC-változók kezelése gépi kódban	314
4.2.1. Változók definiálása	315
4.2.2. Változók átalakítása	316
4.3. Néhány speciális BASIC program	320
4.3.1. Tárcímszerkesztő	320
4.3.2. Speciális gépi kód-szimulátor	321
4.3.3. BASIC és gépi kód — autostart program	326
4.3.4. Programgenerátor programból — a BASIC-betöltők előállítása	327
4.4. A ROM-ok felhasználhatósága	329
4.5. BASIC programozás során használható trükkök és megoldások	340
Függelék	345
A számítástechnika fejlődése	345
A számítógépek működésének logikai alapjai — logikai algebra	346
A C64-es logikai algebrája	349
A C64-esen alkalmazott számrendszerek	350
A BCD-kód	351
Negatív számok bináris rendszerben — a komplement	352
A MOS 6510 utasításkészlete	354
Standard utasítások	355
Tiltott utasítások	356
A MOS 6510 utasításai és hatásuk a jelzőkre	357
Kódszámítási táblázat	359
Kislexikon	365
Irodalomjegyzék	371

Bevezető

A Commodore cég igazi bestsellert dobott piacra, amikor 1982-ben megjelent a C64-es jeles napra készű számítógépével. Talán maguk sem sejtették, hogy 1990-re mintegy 7 millió darabot fogtak értékesíteni! A gép akkoriban valódi csúcstechnológiának számított, a processzort eredetileg katonai célokra is használták. Teljesítmény — ár viszonyában sokáig felülmúlta a piacon kínált konkurensok hasonló termékeit, ezért egy ideig a fejlett világban is a legjobb házi számítógép (home computer) volt.

Magyarországon igazi „boom”-nak bizonyult. Miután számos vállalat, gazdálkodó vásárolta az akkor még igencsak drága gépet, a forgalom növekedése maga után vonta árának csökkenését is. Kedvezőbb árára lehetővé tette, hogy magánszemélyek is megvásárolhassák. A C64-es már nevet szerzett akkorra, amikor az elmúlt években egy karácsony előtt az országba érkezett egy nagyobb C16-os szállítmány. A Commodore 64 sikerét is bizonyítja, hogy „kisebbségi” testvérét akkor szinte elkapkodták.

Legnagyobb varázsa talán mégsem csak az alacsonyabbá vált ára volt, hanem az, hogy a korábban elvont, elefántcsonttoronyba zárkózott számítástechnikát elsőként tette le az egyszerű emberek asztalára. Emberközpontú használata barátságosabbá tette, és biztosította a gyors sikerélményt is. Magyarországon főleg a C64-es „hátán” nőtt fel a számítástechnikai kultúra.

A technika fejlődése azonban mára átlépte a C64-est is. Most már valóban csak az otthonokban használják, elsősorban szórakozásra. Tegyük hozzá, hogy eredetileg erre tervezték. A hardver fejlődése kiszorította az ipari-kereskedelmi-tudományos alkalmazásokból, de arra is amire való, játékosan oktató asztali számítógépnek ma is kiváló. A C64-esen keresztül ismerkedni a számítástechnika és számítástudomány világával — ma is érdekes, lebilincselő és neveteltető utolsósorban olcsó vállalkozás. Vagy játék?

A Commodore 64-es működéséről, programozásáról és használatáról az elmúlt évek folyamán számos könyv jelent meg, többek közt a Novotrade Rt. gondozásában is. Ezek a könyvek egy-egy felhasználói területet vagy a gép egy-egy elemét ismertették, illetve a BASIC nyelven való programozás megismeréséhez és elsajátításához nyújtottak segítséget. Most a Commodore használatához szükséges tudnivalókat igyekeztünk összefogni, hogy ezáltal a Commodore-tulajdonosok mind jobban kihasználhassák a gép nyújtotta lehetőségeket.

Bemutatjuk a C64-es legfontosabb áramkörü elemeit: a processzort, a kép- és egyéb jelek előállítására szolgáló chipet, a tárchipeket és a tárkezelést megvalósító segédáramkört. A

egyes integrált áramkörök működését és vezérlését assembly, illetve BASIC nyelven írt példákkal szemléltetjük, bár az assembly és a BASIC utasításokat csak a következő két fejezetben ismertetjük részletesen. A hardverelemek után szólunk röviden a gép csatlakozóiról, a kivezetésekről és szerepükről, hogy ezzel teljessé tegyük a C64-es vezetékeiről és hardveréről alkotható képet.

A BASIC-kel foglalkozó részben térünk ki a háttértárolók szerepére és az adattárolás megvalósítására, összehasonlítjuk a kazettás és a lemezes egység működését mind műszaki szempontból, mind pedig a felhasználót érintő kérdések vonatkozásában. Itt ismertetjük az operációs rendszer utasításait is, melyek az egyes elemek közvetlen elérését teszik lehetővé.

Könyvünk negyedik fejezetében pedig a BASIC és assembly nyelvű programok ötvöztetésével foglalkozunk. Leírunk néhány olyan ötletet, bemutatunk néhány rutint, melyeket a felhasználó saját programjaiba is be tud építeni, s melyekkel a kényesebb programozási kérdések áthidalhatók.

A függelékben röviden áttekintjük a számítástechnika fejlődéstörténetét, összefoglaljuk a számítógépek működésének logikai alapjait. Itt ismertetjük a logikai algebra legfontosabb műveleteit, valamint a C64-esben használatos számrendszereket. A függelékben szerepeltetjük azokat a táblázatokat is, amelyek a processzor teljes utasításkészletét mutatják be. A kötetben szereplő idegen szavak és rövidítések értelmezéséhez a könyv végén található kislexikon nyújt segítséget.

Könyvünkben a Commodore-irodalom jelölésrendszeréhez alkalmazkodtunk. A különböző számrendszerekben felírt számokat is ennek megfelelően jeleztük. A szám előtt a % bináris számrendszert, a # decimális, a \$ pedig hexadecimális felírást jelöl. Például a decimális 28 kettes, illetve tizenhatos számrendszerben:

#28 = %00011100

#28 = \$1C (\$001C)

A 256-nál nagyobb számok négyjegyű hexadecimális megfelelőt kapnak, függetlenül attól, hogy 4 vagy csak 3 értékes jegyük van-e.

A hardver ismertetésénél az egyes rövidítések fölötti egyenes vonal azt jelenti, hogy a jelhez tartozó áramköri vezetékek alacsony állapotban aktív. Ilyen például az $\overline{\text{IRQ}}$.

1. A C64-es hardvere

Technikai adatok

Processzor: MOS 6510 (újabbban 8500)
Órajel: 1 MHz
ROM: 20 kbyte
RAM: 64 kbyte, dinamikus
Cím/adatbusz: 16/8 bit

A processzor perifériái

Video: MOS 6567, 6569 (újabbban 8565)
Audio: MOS 6581 (újabbban 8580)
Input-output: MOS 6526 (6526A)
Belső vezérlés: 825100

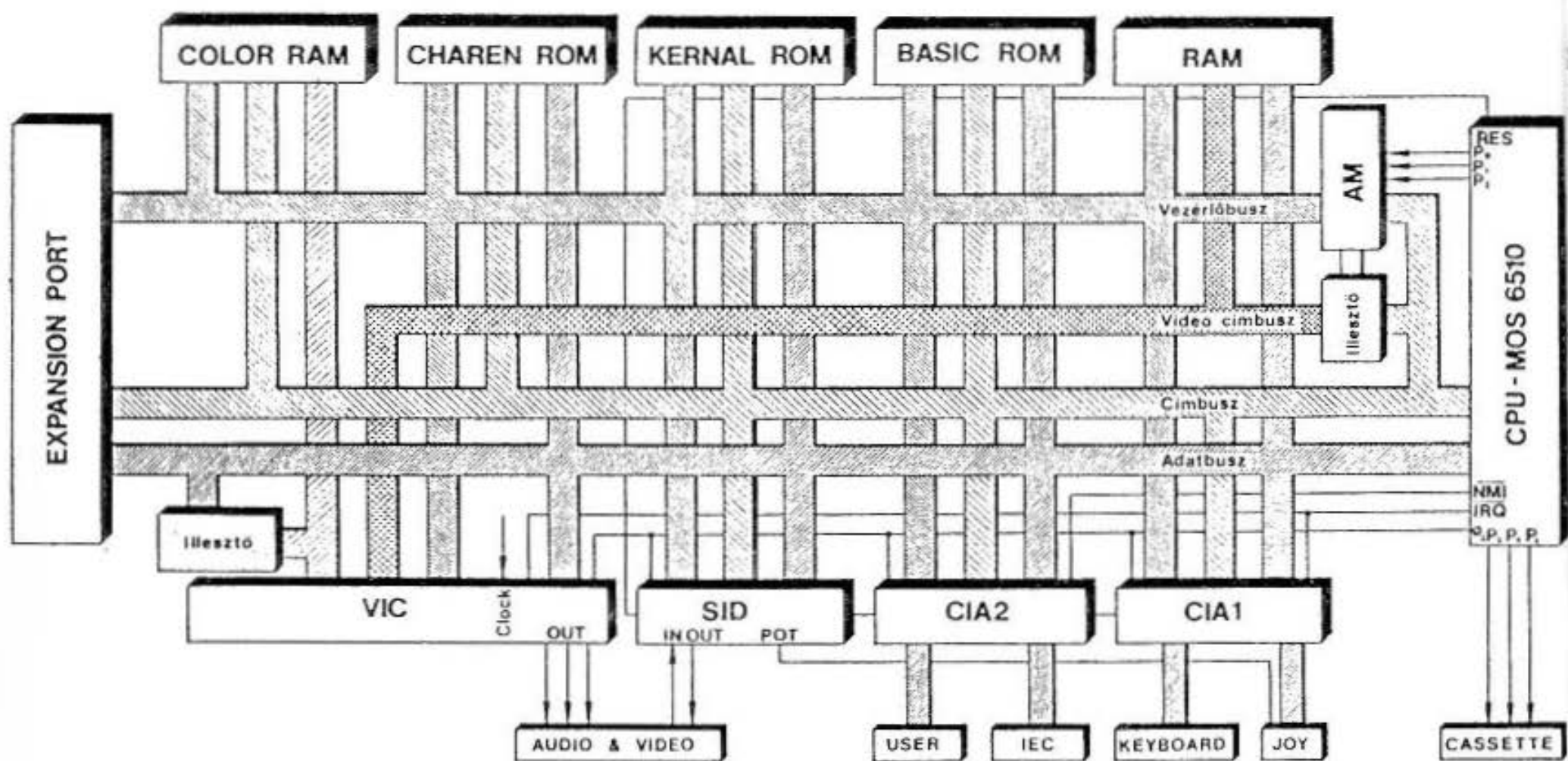
A ROM

BASIC: MOS 2364, 64 kbit (8 kbyte)
KERNAL: MOS 2364, 64 kbit (8 kbyte)
CHAREN: MOS 2332, 32 kbit (4 kbyte)

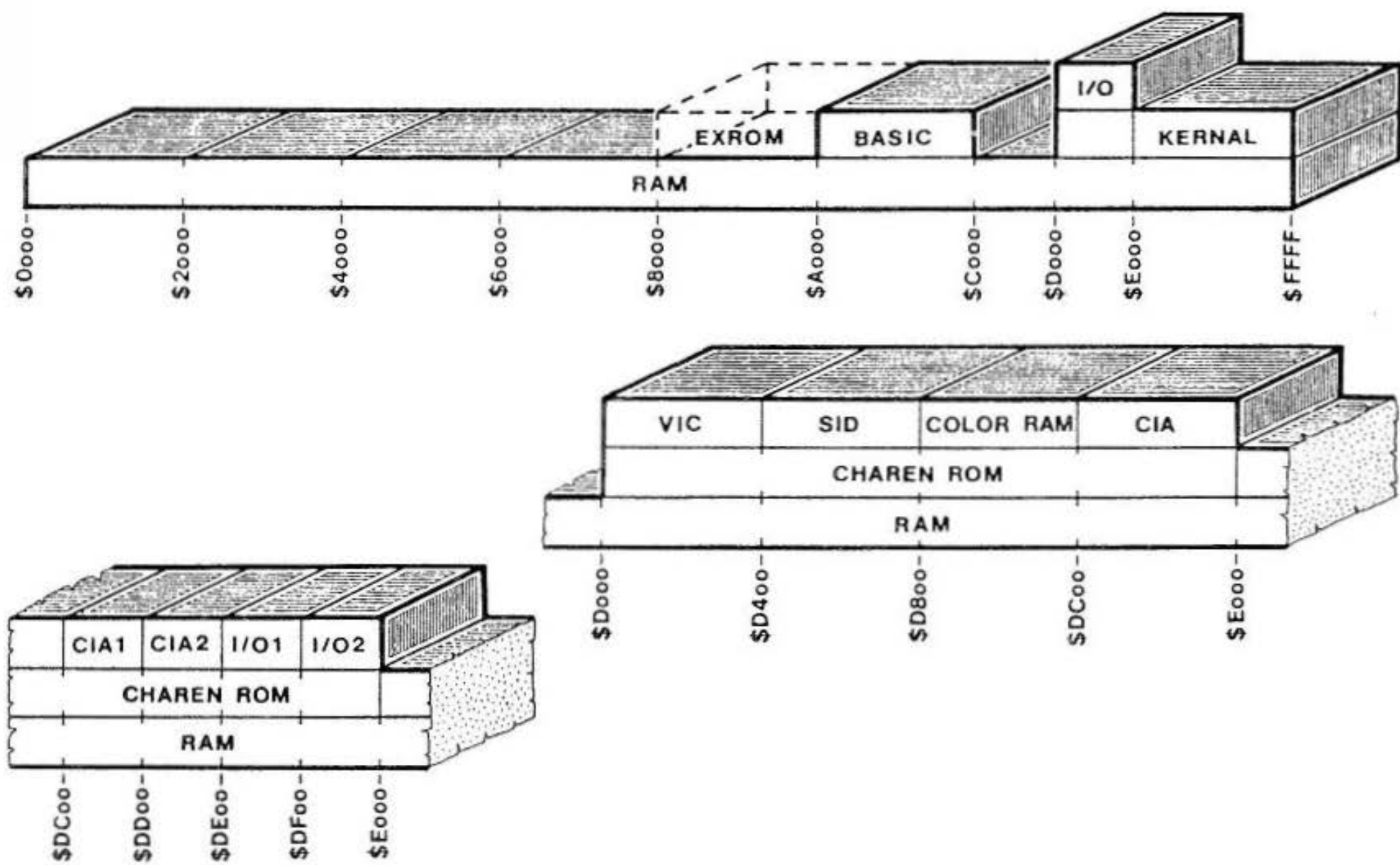
A RAM

4164, 8 db 64 kbites (újabbban 2 db 2464) áramkör
2114-30L, 1 db 4 kbites áramkör

A C64-es nyitott hardverű, 8 bites számítógép. A nyitottság többek között azt is jelenti, hogy a gép Z80 processzor mellett is tud működni, és a bővítőrésbe helyezett Z80 processzorkártya segítségével átalakítható CP/M rendszerű számítógéppé is. Ez ugyan kiterjeszti kompatibilitásának határait, de a gyakorlatban nem nagyon terjedt el. A 6510 processzorral a Z80 szoftveresen is emulálható, ám ez meglehetősen lassú megoldás.



1. ábra. A rendszer elvi felépítése



2. ábra. A memóriaszegmensek átlapolása

A technikai adatok felsorolásából kitűnik, hogy a számítógép tára 64 kbyte-nál jóval nagyobb. A processzor 16 címvezetékkel azonban csak 65536 byte-ot képes megcímezni. Ennyit csak maga a RAM lefoglal. Ezt a látszólagos ellentmondást a *multiplex* nevű technikai trükk oldja fel. A multiplexelés teszi lehetővé, hogy fizikailag azonos címen található, egymástól eltérő tárolók más-más időben ugyan, de elérhetőek legyenek a processzor és a felhasználó számára. Ilyenkor azt mondjuk, hogy a memória egyes szegmensei egymásra lapoltak.

A multiplex-technikát — a többszörözést — igen gyakran alkalmazzák a digitális elektronikában. Esetünkben ez körülbelül azt jelenti, hogy a processzor időben más-más alkalommal más és más jelet küld, illetve fogad ugyanazon a vezetéken. Megfelelően nagy ütemfrekvencia esetén a működés szakaszossága nem érzékelhető. A multiplex rendszerekben egy ütemező-vezérlő áramkör biztosítja, hogy ne keletkezzen zavar működés közben. A C64-esben ez a vezérlő az AM (Address Manager); egy olyan FPLA (Field Programmable Logic Array) áramkör, amelyet a felhasználó (ez esetben a MOS) programoz.

A C64-es belső elektronikája 33 integrált áramkört, valamint számos tranzisztort, diódát és más passzív alkatrészt foglal magába. Természetesen az integrált áramkörök a meghatározók:

1.	6510	1 db	Processzor
2.	6567	1 db	VIC (Video Interface Controller)
3.	6526	2 db	CIA (Complex Interface Adapter)
4.	6581	1 db	SID (Sound Interface Device)
5.	2364A	2 db	ROM, 8 kbyte (8K x 8 bit)
6.	2332A	1 db	ROM, 4 kbyte (4K x 8 bit)
7.	4164	8 db	RAM, 8 kbyte (64K x 1 bit)
8.	2114-30L	1 db	RAM, 1K x 4 bit
9.	825100	1 db	AM, programozható logikai áramkör
10.	74LS257	2 db	2 az 1-hez multiplexer x 4
11.	74LS258	1 db	2 az 1-hez multiplexer x 4
12.	74LS139	1 db	2 bites demultiplexer 2-ről 4-re
13.	74LS373	1 db	D típusú közbenső tár, 8 bites
14.	74LS08	1 db	ÉS kapu x 4
15.	74LS74	1 db	D típusú tár x 2
16.	74LS193	1 db	Számláló, 4 bites előre-hátra
17.	74LS629	1 db	VCO (Voltage Controller Oscillator)
18.	MC4044	1 db	PLL áramkör (Phase Locked Loop)
19.	MC4066	2 db	Analóg kapcsoló x 4
20.	7406	1 db	Inverter x 6
21.	7805	1 db	Feszültségszabályozó, 5 V-os
22.	7812	1 db	Feszültségszabályozó, 12 V-os

Az áramköri panelen egy belső és kilenc külső csatlakozó található: a billentyűzet (belső), a két joystick, a tápfeszültség csatlakozója, a bővíítőport, az RF kimenet, az audio-video kimenet, a soros busz, a kazettás egység csatlakozója, valamint a user port.

A billentyűzet 66 gombos. Az alfanumerikus klaviatúra mellett 4 funkcióbillentyű található.

1.1. A processzor

A C64-esben a Commodore cég leányvállalata, a kaliforniai MOS félvezetőgyártó által előállított 6510 jelű processzort alkalmazzák. A MOS 6510 8 bites, 1 MHz-es processzor, teljesítményét tekintve ma már a középmezőny alsó felében helyezkedik el (megjelenésekor csúcstechnológiát tartalmazott).

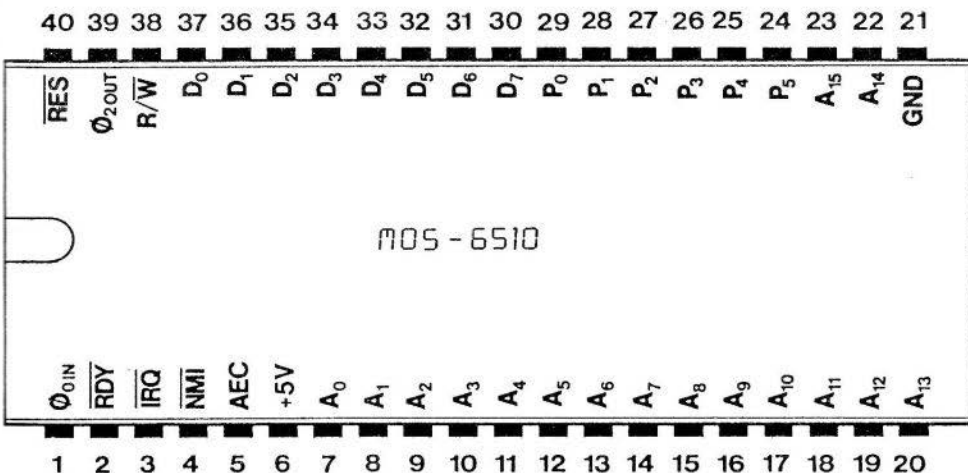
A MOS a 6510-es mellett teljes periferiális integrált áramköri skálát is gyárt. Ezeket együttesen a 65XX processzorcsalád néven szokták említeni. Néhány tagja:

- 6502: CPU, a 6510 elődje, a lemezegység processzora,
- 6504: CPU, az MPS 802 nyomtató processzora,
- 6510: CPU, a C64-es processzora,
- 6522: VIA, Versatile Interface Adapter, perifériameghajtó,
- 6526: CIA, Complex Interface Adapter, a VIA továbbfejlesztése,
- 6569: VIC, Video Interface Controller, videojel-generátor,
- 6581: SID, Sound Interface Device, szintetizátor és egyéb ROM chipek.

A MOS 6510 a 8 bites processzorok családjába tartozik, de társaira (Z80, 8080, 6800 stb.) csak nagy vonalakban hasonlít. Adatbusza 8 bit, címbusza 16 bit egyidejű továbbítására alkalmas. Hat I/O vezetéke és hat belső regisztere van. A C64-esben megközelítőleg 1 MHz ütemfrekvenciával üzemel.

A processzor egy 40 lábú DIL-tokban foglal helyet; tűinek megnevezése és jelentése a 3. ábrának megfelelő sorszámozás szerint:

1. OIN: órajel bemenet. PAL: 985,2 kHz
2. RDY: READY. Ha alacsony, a processzor adatot vár az adatbuszon.
3. IRQ: Interrupt Request, megszakításkérelem. Ha alacsony, megszakítás következik.
4. NMI: Non Maskable Interrupt, megszakításkérelem. Ha alacsony, megszakítás következik.
5. AEC: Address Enable Control, címzésengedélyezés. Ha alacsony, a processzor a busz-vonalat nagy ellenállású állapotba hozza. A periféria IC-k ekkor végezhetnek buszműveleteket.
6. V_{cc}: tápfeszültség, +5 V.



3. ábra. A processzor tokozása

- 7—20. A₀-A₁₃: címbusz, háromállapotú.
 21. GND: Ground, rendszerföld.
 22—23. A₁₄-A₁₅: címbusz.
 24—29. P₅-P₀: a processzor I/O vezetékei (portok).
 30—37. D₇-D₀: adatbusz, háromállapotú.
 38. R/W: Read/Write. 0 = íráshozzáférés, 1 = olvasás.
 39. Φ_{2OUT}: órajel kimenet. Vezérlő szinkronjel a többi egység számára.
 40. RES: RESET. Ha alacsony, a processzor alapállapotba hozza magát.

1.1.1. A 6510 I/O portjai és belső regiszterei

A processzor I/O vezetékeinek feladata a kazettás egység és a processzor közötti adatforgalom biztosítása és a belső tárkonfiguráció beállítása. A hat I/O vezeték a 24—29. lábakon jelenik meg.

24. CASS MOTOR: a kazettás egység motorvezérlője, 1 = kikapcsolva.
 25. CASS SENSE: a kazettás egység állapotszenzora, 1 = PLAY lenyomva.
 26. CASS WRT: adatkimenet.
 27. CHAREN: a \$D000—DFFF címtartomány állapota, 1 = I/O chipek.
 28. HIRAM: a \$E000—FFFF címtartomány állapota, 1 = ROM.
 29. LORAM: a \$A000—BFFF címtartomány állapota, 1 = ROM.

A 6510 belső regiszterei:

1. AC: akkumulátor, 8 bites
2. XR: X indexregiszter, 8 bites
3. YR: Y indexregiszter, 8 bites
4. SP: veremmutató, 8 bites (Stack Pointer)
5. PC: programszámláló, 16 bites (Program Counter)
6. PS: állapotregiszter, 8 bites (Processor Status)

Az állapotregiszter minden bitjének külön funkciója van.

7. N: Negative flag, negativitásjelző, HI-aktív
6. V: Overflow flag, túlsordulásjelző, HI-aktív
5. - NC, nem használt, mindig magas
4. B: Break flag, megszakítás programból, LO-aktív
3. D: Decimal flag, decimális módot jelző bit, HI-aktív
2. I: Interrupt flag, megszakításjelző, LO-aktív
1. Z: Zero flag, nullajelző, HI-aktív
0. C: Carry flag, átvitelbit.

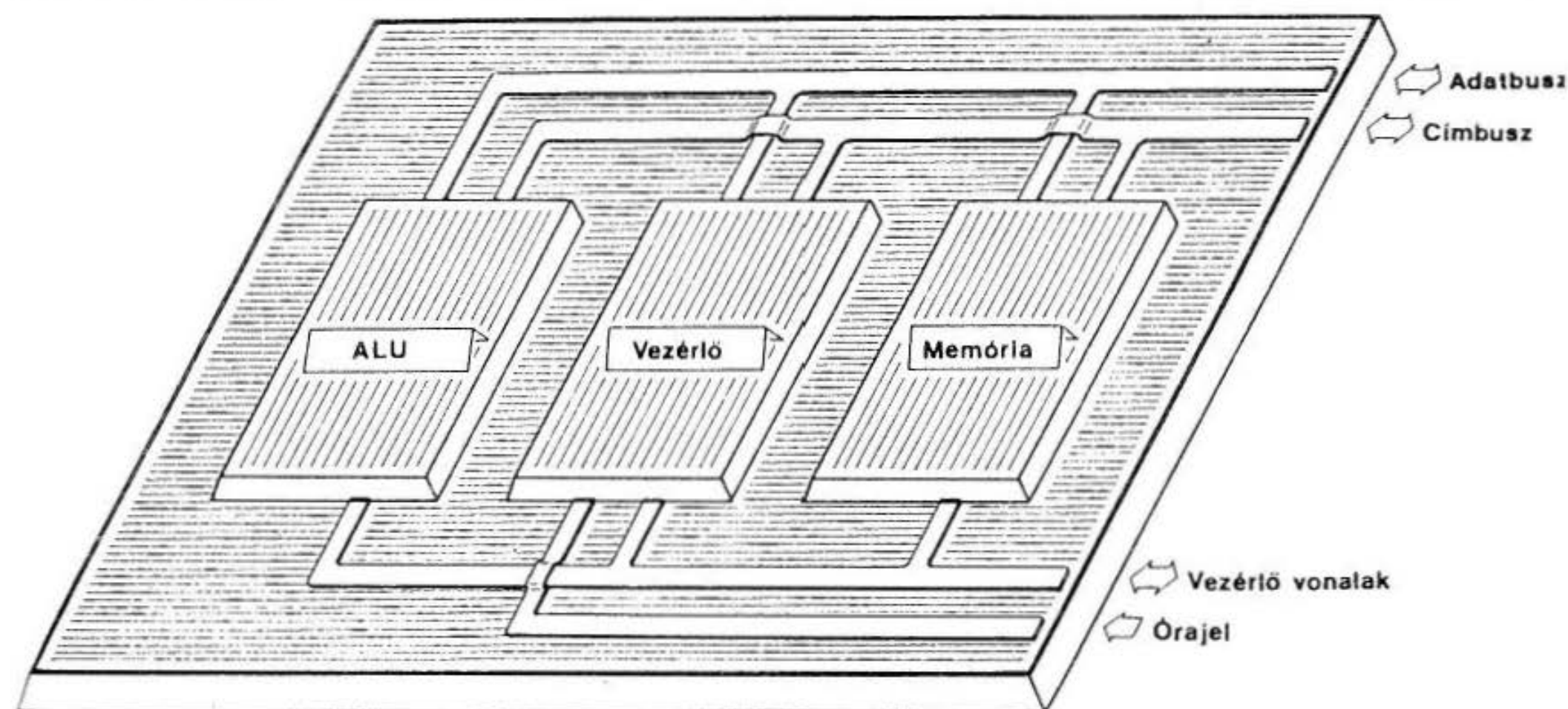
1.1.2. A processzor felépítése és működése

A 6510 működése szempontjából három nagyobb szerkezeti egységre tagolható. Részei: az aritmetikai és logikai egység (ALU), a vezérlőegység (utasításdekódoló és buszvezérlő) valamint a belső tár (regiszterek). A három egység a cím- és adatbuszon, illetve a belső vezérlővezetékeken tart egymással kapcsolatot.

Az egységek között a munkamegosztás a következő. A vezérlőegység beolvassa az adatbuszról az utasításkódot, dekódolja, majd átadja az ALU-nak az aritmetikai paramétereket, végül az eredménynek megfelelően beállítja a regisztereket, a cím- és az adatbuszt. A regiszterekben az utasításban kijelölt művelet(ek) eredménye(i) jelennek meg.

A processzor az utasításokat nem egy ütemciklus alatt hajtja végre, hanem legkevesebb két ütemciklus alatt, de vannak olyan utasítások is, amelyek négy vagy öt ütemciklust igényelnek. Például a TAX utasítás végrehajtásához 2 ütemciklus szükséges. Az első ciklus alatt a vezérlő dekódolja az utasítást, és az akkumulátor tartalmát kihelyezi a belső buszra. A következő ciklus alatt az X regiszter átveszi a buszon megjelent byte-ot, az ALU pedig beállítja az állapotregisztert. Ezzel együtt a vezérlő növeli az utasításszámlálót.

A processzor *három megszakítási fokozattal* rendelkezik. Ezek elsőbbségi sorrendben: RESET, NMI és IRQ.



4.ábra. A processzor elvi felépítése

A RESET, vagyis a processzor alaphelyzetbe állítása akkor jön létre, amikor a processzor $\overline{\text{RES}}$ vonala alacsony jelszintet vesz fel. A folyamat a következő módon zajlik le: a RES jel észlelése után a processzor megszakítja az éppen futó programot, az utasításszámlálóba betölti a \$FFFC — FFFD címeken található byte-okat, és munkáját e címtől folytatja. Ezután következik a RAM inicializálása, a CIA, a VIC és a SID megfelelő regisztereinek alapállapotba hozása. Erről a megfelelő ROM rutinok gondoskodnak. A RESET valójában nem megszakítás, de működése hasonló.

Szoftveresen nem tiltható NMI megszakítást a RESTORE billentyű lenyomása, a CIA2, illetve a bővíítőportba helyezett külső egység hozhat létre. Amint az NMI vonal alacsony jelszintet vesz fel, a feldolgozott utasítást még befejezi a processzor, de ezt követően az utasításszámlálót és az állapotregisztert a verembe menti, a megszakításjelzőt magasra állítja, majd az utasításszámlálóba új értéket, a \$FFFA — FFFB címeken található byte-okat tölti. Ezután végrehajtja a megszakítási rutint, amelynek RTI-vel kell véget érnie. Ekkor visszatölti a veremből az állapotregisztert és az utasításszámlálót.

A harmadik megszakítási mód, az IRQ, az állapotregiszter megszakításjelzőjének magasra állítása után letiltható. IRQ-t létrehozhat a programozó a BRK utasítással, a CIA1, a VIC, illetve a bővíítőportba helyezett külső egység. Működése hasonló az NMI-hez. A megszakításrutin vektora viszont más: \$FFFE — FFFF. A verembe mentett állapotregiszter vizsgálatával a szoftveres és hardveres IRQ-k szétválaszthatók.

A processzor a *tárat* laponként kezeli. 256 ilyen lap van, ezek egyenként 256 byte-ból állnak. A lapok között kitüntetett szerepet tölt be a *nulláslap*, amely nevét onnan kapta, hogy bármelyik byte-jának címe \$00-val kezdődik. A másik különleges státusú lap a *verem*, amely a RAM \$0100 — 01FF területét fedi le, és amelyet a processzor saját céljaira foglal le. Ebbe a 256 byte-os visszatérési verembe kerülnek azok az adatok, amelyek a program végrehajtásához azért szükségesek, hogy a processzor az elágazások — alprogramok végrehajtása —

után visszatalálhasson az elágazás utáni első utasításra. Ide kerülnek tehát a processzor működése során keletkező visszatérési címek, illetve a megfelelő utasítások hatására az állapot regiszter és az akkumulátor tartalma. A veremből mindig az az érték vehető ki először, amire utoljára került bele.

1.1.3. A processzor nyelve

A 6510 processzor olyan önálló gépi nyelvet alkalmaz, amelyet az idegen gyártók hasonló 8 bites processzorainak egyike sem használ, és speciális fordítóprogram nélkül nem is képes végrehajtani. (A 6502 nyelv szempontjából természetesen teljesen megegyezik a 6510 processzorral; az egyiken írt programok a másikon hiba nélkül lefutnak.) Egy gépi nyelv használhatóságát címezsmódjainak száma és azok kezelhetősége jellemzi a legjobban.

A processzor 256 utasítást különböztethet meg. E 256 lehetséges utasítás jelentős része kisebb-nagyobb csoportokra osztható, amelyek azonos műveletet végeznek más-más tárcsák között. Az azonos jellegű utasítások között különbséget a címezsmódjában tehetünk. A 6510-es processzor nyelve 13 címezsfajtát különböztet meg. A 13 változat egyszerre egyik műveletnél sem szerepelhet, mert a processzor tervezői a 8 bites utasításkódban csak 3 bitet használhattak fel a címezsmód kiválasztására. A legtöbb címezssel operáló műveleteknél is csak 8 címezsmód ismert; 3 biten 8 különböző állapot hozható létre. A címezsmódok működésének ismertetésére egy teljes fejezetet szenteltünk a gépi nyelvvel és programozással foglalkozó részben, ezért ezeket itt nem részletezzük.

A 6510 utasításkészlete kétfelé bontható: egyrészt az általánosan elfogadott kódokra, másrészt a tiltott kódokra. Az előbbi csoportot a szabványos vagy standard jelzővel különböztethetjük meg a tiltott kódoktól. A legtöbb program, így a BASIC interpreter és az operációs rendszer is a standard utasításkészletet alkalmazza. Ezekben tiltott kód nem szokott előfordulni.

A szabványos utasításkészlet egymástól eltérő mnemonikkal jelölt 56 utasítását az 1. táblázatban foglaltuk össze. Természetesen a címezsmódok miatt az utasítások száma jóval több, hiszen egyik-másik mnemonikhoz 8 címezsfajta is használható. A táblázatban az utasítás mnemonik mögött álló szám jelzi azt, hogy az adott utasításhoz hány címezsfajta, hány különböző kód tartozik. Az 56 utasításhoz összesen 151 kód lehetséges.

A tiltott utasításkészlet 27 különböző műveletet tartalmaz. Ezeknek csak egy része új művelet, a jelentősebb csoportot két vagy több szabványos műveletet integráló utasítások alkotják.

1. táblázat. A szabványos utasításkészlet elemei

1. ADC	8	összeadás	29. JSR	1	szubrutinhívás
2. AND	8	logikai ÉS művelet	30. LDA	8	betöltés
3. ASL	5	szorzás kettővel	31. LDX	5	betöltés
4. BCC	1	relatív ugrás	32. LDY	5	betöltés
5. BCS	1	relatív ugrás	33. LSR	5	osztás kettővel
6. BEQ	1	relatív ugrás	34. NOP	1	nincs művelet
7. BIT	2	bitteszt	35. ORA	8	logikai VAGY művelet
8. BMI	1	relatív ugrás	36. PHA	1	veremművelet
9. BNE	1	relatív ugrás	37. PHP	1	veremművelet
10. BPL	1	relatív ugrás	38. PLA	1	veremművelet
11. BRK	1	megszakítás programból	39. PLP	1	veremművelet
12. BVC	1	relatív ugrás	40. ROL	5	forgatás balra
13. BVS	1	relatív ugrás	41. ROR	5	forgatás jobbra
14. CLC	1	státusbit művelet	42. RTI	1	viSSzatérés megszakításból
15. CLD	1	státusbit művelet	43. RTS	1	viSSzatérés szubrutinból
16. CLI	1	státusbit művelet	44. SBC	8	kivonás
17. CLV	1	státusbit művelet	45. SEC	1	státusbit művelet
18. CMP	8	összehasonlítás	46. SED	1	státusbit művelet
19. CPX	3	összehasonlítás	47. SEI	1	státusbit művelet
20. CPY	3	összehasonlítás	48. STA	7	tárolás
21. DEC	4	dekrementálás	49. STX	3	tárolás
22. DEX	1	dekrementálás	50. STY	3	tárolás
23. DEY	1	dekrementálás	51. TAX	1	transzfer művelet
24. EOR	8	logikai kizáró VAGY	52. TAY	1	transzfer művelet
25. INC	4	inkrementálás	53. TSX	1	transzfer művelet
26. INX	1	inkrementálás	54. TXA	1	transzfer művelet
27. INY	1	inkrementálás	55. TXS	1	transzfer művelet
28. JMP	2	abszolút ugrás	56. TYA	1	transzfer művelet

2. táblázat. A tiltott utasítások

1. AAX	1	A AND X	15. INB	7	INC-SBC
2. ABS	*	STOP	16. LDT	6	LDA-TAX
3. ANL	1	AND-LSR	17. LSE	7	LSR-EOR
4. ANN	1	AND művelet	18. NO1	*	nincs művelet
5. ANX	1	AND művelet	19. NO2	*	nincs művelet
6. ASO	7	ASL-ORA	20. NO3	*	nincs művelet
7. AXI	1	AND művelet	21. RAD	7	ROR-ADC
8. AXR	1	A AND X	22. RAN	7	ROL-AND
9. AXS	1	több művelet	23. TAN	1	AND művelet
10. AXX	1	A AND X	24. TSA	1	AND művelet
11. AXY	1	AND művelet	25. XAS	1	AND-SBC
12. AYY	1	AND művelet	26. XYA	1	AND művelet
13. DAR	1	AND-ROR	27. YXA	1	AND művelet
14. DEM	7	DEC-CMP			

* Az ABS művelet 12 kódot, az NO1 művelet 6 kódot, az NO2 művelet 14 kódot, és az NO3 művelet 7 kódot foglal. A mnemonik után álló szám a címzések számát jelenti.

A 27 művelet 103 kódot jelent. $103 + 151 = 254$, tehát két utasítás hiányzik. Ezeket a felsorolásból azért hagytuk ki, mert a \$2B kód azonos az ANN utasítás \$0B kódjával, valamint a \$EB kód azonos az SBC utasítás \$E9 kódjával. A teljes utasításkészletet a függelékben közöljük; a gyors keresést elősegítendő kétféle csoportosításban is. A függelékben megadjuk a különböző utasítások jelzőkre gyakorolt hatását is.

1.2. A videojel-generátor (VIC)

A processzornak nincs kapacitása a rendszeren belül előálló összes vezérlőfeladat elvégzésére, így az órajelek előállítására és a RAM ciklikus felfrissítésére sem. Ezeket a feladatokat a MOS 6567 (6569) típusú Video Interface Controller látja el. Ez a chip állítja elő azonban a televíziós képet is.

A VIC képszerkesztését a következők jellemzik:

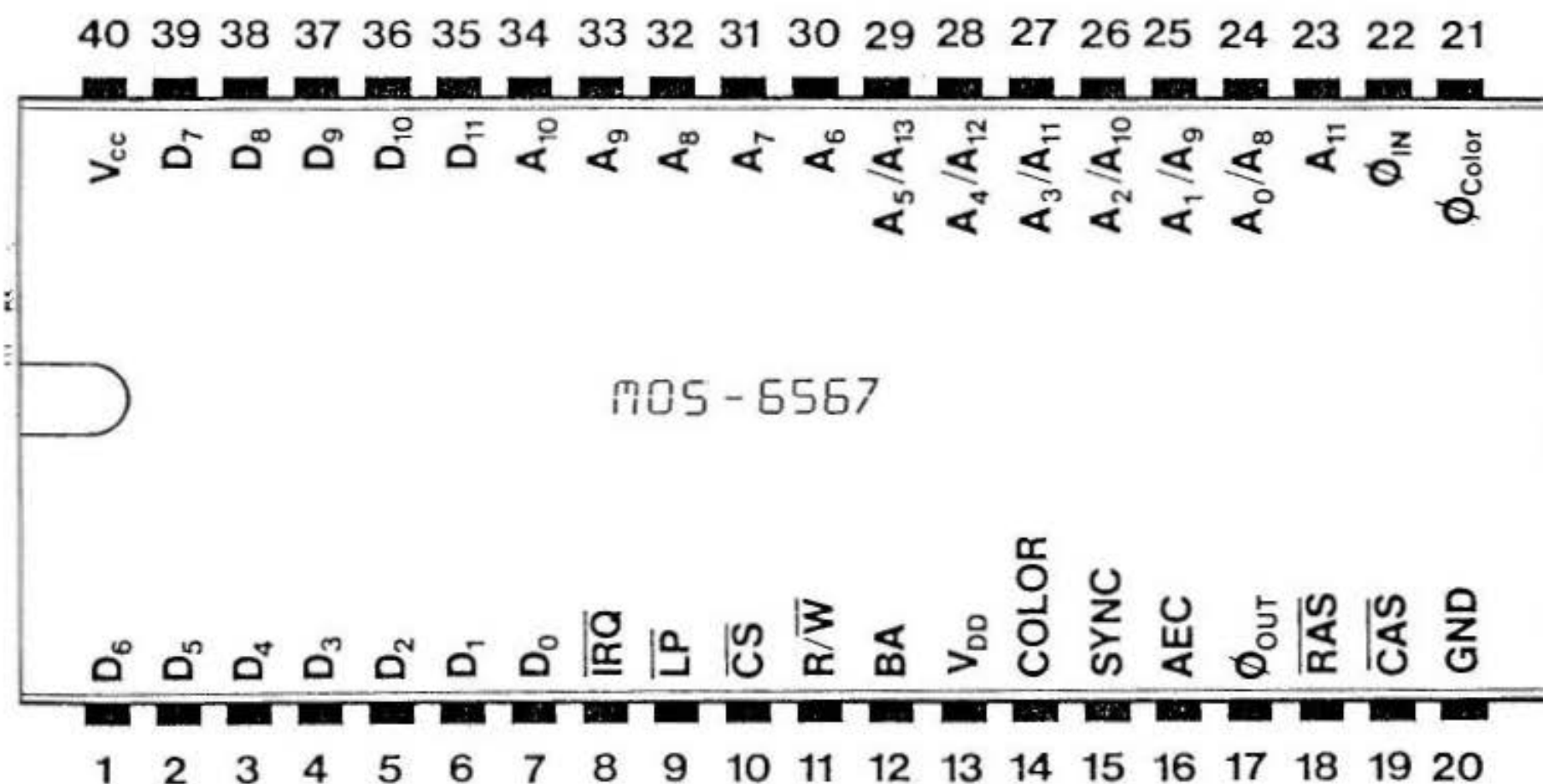
- karakteres grafika, 40 oszlop — 25 sor;
- High Resolution (HIRES) grafika, 320 oszlop — 200 sor;
- 8 db önálló 24 x 21 pixeles sprite;
- 16 különböző szín.

A VIC hardvere emellett biztosítja a következőket:

- eltolható videoram;
- eltolható karaktergenerátor;
- fényceruza használata;
- megszakítástechnika.

A processzor számára a VIC külső egység, és illesztése is ennek megfelelő. A VIC speciális helyzeténél fogva, valamint amiatt, hogy a processzor feladatainak egy részét átvállalja, nagyobb hatáskörrel rendelkezik. Kezében tartja a rendszerbusz ütemezését, sőt a vezérlést is veheti a processzortól. Erre azért van szükség, mert a memória bizonyos (esetlegesen változó) területeihez a képszerkesztés miatt hozzá kell férnie. Míg a processzor a saját feladatai időben halaszthatja, addig a VIC-nek szolgáltatnia kell a monitor számára a képernyő felépítéséhez szükséges jelet, amely a szigorúan meghatározott frekvenciamenet miatt nem késleltethető. Emiatt a VIC a processzorhoz képest előnyt élvez a memória olvasása tekintetében. Mivel a VIC képszerkesztése egyik regiszteren keresztül felfüggeszthető, ezért a különösen számításigényes programokat, amelyek a tárat is erősen igénybe veszik, érdemes kikapcsolt képernyő mellett futtatni. Persze ezzel jelentős gyorsítás nem érhető el, mert a VIC azokat a pillanatokban igyekszik felhasználni a munkája számára, amikor a processzor amúgy sem férhet hozzá a rendszerbuszhoz. Az időzítés összehangolása egyszerűbb, mintsem gondolnánk: ha a rendszerütem négyszögjele éppen magas fázisban van, a processzor, egyébként pedig a VIC használja a rendszerbuszt. Amennyiben a fél rendszerütem nem lenne elég a VIC-nek ahhoz, hogy a megfelelő adatot kiolvassa a memóriából, az AEC vezetéket, amelyet a hozzáférés kezdetekor alacsonyra állított, továbbra is alacsony állapotban tartja, ezáltal a processzor továbbra sem kísérel meg a buszhoz férni.

A VIC 40 lábú DIL-tokban kerül forgalomba. Túlinek jelentését és szerepét az 5. ábra jelöléseinek megfelelően ismertetjük.



5. ábra. A VIC tokozása

- 1—7. D₆-D₀: a processzor adatbusza.
8. $\overline{\text{IRQ}}$: Interrupt Request, kimenet, megszakításkérelem. Alacsony, ha az IMR és az IRR regiszterek (\$D01A és \$D019) megfelelő bitjei azonosan magasak.
9. $\overline{\text{LP}}$: Light Pen Strobe, bemenet, fényceruza.
10. $\overline{\text{CS}}$: Chip Select, bemenet, vezérlőjel. A VIC a processzorbuszon csak a $\overline{\text{CS}}$ láb alacsonyra állítása után végezhet műveletet. A vezérlőjelet a tárkezelő egység, az AM szolgáltatja.
11. R/ $\overline{\text{W}}$: Read/Write, buszvezérlő jel. Írás, ha alacsony; olvasás, ha magas.
12. BA: Bus Available, a busz foglaltságát jelző vonal. Alacsony, ha a buszon még nem jelent meg a várt adat.
13. V_{DD}: +12 V tápfeszültség.
14. COLOR: kimenet, színinformáció.
15. SYNC: kimenet, sor- és képszinkron jel.
16. AEC: Address Enable Control, kimenet, címzésengedélyezés. Ha alacsony, a VIC használja a rendszerbuszt.
17. Φ_{OUT} : kimenet, rendszerütem.
18. $\overline{\text{RAS}}$: Row Address Select, kimenet, sorválasztó a RAM címzéséhez (a dinamikus RAM vezérlőjele).
19. $\overline{\text{CAS}}$: Column Address Select, kimenet, oszlopválasztó a RAM címzéséhez (a dinamikus RAM vezérlőjele). A dinamikus RAM címzési folyamatát a 18. vonallal együtt szabályozza.
20. GND: Ground, rendszerföld.
21. Φ_{COLOR} : bemenet, színfrekvencia.
22. Φ_{IN} : bemenet, Dot Clock.

- 23. A₁₁: a processzor címbusza.
- 24—29. A₀-A₅/A₈-A₁₃: multiplexelt RAM címbusz.
- 30. A₆: a videoram címbusza.
- 31. A₇: a videoram címbusza.
- 32—34. A₈-A₁₀: a processzor címbusza.
- 35—38. D₁₁-D₈: a COLOR-RAM adatbusza.
- 39. D₇: a processzor adatbusza.
- 40. V_{cc}: tápfeszültség, 5 V.

Az *ütemjel* előállításához a C64-esen belül három fontos frekvencia van. A legnagyobb frekvencia a kvarcstabilizált oszcillátor (rezgőkör) rezgése. $\Phi_{\text{COLOR}} = 17,7344 \text{ MHz}$. A VIC ebből a frekvenciából állítja elő a televíziós képhez szükséges alapfrekvenciákat (világosságjel, színjel, sorfrekvencia, képfrekvencia stb.). A processzor számára a VIC állítja elő az ütemfrekvenciát, ehhez azonban néhány áramkör a kvarcoszcillátor frekvenciáját előosztó fokozattal 8/18-ad részére csökkenti. Az így kapott frekvencia a Dot Clock, értéke 7,8819 MHz. A processzor ütemfrekvenciája ennek 8-ad része: $\Phi_0 = 985,25 \text{ kHz}$. A Φ_0 tehát a kvarcoszcillátor frekvenciájának 18-ad része. A frekvencia 18 részre osztásához PLL (Phase Locked Loop — fázisban szabályozott ciklus) áramkört alkalmaznak, amellyel tetszőleges osztási arányt el lehet érni. (2 hatványai alapján csak a digitális osztóáramkörök végzik az osztást.)

Meg kell említenünk, hogy ezek a frekvenciaértékek csak az Európában forgalmazott gépekre érvényesek, mert Japánban és az USA-ban más a videoszabvány (nem PAL, hanem NTSC). Ezért az amerikai piacon eladott gépek megfelelő frekvenciái: $\Phi_{\text{COLOR}} = 14,318 \text{ MHz}$, Dot Clock = 8,182 MHz, $\Phi_0 = 1,0227 \text{ MHz}$, az osztási viszony pedig 14.

A VIC regisztereinek eléréséhez a processzor az AM segítségével a $\overline{\text{CS}}$ vonalat alacsonyra állítja. A VIC csak 16 kbyte címterülettel dolgozhat, mert a 16 bites busz legfelső két bitje nem áll rendelkezésére. A tokon azonban nem 14, csak 7 csatornát találunk, mivel a RAM címezése két részletben, multiplexelt rendszerben ugyanazon csatornán más-más időpontban következik be. Ez magyarázza azt is, hogy a tok 24—29. lábainak kettős elnevezése van (pl. a 24. láb egyszerre a 0-ás és a 8-as bit hordozója). A multiplex-eljárást a RAM speciális szervezése indokolja. A RAM ugyanis a címet 8 bites egységekben, előbb alsó, majd felső címbite-ok formájában igényli. Azt, hogy mikor melyik az érvényes, a $\overline{\text{RAS}}$ és a $\overline{\text{CAS}}$ jel állapota szabja meg.

A VIC specifikációjában már említettük, hogy a videoram eltolható. Ahhoz, hogy ez a teljes RAM-ra igaz legyen, a VIC számára nem elegendő a 14 címbit. A teljes RAM címezéséhez a két hiányzó bitet a CIA2 A portja biztosítja. A 0-ás és 1-es bit szolgáltatja a 14-es és 15-ös címbiteket. Ez a megoldás ugyan némileg bonyolítja a hardvert, de zavart nem okoz. Ezek a bitek inverteren keresztül jutnak a RAM-hoz, ezért alacsony állapotuk eredményez magas címbitet.

A COLOR RAM eléréséhez a VIC a 35, 36, 37 és a 38-as lábakat használja. Ezekhez csatlakozik adatvezetékeivel a színtár. Viszont felmerülhet a kérdés: miért van megkétszerezve a címbusz néhány vezetéke? Hogyan lehetséges az, hogy a VIC multiplexelt címbuszán kívül a 23. és 32—34. lábakon ismét jelentkezik az A₈-A₁₁ vonal? Ez amiatt van, mert a

VIC-nek a képszerkesztéshez a képernyő- és színtárt egyszerre kell elérnie. A két memória azonban egymástól általában messze lévő két tárterületen helyezkedik el, s ezeket egyszerre csak külön buszon címezheti meg.

1.2.1. A VIC regiszterei

A regiszterek szerepének ismertetésekor a sorszám után zárójelben megadjuk a megfelelő tárcímét is.

R ₀	(\$D000)	S ₀ X ₀ -S ₀ X ₇ : a 0. sprite X koordinátája
R ₁	(\$D001)	S ₀ Y ₀ -S ₀ Y ₇ : a 0. sprite Y koordinátája
R ₂	(\$D002)	S ₁ X ₀ -S ₁ X ₇ : az 1. sprite X koordinátája
R ₃	(\$D003)	S ₁ Y ₀ -S ₁ Y ₇ : az 1. sprite Y koordinátája
R ₄	(\$D004)	S ₂ X ₀ -S ₂ X ₇ : a 2. sprite X koordinátája
R ₅	(\$D005)	S ₂ Y ₀ -S ₂ Y ₇ : a 2. sprite Y koordinátája
R ₆	(\$D006)	S ₃ X ₀ -S ₃ X ₇ : a 3. sprite X koordinátája
R ₇	(\$D007)	S ₃ Y ₀ -S ₃ Y ₇ : a 3. sprite Y koordinátája
R ₈	(\$D008)	S ₄ X ₀ -S ₄ X ₇ : a 4. sprite X koordinátája
R ₉	(\$D009)	S ₄ Y ₀ -S ₄ Y ₇ : a 4. sprite Y koordinátája
R ₁₀	(\$D00A)	S ₅ X ₀ -S ₅ X ₇ : az 5. sprite X koordinátája
R ₁₁	(\$D00B)	S ₅ Y ₀ -S ₅ Y ₇ : az 5. sprite Y koordinátája
R ₁₂	(\$D00C)	S ₆ X ₀ -S ₆ X ₇ : a 6. sprite X koordinátája
R ₁₃	(\$D00D)	S ₆ Y ₀ -S ₆ Y ₇ : a 6. sprite Y koordinátája
R ₁₄	(\$D00E)	S ₇ X ₀ -S ₇ X ₇ : a 7. sprite X koordinátája
R ₁₅	(\$D00F)	S ₇ Y ₀ -S ₇ Y ₇ : a 7. sprite Y koordinátája
R ₁₆	(\$D010)	S ₀ X ₈ -S ₇ X ₈ : az X koordináta 9. bitje, rendre 0 — 7. sprite
R ₁₇	(\$D011)	a VIC első ellenőrző-vezérlő regisztere

- | |
|---|
| 0 — 2. bit: YSCL ₀ -YSCL ₂ , a képernyő Y irányú eltolási tényezője
3. bit: RSEL, ha 0: 24 sor, ha 1: 25 sor a képernyőn
4. bit: BLNK, ha 0, a képernyő kikapcsolva
5. bit: BSM, ha 1, bittérképes üzemmód (HIRES)
6. bit: ECM, ha 1, bővített színes üzemmód
7. bit: RC ₈ , a RASTER regiszter (\$D012) 9. bitje |
|---|

R ₁₈	(\$D012)	RC ₀ -RC ₇ : Raszter R/W. Olvasás: az éppen aktuális rasztensor. Írás: az a rasztensor, amelyben a VIC megszakítást vált ki
-----------------	----------	---

- R₁₉ (\$D013) LPX₀-LPX₇: LIGHT PEN X, a fényceruza X koordinátája. Olvasás: az éppen aktuális rasztensor. Írás: az a rasztensor, ahol a fényceruza megszakítást vált ki
- R₂₀ (\$D014) LPY₀-LPY₇: LIGHT PEN Y, mint fent, de az Y koordinátára
- R₂₁ (\$D015) SE₀-SE₇: SPRITE ENABLE, sprite-engedélyezés. Ha a megfelelő bit magas, a hozzá tartozó sprite láthatóvá válik
- R₂₂ (\$D016) a VIC második ellenőrző-vezérlő regisztere

- 0—2. bit: XSCL₀-XSCL₇ a képernyő X irányú eltolási tényezője
3. bit: CSEL, ha 0: 38 oszlop, ha 1: 40 oszlop a képernyőn
4. bit: MCM, ha 1, többszínű üzemmód
5. bit: RST, értéke nulla
6. bit: NC, nem használt, mindig magas
7. bit: NC, nem használt, mindig magas

- R₂₃ (\$D017) SEXY₀-SEXY₇: SPRITE EXPAND Y. Ha a megfelelő bit magas, a sprite minden pontja kettőződik Y irányban
- R₂₄ (\$D018) a karaktergenerátor és a képernyőtár címbitjei

0. bit: NC, nem használt
- 1—3. bit: CB₁₀-CB₁₃, a karaktergenerátor báziscímének 10—13. bitjei
- 4—7. bit: VS₁₁-VS₁₃, a képernyőtár báziscímének 11—13. bitjei

- R₂₅ (\$D019) IRR: Interrupt Request Register, megszakításkérelem. Csak olvasható!

0. bit: RIRQ, rasztermegszakítás-kérelem jelzése
1. bit: ISBC, sprite-háttér ütközés megszakításkérélmé
2. bit: ISSC, sprite-sprite ütközés megszakításkérélmé
3. bit: LPIRQ, fényceruza által kiváltott megszakítás jelzése
- 4—6. bit: NC, nem használt
7. bit: IRQ, megszakításjelző; magas, ha bármelyik bit magas

- R₂₆ (\$D01A) IMR: Interrupt Mask Register, megszakításmaszkolás. Csak írható! A regiszter bitjei maszkként működnek. Az IRR párhuzamos bitje a feltétel teljesülése esetén is csak akkor magas, ha az IMR bitje is magas. A regiszter nem olvasható, illetve az olvasás eredménye nem fedti a regiszter tartalmát.

R27	(\$D01B)	BSP ₀ -BSP ₇ : háttér-sprite elsődlegességének eldöntése. Ha a sprite-hoz tartozó bit magas, a háttér az elsődleges.
R28	(\$D01C)	SMC ₀ -SMC ₇ : a sprite-ok multicolor beállítása; ha a bit magas, a hozzá tartozó sprite többszínű.
R29	(\$D01D)	SEXX ₀ -SEXX ₇ : SPRITE EXPAND X; a sprite-ok X irányú kiterjesztése. Ha a megfelelő bit magas, a sprite kétszeresére nyúlik.
R30	(\$D01E)	SSC ₀ -SSC ₇ : sprite-sprite ütközés detektor. Az érintett sprite-ok bitjei magasak, ha ütközés történt.
R31	(\$D01F)	SBC ₀ -SBC ₇ : sprite-háttér ütközés detektor. Az érintett sprite-nak megfelelő bit magas, ha érintés történt.
R32	(\$D020)	0 — 3. bit: keretszín
R33	(\$D021)	0 — 3. bit: háttérszín 0
R34	(\$D022)	0 — 3. bit: háttérszín 1
R35	(\$D023)	0 — 3. bit: háttérszín 2
R36	(\$D024)	0 — 3. bit: háttérszín 3
R37	(\$D025)	0 — 3. bit: SMC ₀ , sprite multicolor mód, szín #0
R38	(\$D026)	0 — 3. bit: SMC ₁ , sprite multicolor mód, szín #1
<i>A 4 — 7. bitek az R32 — R38 regisztereknél nem használatosak</i>		
R39	(\$D027)	0 — 3. bit: a 0. sprite alapszíne
R40	(\$D028)	0 — 3. bit: az 1. sprite alapszíne
R41	(\$D029)	0 — 3. bit: a 2. sprite alapszíne
R42	(\$D02A)	0 — 3. bit: a 3. sprite alapszíne
R43	(\$D02B)	0 — 3. bit: a 4. sprite alapszíne
R44	(\$D02C)	0 — 3. bit: az 5. sprite alapszíne
R45	(\$D02D)	0 — 3. bit: a 6. sprite alapszíne
R46	(\$D02E)	0 — 3. bit: a 7. sprite alapszíne
<i>A 4 — 7. bitek értéke a fenti nyolc regiszter mindegyikénél mindig magas.</i>		

1.2.2. A VIC ábrázolási lehetőségei és a sprite-kezelés

A VIC két alapvető ábrázolási módja a *karakteres* és *nagyfelbontású* grafika, ezek mindegyikéhez társulhat az ún. *sprite-grafika* is. Mindkettőre érvényes az *egyszínű* (HIRES) és a *többszínű* (MULTICOLOR) grafikai megjelenítés.

Így a következő ábrázolási kombinációk lehetségesek:

- karakteres normál grafika (alaphelyzet),
- karakteres többszínű grafika (bővített színes üzemmód),
- karakteres grafika színes üzemmódban (multicolor),
- karakteres grafika sprite-okkal,

- nagyfelbontású grafika egyszínű üzemmódban,
- nagyfelbontású grafika többszínű üzemmódban,
- nagyfelbontású grafika sprite-okkal.

Minthogy a sprite-ok is lehetnek egy-, illetve többszínű üzemmódban, ezért ennél több esetben is lehetséges, különösen azt figyelembe véve, hogy a sprite-grafika bármelyik másik üzemmódhoz kapcsolható. Azonban ezek még mindig csak az egyszerűbben programozható megoldások. Programozási trükkök segítségével — a megszakítási technika lehetőségeinek kihasználásával — olyan megoldások is elérhetők, mint pl. ha 8 sprite-nál több szerepel egyszerre, a képernyő megosztott, egyszerre látható néhány sor karakteres grafika nagyfelbontású grafikával, a sprite-ok a szerkesztési mezőn (a háttéren) kívül (alatta vagy felette, esetleg is-is) jelennek meg, a képernyő keretének és/vagy háttérének színe egy időben különböző színű sávokra van felosztva.

A VIC 16 szín előállítására képes. A 16 szín csak bizonyos korlátozások mellett látható egyszerre a képernyőn. (Karakteres üzemmódban nincs ilyen korlátozás.) Nagyfelbontású üzemmódban egy 8 x 8 pont méretű rácson belül maximum három különböző szín jelenhet meg egyszerre. Három különböző színt ölthetnek a sprite-ok is. A színek és a hozzájuk tartozó kódok:

Fekete 0	Fehér 1	Vörös 2	Türkiz 3
Lila 4	Zöld 5	Kék 6	Sárga 7
Narancs 8	Barna 9	Rózsaszín 10	Szürke 1 11
Szürke 2 12	Világoszöld 13	Világoskék 14	Szürke 3 15

Az egyszínű, normál üzemmódú karakteres grafika különösebb magyarázatot nem igényel. A kezelése egyszerű, a gép bekapcsolása után ez az ábrázolásmód használatos.

Jellemzői:

- 25 sorban 40 oszlop (1000 karakter)
- a képernyő háttérszíne a 16 szín bármelyike lehet
- a karakterek színe 16-féle lehet
- egy-egy karakter valójában 8 x 8-as pontmátrix, azonban csak egy egységként kezelhető.

Ebben az üzemmódban a háttér színét a 33. regiszter (\$D021) adja meg. A karakterek színét a színtár határozza meg. A színtár a \$D800 — DBFF területen helyezkedik el, és hozzárendelése a képernyőtárhoz egyértelmű: \$0400 → \$D800, \$0401 → \$D801 stb.

Bővített színes üzemmódú karakteres grafika esetén a karakterek mögötti háttérszín másképp is lehet. A képernyőtárban 256 különböző karakterhez rendelhetünk mutatót; ez nyolc biten ábrázolható. Ebben az esetben a háttérszín csak egyféle lehet. A szín kódját a VIC

33. regisztere (\$D021) tartalmazza. A többszínű üzemmód megvalósításához a VIC-nek pótlólagos információra van szüksége annak eldöntéséhez, hogy a háttér színét a 33 — 36. regiszterek közül melyiknek a tartalmához igazítsa. Mivel a hardver erre külön lehetőséget nem biztosít, ezért a gép tervezői a következő megoldást választották.

A bővített színes üzemmódban ábrázolható, egymástól különböző karakterek száma legyen 64. Ebben az esetben a karakterek kiválasztásához 6 bit is elegendő. A fennmaradó 2 bit, amely a képernyőtárban található byte 2 felső bitje, a szín kiválasztására szolgál. A két bit négy egymástól eltérő kombinációt alkothat:

0 0	a szín kódja a 33. regiszterben (\$D021)
0 1	a szín kódja a 34. regiszterben (\$D022)
1 0	a szín kódja a 35. regiszterben (\$D023)
1 1	a szín kódja a 36. regiszterben (\$D024)

Tehát azon az áron, hogy csak az alfanumerikus karaktereket ábrázolhatjuk, azt a lehetőséget nyerjük, hogy minden különösebb programozási procedúra nélkül megváltoztathatjuk bizonyos karakterek mögött a háttér színét. Ez elsőrendű kiemelési lehetőséget kínál, amelyet nem biztosít az ún. reverse-ábrázolás sem.

Az üzemmódot a VIC 17. regiszterének (\$D011) 6. bitje állítja be, illetve kapcsolja ki.

A *színes (multicolor) üzemmódú* karakteres grafika négyszínű karaktereket eredményezhet. Bekapcsolását a 22-es regiszter (\$D016) 4. bitjének magas állapota jelenti. A színek kiválasztást alapvetően befolyásolja, hogy az ún. színtárban tárolt félbyte legmagasabb (azaz 3.) bitje magas-e. Mert ha nem, akkor a multicolor mód nem érvényesül. Ekkor a pontok színét (ahol bitmintában 1 áll) a színtár maradék három bitje határozza meg, a többi pont színe pedig azonos a háttér színével. Ha viszont a színtár 3. bitje magas, az ábrázolás bitpárok alapján történik. A mátrix ekkor 4 x 8 bites, a pontok dupla szélesek.

A színeket az alábbi hozzárendelés alapján választja ki a VIC:

0 0	33-as regiszter (\$D021)
0 1	34-es regiszter (\$D022)
1 0	35-ös regiszter (\$D023)
1 1	a színtár alsó három bitje

Ez a megoldás négyszínű karaktereket eredményez, azonban az egyik szín karakterről karakterre változhat, nyolc különböző szín bármelyike lehet.

A *nagyfelbontású grafikus* (high resolution bit mode, HIRRES) *ábrázolási mód* gyökeresen eltér a karakteres ábrázolásoktól. A tár és a képernyő között lényegében közvetlen hozzárendelés áll fenn. A különbség főként a képernyőtár méretében mutatkozik. Míg a karakteres ábrázolásnál a tár tulajdonképpen egy mutatót tárol, amely a bitminta pozícióját mutatja a karaktergenerátoron belül, addig nagyfelbontás esetében a bitek állapota látható a képernyőn.

A képernyőtár mérete karakteres grafikánál 1000 byte, nagyfelbontású grafikánál 8000 byte. A képernyő ekkor nem 25 x 40 egységből, hanem 200 sorból és 320 oszlopból áll. A tárolás és a képernyő egymáshoz rendelését a 6. ábra szemlélteti.

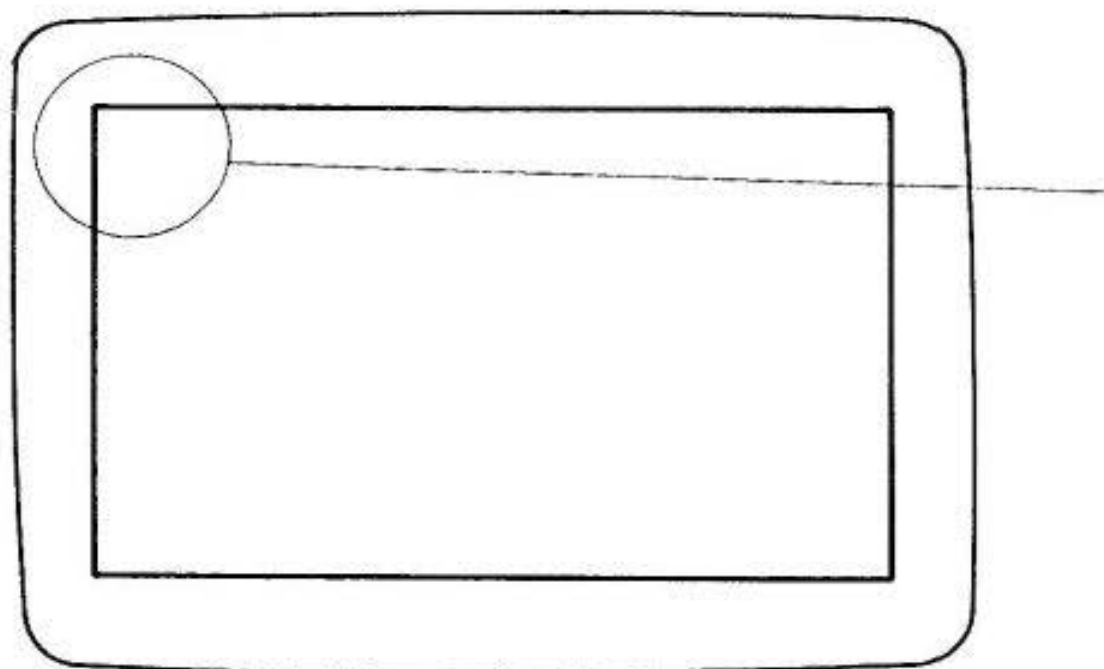
A színtár ekkor nem vesz részt a pontok színének alakításában. Ezt a szerepet a képernyőtár karakteres része veszi át. Minden byte alsó félbyte-ja a háttér színét, felső félbyte-ja pedig a bekapcsolt pontok színét szabályozza. Így ugyan lényegében a látható pontokat egyszerű egyszínűre választani, azonban minden egyes karakterhelyen más és más lehet a pontok és a háttér színe. Alkalmazhatjuk tehát mind a 16 színt, ám egy karakterhelyen egyszerűen csak egy látható ezekből. Ez utóbbi megkötés miatt a többszínűség gyakorlatilag hasznavehetetlen lehetőség, ezért is nevezhetjük az ábrázolási módot egyszínű nagyfelbontásnak.

Bekapcsolását a 17. regiszter (\$D011) 5. bitje végzi. Amennyiben magas, a grafika nagyfelbontású, amennyiben alacsony, karakteres.

Az előzőekben említett hátrányt, hogy egy (8 x 8 bit) karakterhelyen belül nem tudunk több színt megjeleníteni, kiküszöböli a *többszínű nagyfelbontású üzemmód* (multicolor bit mode). A multicolor üzemmód alkalmazásának azonban ára van. Igaz ugyan, hogy egy karakterhelyen belül a háttérpontok színét is figyelembe véve négy különböző színt jeleníthet meg, de a felbontás 320 x 200 pixelről 160 x 200 pixelre csökken. Egy pontnak ebben az esetben egy bitpár felel meg. A bitpár tartalma határozza meg a pont színét a következő módon:

- 0 0 a 3. regiszter tartalma
- 0 1 a képernyőtár felső félbyte-ja
- 1 0 a képernyőtár alsó félbyte-ja
- 1 1 a színtár tartalma

Ezt az üzemmódot a 17. regiszter (\$D011) 5. bitjének, és a 22. regiszter (\$D016) 4. bitjének magasra állítása kapcsolja be.



2	0	0	0
		1	
		2	
		3	
		4	
		5	
		6	
		7	
2	0	C	8
		9	
		A	
		B	

2	0	0	8
			9
			A
			B
			C
			D
			E
			F
2	0	D	0

6. ábra. A tár és a képernyő hozzárendelése nagyfelbontású grafika esetén

A sprite egy 24 x 21 pixelből álló grafikai elem, amely a képernyőre mintegy „rálapolódik”. Minthogy a sprite jóval kisebb, mint a képernyő felülete, ezért a VIC biztosítja a figura koordinátákhoz kötött irányítását, lényegében a mozgatását. A sprite pontjainak és a tár bitjeinek egymáshoz rendelését a 7. ábra szemlélteti.

1. Egyidejűleg nyolc sprite-tal dolgozhatunk, amelyek 0-tól 7-ig számozottak.
2. A sprite-ok egyenként, külön-külön is lehetnek multicolor vagy HIREN állapotúak.
3. A sprite-ok kiterjedését kétszeresére külön-külön is megváltoztathatjuk vízszintes, illetve függőleges irányban.
4. A sprite-ok érintkezését ellenőrizhetjük.
5. A sprite-ok és a háttér érintkezését is detektálhatjuk.
6. A sprite-ok elsődlegesek a háttéralakzatokkal szemben, ez azonban megváltoztatható.
7. A sprite-ok egymás közötti elsődlegessége rögzített.
8. A sprite-ok helyzete egymástól független.

0 1 2 3 4 5 6 7 8 15 16 23

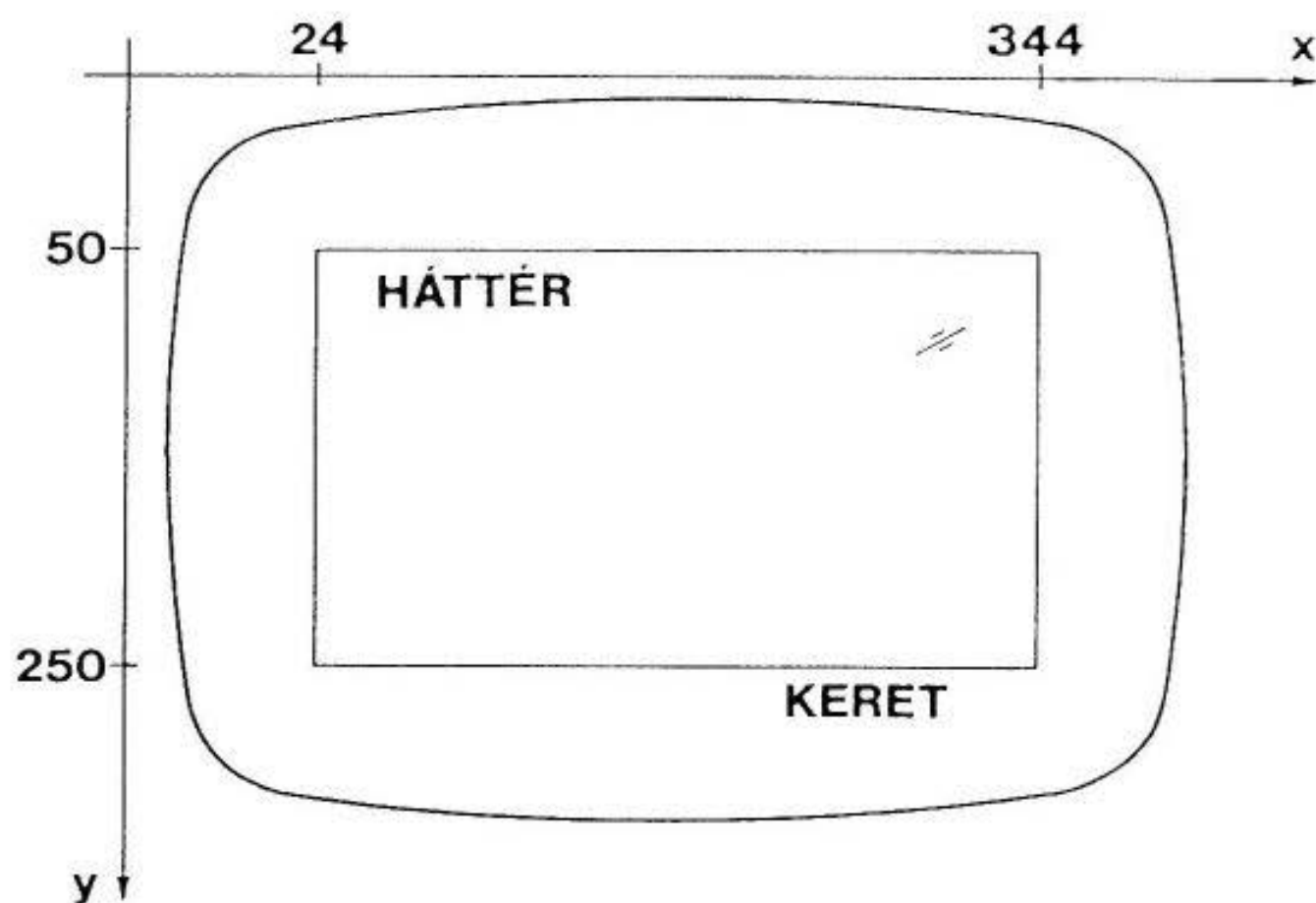
	7	6	5	4	3	2	1	0				
0	\$	0	3	4	0							
1							3					
2							6					
3							9					
4							C					
5							E					
⋮												
15	\$	0	3	6	D							
16				7	0							
17							3					
18							6					
19							9					
20							C					

7. ábra. A sprite és a tár leképezése

A VIC a sprite-mutatókat a mindenkor képernyőtár legfelső nyolc byte-ján keresi. Alaphelyzetben ez a terület a 2040-2047 (\$07F8-07FF) byte-okra esik. A VIC címzését bármely 16 kbyte-os lapra állítjuk is, a sprite-okhoz tartozó információkat, adatokat a VIC *mindig RAM-ban keresi*. Így nem kell attól tartani, hogy a felső lapokon elhelyezkedő ROM-ok zavarhatják a sprite-ábrázolás programozási tennivalóit. Az említett területmutatók értéke attól függ, hogy az ábrázolni kívánt sprite melyik 64 byte-os blokkban helyezkedik el. (Például a 832-es relatív címen a 13. sprite adatai kezdődnek.) Ennek megfelelően kell a mutatókat beállítani.

A tiszta kép kedvéért vegyünk egy példát! Ha a bekapcsolás utáni alaphelyzetben a 7. sor számú blokkban található bitmintát kívánjuk megjeleníteni, akkor a következő műveleteket kell elvégezni: (1) beállítjuk a sprite-blokk mutatóját, (2) a sprite-ot a látható mezőbe pozicionáljuk, (3) bekapcsoljuk a sprite-ot. Lássuk mindezt részletesen!

1. Kiválasztjuk céljainkra a 0. sprite-ot, illetve annak regisztereit. A blokkmutatót a \$07F8 címre (2040) kell betölteni. Ez most 7, tehát: POKE 2040, 7
2. Mivel a képernyő kereten belüli része és külső része a VIC elé ugyanolyan megjelenítésfeladatokat állít, ezért a képernyőn a pozicionálást a rastersorok abszolút értéke határozza meg (ld. a 8. ábrát is).



8. ábra. A képernyő: keret és háttér

A rasterkoordináták a Descartes-féle koordináta-rendszer IV. síknegyedének megfelelően értelmezhető abszolút értékű egész számok. Amint a 8. ábrán is látszik, a háttér bal felső sarka a (24,50) koordinátákhoz kötött. A jobb alsó sarok koordinátáit a képernyő pixelméretei határozzák meg, azaz: (344, 250).

Tudjuk, hogy egy normál méretű sprite 24 x 21 pontból áll. Ezenfelül még azt is felfedezhetjük, hogy a sprite-ok koordináta-regisztereinek bekapcsolás után zéró értékekkel vannak fel-

Értve. Így a sprite-ot hiába is kapcsoljuk be, az a képernyő kerete alatt nem lesz látható. Emiatt a 0. sprite koordinátáit állítsuk be a (100, 100) koordinátákra:

```
POKE 53248, 100    x koordináta,
POKE 53249, 100    y koordináta.
```

A Most pedig engedélyezzük a sprite láthatóságát. Mivel a 0. sprite-ot kívánjuk látni, ezért a megfelelő regiszterben (53269) a 0. bitet kell magasra állítani:

```
POKE 53269,1
```

A fenti műveletek végrehajtása után már látható az eredmény: egy fehér színű, szabályosnak nem nevezhető alakzat, amelyben néhány „remegő” pont is van — ezek száma, ha le-
tünk néhány billentyűt, egy pillanatra megsokasodik. A képernyőre hozott alakzat nem más,
mint a processzor visszatérési verme, a verem legfelső része. A remegés az interpreter és az
operációs rendszer normális működésének látható jele.

A sprite-ábrázolás mikéntjének megértéséhez ez a leírás meglehetősen kevés támpontot
nyújthat. A profi sprite-kezelés jószerével csak gyakorlás útján sajátítható el. Viszonylag
könnyebben tanulhatunk a GRAPHICS BASIC interpreter-szoftver segítségével. A prog-
ramnyelv majdnem emberi nyelven segíti a sprite-okkal kapcsolatos problémák programozá-
sát.

A sprite-ok kezelésének megértéséhez jónéhány példát kellene tanulmányozni. Az alábbi
program arra a célra készült, hogy a fontosabb funkciókat demonstrálhassuk, s így megismer-
essük a sprite-ok kezelését.

Programunk egyetlen sprite-ot definiál, jelenít meg és mozgat.

```
100 PRINT CHR$(147)      ; Képernyőtörlés
110 VIC=53248             ; a VIC kezdőcíme
120 POKE VIC+32,6         ; a Képernyő Kerete kék
130 POKE VIC+33,0         ; a Képernyő háttere fekete
140 GOSUB 1000            ; a sprite adatainak betöltése
150 POKE 2040,11         ; a 0. sprite adatai a 11. blokkban
160 POKE VIC+39,2         ; a 0. sprite színe vörös
170 POKE VIC+29,1         ; a 0. sprite X irányú kiterjesztése
180 POKE VIC+21,1         ; a 0. sprite engedélyezése
190 A=1                  ; a függőleges mozgás egysége
200 Y=50                 ; a függőleges mozgás felső határa
210 FOR I=-1 TO 1 STEP 1/60 ; a vízszintes mozgás ciklusa
220 POKE VIC,SIN(I)*90+155 ; a vízszintes koordináta beállítása
230 POKE VIC+1,Y         ; a függőleges koordináta beállítása
240 Y=Y+A               ; a függőleges mozgás biztosítása
250 IF Y>230 OR Y<50 THEN A=-A ; az alsó-felső határ ellenőrzése
260 NEXT I
270 GOTO 210             ; végtelen ciklus, kilépés : RUN/STOP
                           ; szubrutin a sprite adatok betöltéséhez

1000 FOR I=0 TO 63
1010 READ X : S=S+X
1020 POKE 11*64+I,X      ; a 11. sprite-blokkba
1030 NEXT I
1040 IF S<>8253 THEN STOP
1050 RETURN
1060 DATA 0, 0, 0, 0    ; a sprite bitmintája
```



```

1070 DATA 0, 0, 0,255, 0, 3
1080 DATA 129,192, 14,126,112, 25
1090 DATA 255,152, 55,255,236,111
1100 DATA 255,246,111,255,246,223
1110 DATA 255,251,223,255,251,223
1120 DATA 255,251,111,255,246,111
1130 DATA 255,246, 55,255,236, 25
1140 DATA 255,152, 14,126,112, 3
1150 DATA 129,192, 0,255, 0, 0
1160 DATA 0, 0, 0, 0, 0, 0

```

Ahhoz, hogy egy sprite láthatóvá váljék, meg kell adni néhány paraméterét. Először rögzíteni kell, hogy melyik sprite-ot kívánjuk kezelni (0 — 7). A példaprogram a 0. sprite-ot állítja elő. Első dolgunk legyen az, hogy meghatározzuk a sprite-hoz tartozó blokkot. (Feltételezzük, hogy a megfelelő blokkban már ott van a sprite bitmintája.) Alaphelyzetben a #2040-es tárcímre kell betöltenünk a mutatót. A sprite adatainak a #704 — 767 területet választottuk, ez a 11. blokk. Tehát POKE 2040,11. Állítsuk be a sprite színét. A sprite nagyfelbontású, ezért csak egy színregisztert kell beállítanunk. A szín legyen vörös, tehát POKE 53287,2. Ezt követően el kell döntenünk, hogy *x*, illetve *y* irányban kiterjesztjük-e a sprite-ot. Ha csak az *x* irányban állítjuk be: POKE 53277,1. Általános esetben most következne a sprite koordinátáinak beállítása, de a példaprogram más módszert követ. A koordináták megadása után láthatóvá kell tenni a sprite-ot. Ehhez az 53269-es tárcím megfelelő (itt 0.) bitjének beállítása szükséges.

A példaprogram nemcsak megjeleníti, hanem mozgatja is a sprite-ot. Ezért nem állította be a koordinátákat az egyébként megfelelő helyen. A programban felírt ciklus úgy állítja be a sprite *x* és *y* koordinátáját, hogy az mozgása közben egy csavarvonalban mozgó gömböt szimulál, amely vissza-visszaverődik a mennyezeten és a padlón. (A gömb kissé túlzás, az inkább ellipszis, mint kör.) A program nem áll meg magától, mert a 270. sorban található GOTO végteleníti. A programot a RUN/STOP billentyűvel állíthatjuk meg.

A sprite egyéb jellemzőit az alábbiakban röviden ismertetjük.

Elsőbbség: a legmagasabb elsőbbségi kódú sprite a 0. Ez azt jelenti, hogy azonos koordináták esetén az összes többi takarja. Az elsőbbség a sprite-ok sorszámának növekedésével csökken.

Nagyítás, kiterjesztés: a sprite-ot *x* és *y* irányban terjeszthetjük ki egyszerre vagy külön-külön is. A kiterjesztés *x* irányban a \$D01D, *y* irányban a \$D017 regiszter megfelelő bitjének magasra állításával történik.

Érintkezés, ütközés: kétféle ütközés detektálható. Az egyik a sprite és a háttér, a másik két sprite közötti érintkezés következménye. Sprite és háttér érintkezését a \$D01F regiszterben, sprite és sprite érintkezését a \$D01E regiszterben vizsgálhatjuk.

Sprite multicolor mód: hatására a felbontás *x* irányban a felére csökken, viszont lehetővé válik egy sprite-on belül 3 szín használata.

A sprite bitmintája és a hozzá rendelt szín:

0 0	a pont nem látható (azonos színű a háttérrel)
0 1	a 37. regiszterben (\$D025) levő színkód szerint
1 0	a sprite színregisztere (\$D027-D02A) adja a megfelelő kódot
1 1	a 38. regiszterben (\$D026) levő színkód szerint

A sprite-oknak egy változó és két közös színük lehet.

1.2.3. A VIC címzési megoldásai

A VIC tokozásánál már említettük, hogy a VIC egyszerre mindössze 16 kbyte-ot tud önállóan megcímezni. A RAM négyszer ekkora, ezért az egyik CIA (a 2-es) két portbitet biztosít a címzés megvalósításához: a \$DD00 (56576) címen található 0. és 1. biteket. Segítségükkel választhatjuk ki a négy 16 kbyte-os lap valamelyikét. A bitek ún. LO-aktívak, azaz alacsony állapotuk eredményez magas címbitet.

A VIC a 16 kbyte-os címtérületen belül a 24. regiszterben (\$D018 vagy #53272) található érték alapján választja ki a képernyőtár, illetve a karaktergenerátor helyét.

3. táblázat. a VIC címzésének beállítása (24. regiszter)

Képernyőtár				Kar. gen.					Képernyőtár		Karaktergenerátor
7	6	5	4	3	2	1	0		offset		offset
0	0	0	0	0	0	0	x		0		0
0	0	0	1	0	0	1	x		1024		2048
0	0	1	0	0	1	0	x		2048		4096
0	0	1	1	0	1	1	x		3072		6144
0	1	0	0	1	0	0	x		4096		8192
0	1	0	1	1	0	1	x		5120		10240
0	1	1	0	1	1	0	x		6144		12288
0	1	1	1	1	1	1	x		7168		14336
1	0	0	0				x		8192		
1	0	0	1				x		9216		
1	0	1	0				x		10240		
1	0	1	1				x		11264		
1	1	0	0				x		12288		
1	1	0	1				x		13312		
1	1	1	0				x		14336		
1	1	1	1				x		15360		

Amint a 3. táblázatból kitűnik, a képernyőtárat 1 kbyte-os, a karaktergenerátort 2 kbyte-os lépésekkel lehet mozgatni. Az offsetek a lap báziscíméhez hozzáadandók. A 16 kbyte-os lapok BASIC-ben a következő utasítások útján érhetők el:

0. \$0000 - \$3FFF	0 - 16383 : POKE 56576,133 : POKE 648,4
1. \$4000 - \$7FFF	16384 - 32767 : POKE 56576,133 : POKE 648,68
2. \$8000 - \$BFFF	32768 - 49151 : POKE 56576,137 : POKE 648,132
3. \$C000 - \$FFFF	49152 - 65535 : POKE 56576,136 : POKE 648,136

A 648-as címen a BASIC interpreternek azt adjuk meg, hogy a VIC a képet honnan állítja elő.

A CHAREN elhelyezkedésének megértéséhez némi magyarázattal tartozunk. A CHAREN ROM a tárban a \$D000 — DFFF területen helyezkedik el. Amennyiben utánanézzünk, hogy alaphelyzetben a VIC 24. regiszterében (\$D018) található bitminta milyen címet eredményez, meglepetés vár ránk: a cím nem 53248, hanem 4096! Az ellentmondást az Adres Manager, a tárkezelő egység programozása oldja fel. A fejezet elején már utaltunk arra, hogy a VIC másképpen címzi a RAM-ot és a COLOR RAM-ot, és a karaktergenerátor elérése egyedi. A VIC saját vezetékeivel csak 16 kbyte-ot képes megcímezni a 64 kbyte-os címterületből, a teljes címterület eléréséhez a hiányzó két legfelső bitet a CIA2 A portjának (\$DD00) alsó két (0. és 1.) bitje, a VA₁₄ és a VA₁₅ biztosítja. A tárkezelő egység ennek a két bitnek az értékétől függően címzi meg a karaktergenerátort. Amennyiben a címbitek alacsonyak, a VIC a \$D000 kezdőcímű ROM-ból veheti a karakterek bitmintáit. A bitek megváltoztatása esetében a tárkezelő az átcímzést mellőzi, vagy a megfelelő 16 kbyte-os területre irányítja a VIC hozzáférési kísérleteit.

A karaktergenerátorral kapcsolatos trükkök még nem merültek ki, mert helyét a processzorport (\$0001) is vezérli. A port 2. bitjének alacsony vagy magas állapota szabja meg, hogy a karaktergenerátor a RAM-ban vagy a ROM-ban helyezkedik-e el.

Vizsgáljunk meg egy gyakorlati példát! A programozó, aki helytakarékosagra törekszik és a karakterkészletében ékezetes betűket is akar szerepeltetni, átdolgozza a karaktergenerátort. Lemásolja a RAM-ba, módosítja, majd átkapcsol rá. Ezt úgy is megteheti, hogy a tárkészletbe-kasul feltúrja, és valahova a BASIC programtár kellős közepébe helyezi el az új karaktergenerátort. Ez a megoldás számtalan probléma forrása. Sokkal elegánsabb és takarékosabb megoldás is van.

Tudjuk, hogy a \$D000 — DFFF terület háromszorosan foglalt, valamint azt is, hogy a 64 kbyte RAM folyamatos, tehát az említett területen van 4 kbyte el nem érhető és emiatt gyakorlatilag használhatatlan RAM. Tegyük ide az új karaktergenerátort. Ezt azért tehetjük meg, mert a VIC akkor is a RAM-ban keresi a kép felépítéséhez szükséges adatokat, ha a kezelt terület felett bekapcsolt ROM van. Tehát címezzük meg a 4. 16 kbyte-os lapot. Ez követően egy rövid gépi kódú program gyorsan átmásolja az eredeti CHAREN ROM tartalmát az alatta elhelyezkedő RAM-ba:

```

LDA #$D0
LDY #$00      A kezdőcím
STA $FF
STY $FE      elhelyezése
SEI          A maszkolható megszakítások tiltása
LDA $01
AND #$FB      A tárfelosztás módosítása
STA $01
X1 LDA ($FE),Y
STA ($FE),Y    Másolóciklus
INY
BNE X1        Folytatni a ciklust

```



```

LDA $FF
CMP #$E0
BNE X1      Még nincs vége, folytatni
LDA $01
ORA #$04    Az eredeti tárfelosztás visszaállítása
STA $01
CLI        A megszakítások engedélyezése
RTS

```

Ekkor egy másik program átjavítja a megfelelő karaktereket, így már csak átkapcsolni kell. Beállítjuk a VIC 24. (\$D018) regiszterét (ha kell, egyébként nem muszáj), és beállítjuk a CIA2 A portjának (\$DD00) alsó két bitjét. Ezzel a módosítás lényegében kész. A karakter-generátor tartalmazza a megfelelő betűket, a BASIC-tár nem csökkent. Sőt: ha HIRES grafikát is használunk, az a \$E000 — FFFF területen fog elhelyezkedni, tehát a BASIC-tár továbbra is sértetlen. A megoldás egyetlen hátránya, hogy a \$C000 után található szabad RAM-ból a képernyőtár kihasít 1 kbyte-ot.

A CHAREN 2 x 256 karakter képét tárolja. A \$D000 — D7FF területen a nagybetű-grafikus üzemmód karaktereinek képe található, a \$D800 — DFFF területen pedig a kisbetű-nagybetű üzemmód karaktereinek bitmintái.

A bitminták 8-as csoportokban értelmezhetők. Indexelésük a képernyőkód rendszerét követi, így a 0. a @, az 1. az A stb. Tehát amikor kiadjuk a POKE 1024,1 parancsot, akkor a megjelenő A betű bitmintáját a VIC a \$D008 — D00F területről vette.

Az első 128 x 8 byte a 128 különböző karakter képét, a következő 128 x 8 byte pedig ezek inverzét tárolja.

A HIRES-tár címzéséhez nem találunk regisztereket. Erre azért nincs szükség, mert a HIRES-tár mindig az aktuális 16 kbyte-os lap felső 8 kbyte-jára esik, illetve abból foglal le 8000 byte-ot. Mivel 8 kbyte 8192 byte-ból áll, ezért a HIRES-tár mögött a 253, 254 és 255 mutatójú sprite-terület üres.

A VIC még egy tárterülethez fér hozzá rendszeresen. Ez a COLOR RAM vagy színtár. Ez a terület 1024 byte-ot foglal el, de minden byte-ból csak az alsó négy bitet (0 — 3). A terület címzését a VIC hardveresen kezeli, ezért helye a RAM-ban nem változtatható.

1.2.4. A VIC megszakítási technikája

A VIC megszakítást kérhet akkor, ha sprite háttérrel vagy másik sprite-tal érintkezik, valamint fényceruza kezelése vagy rasztermegszakítás esetén. E lehetőségek választhatók és nem kötelezők.

A *sprite-háttér érintkezéssel* megszakítási módot kedvenc játékprogramjaink jó része alkalmazza. Lényege az, hogy a mozgó sprite haladása közben érintkezésbe kerülhet a képernyőn látható más, nem mozgó ábrákkal, feliratokkal. Ekkor a VIC az IRQ vezetéket alacsonyra állítja, így jelzi a processzornak a megszakításkérélmét. A többi már a processzor és az éppen

futó program megszakításrutinjának dolga. A megszakítási módot a következő módon aktivizálhatjuk:

- az IMR regiszter (\$D01A) 1. bitjét magasra állítjuk,
- a megszakítási rutin elé beiktatjuk saját figyelő rutinunkat,
- a megszakítási vektort a saját rutinunkra állítjuk.

A technikai megvalósításhoz nézzünk egy példát. Tételezzük fel, hogy a képernyőn egy labirintusban kell elvezetni egy kis figurát a bejáratától a kijáratig. Annak programozása, hogy a sprite mikor ér falat és mikor nem, rendkívül körülményes feladat — ráadásul lassú is — a sprite és az aktuális falrészlet koordinátáinak összehasonlítását igénylő feladat esetén. Sokkal célszerűbb, ha az ütközést a VIC figyeli.

A képernyő megrajzolása után le kell futtatni egy kis programot, amely assemblyben akár ilyen is lehet:

INIT	SEI	A megszakítások tiltása
	LDA #NIRQ<	
	STA \$0314	A megszakítási vektor
	LDA #NIRQ>	
	STA \$0315	átállítása
	LDA #%00000010	A maszk
	STA \$D01A	Az IMR átállítása
	CLI	A megszakítások engedélyezése
	RTS	

Természetesen futtatás előtt már a tárban kell lennie az alábbi programnak is:

NIRQ	LDA \$D019	Az IRR
	STA \$D019	Azonnal visszaírni, így törlődik
	AND #%00000010	A maszk
	BEQ A#0	Nincs érintkezés
	LDA #\$01	A jelző
	STA \$FF	beállítása
	JMP \$EA31	A megszakítási rutinnra
A#0	LDA #\$00	A jelző
	STA \$FF	törlése
	JMP \$EA81	Befejezni a megszakítást

A rutin a tár \$00FF címére 1-et tölt, ha a sprite és a háttér érintkezett. Persze ha több sprite-tal dolgozunk, szükségünk lehet arra is, hogy melyik sprite ütközött a háttérrel. Ezt a 31-es regiszter (\$D01F) bitmintája mutatja: a szóban forgó sprite sorszámának megfelelő bit magasra vált. Azaz, ha az 1. sprite ütközött, a \$D01F tartalma \$02 (%00000010).

A megszakításkezeléshez tartozó fontos információ: az IRR regiszter (\$D019) beállítása után mindaddig nem törlődik, míg a regiszterbe nem írnak. Emiatt szükséges olvasás után azonnal felülírni (akár saját magával is), mert csak így biztosítható, hogy a regiszterben mindig aktuális információ legyen.

A *sprite-sprite ütközésre* is érvényesek az előbbieken elmondottak. A folyamat ugyanúgy zajlik le, csupán a megszakításbit és a detektáló regiszter címe más (az IRR 2. bitje, \$D01E).

A VIC megszakítást generál abban az időintervallumban, amikor a *fényceruza* által megjelölt sorba ér a képernyőt felfrissítő elektronsugár. Elméletileg ez abban az időpillanatban

Ellene hogy megtörténjen, amikor az elektronsugár pontosan a fényceruza érzékelője alatt halad el, de ez a hardver néhány s-os késlekedése miatt nem lehetséges. Mindazonáltal leszögezhető: a VIC még azelőtt alacsonyra állítja az IRQ vezetéket, mielőtt az elektronsugár új sort kezdene írni.

A fényceruza aktuális koordinátáit a VIC 19. és 20. regiszteréből (\$D013-14) akkor is kiolvashatjuk, ha a megszakításkérelem nincs aktivizálva. A koordináták olvasását és megjelenítését végző programot gyorsan és könnyen megírhatjuk, például:

```
10 PRINT CHR$(147) PEEK(53267), PEEK(53268) : GOTO 10
```

Amennyiben a fényceruzával megállunk egy ponton, az X (függőleges) koordináta nem változik addig, amíg a fényceruza ismét el nem mozdul. A vízszintes koordinátáról ez sajnos nem mondható el, az időnként elég nagyot változik. Ha pontos értékre van szükség, azt matematikai módszerekkel számíthatjuk ki.

A fényceruza-megszakítási módot a következőképpen kapcsolhatjuk be:

- az IMR regiszter (\$D01A) 3. bitjét magasra állítjuk,
- a megszakítási rutin elé egy megfelelő figyelő rutint állítunk,
- átkapcsoljuk a megszakítási rutin címét (\$0314-0315).

Programban:

```
INIT  SEI                      A megszakítások tiltása
      LDA #NIRQ<
      STA $0314                A megszakítási vektor
      LDA #NIRQ>
      STA $0315                átállítása
      LDA #%00001000          A maszk
      STA $D01A                Az IMR átállítása
      CLI                      A megszakítások engedélyezése
      RTS
```

```
NIRQ  LDA $D019                Az IRR
      STA $D019                Azonnal visszaírni, így törlődik
      AND #%00001000          A maszk
      BEQ AA0                  Nincs érintkezés
      INC $D020                A művelet(ek) helye
      JMP $EA31                A megszakítási rutinra

AA0   JMP $EA81                Befejezni a megszakítást
```

A képernyő kerete jól láthatóan kettéválk abban a sorban, ahol a fényceruza található. A fényceruzát főként grafikus alkalmazásokban használhatjuk (pl. PAINTBOX) szerkesztésre. A fényceruza elsősorban nagy fényerejű tévékészülékeknél és monitoroknál működik kifogástalanul, de azoknál is lehetőleg világos színű legyen a háttér. A fényceruza érzékenységétől erősen függ, hogy milyen az az árnyalat, amelynél még működni képes.

A *rasztersor-megszakítási mód* a legváltozatosabb technikai megoldásokat sugallja. Elve: a VIC másodpercenként 50-szer újítja meg a képet a monitoron. Ez azt jelenti, hogy ugyanazt

a rastersort kb. 20 ms-onként kezdi újraírni. Ez a 20 ms bőven ad arra lehetőséget, hogy állandóan ismétlődő rutinfeladatokat végeztessünk a megszakítási ciklus alatt.

A rastersoros megszakítás kulcsregisztere a VIC 18. regisztere (\$D012). Ebből a regiszterből olvashatjuk ki, hogy az elektronsugár éppen melyik sorban van, illetve ebbe a regiszterbe (\$D012) írhatjuk annak a rastersornak a számát, amelynek felülírása közben a VIC megszakítást fog kérni a processzortól. Ez a regiszter speciális, kettős funkciójú regiszter. chipen belül két részre oszlik: csak írható és csak olvasható regiszterre. A regiszter 9 bite mert a képernyő 256-nál több rastersorból áll. A regiszter 9. bitjét az előző regiszter (\$D011) 7. bitjén keresztül érhetjük el. A regiszterbe írt érték beállítja azt a sort, amelyben a VIC megszakítást fog kérni a processzortól.

A rastersoros megszakítási mód programozása:

- megadjuk a rastersor számát (\$D012),
- beállítjuk az IMR (\$D01A) 0. bitjét,
- megfelelő rutint iktatunk a hardver megszakítás elé,
- aktivizáljuk a megszakítórutint.

1.2.5. A VIC programozása

A rastersoros megszakítás gyakorlati alkalmazása igen változatos lehet. Használhatjuk például két- vagy többszínű háttér, illetve keret esetén, a lehetséges nyolcnál 3-szor több sprite megadásához, vagy a háttéren kívül megjelenített sprite-okhoz.

A *színsávok* kialakítása egyszerű és könnyen követhető példa. A képernyő keretén három szín különül el. A képernyőn jól megfigyelhető a rasztermegszakítás pontossága, mert jól látható, hogyan válnak el a színsávok egymástól. A program két részből áll. Az első rész beállítja a megfelelő regisztereket, és módosítja a hardvermegszakítás vektorát (\$0314-0315). A második rész pedig beállítja a VIC rastersor regiszterét (\$D012) és a képernyő keretének színét (\$D020). A program SYS 10000-rel indítható.

Sokkal látványosabb példája a rasztermegszakításnak a *sprite-sokszorozást* bemutató program. Betöltés és indítás után 24 vándorló sprite-ot számlálhatunk meg. Ez háromszor annyit mint amennyit a VIC engedélyez. A trükk nyitja az, hogy a megfelelő rastersornál a sprite-ok pozícióját megváltoztatjuk, és a képernyő tehetetlenségénél fogva ugyanaz a három sprite nyolc helyen látszik „egyszerre”. A program belső paraméterezése alapján csavarvonalat (forgó csavarvonalat) szimulál. Az egyes sprite-ok közötti távolság módosításával változik a csavarvonal alakja és olykor száma is, hiszen a gömbök akár négy különböző csavarvonalat is formálhatnak.

A program betöltés után a SYS 10000 utasítással indul. Paramétert nem igényel. A program futása közben az F3 és az F1 billentyűvel változtathatjuk meg a gömbök közötti távolságot. Az F1 növeli, az F3 csökkenti a távolságot.


```

2710      100      *=10000      / programkezdet
2710 A9 00      110      LDA #$00      / a szín fekete
2712 9D 20 00      120      STA $0020      / a Keret színe
2715 A9 93      130      LDA #$93      / a Képernyőtörles Kódja
2717 20 D2 FF      140      JSR $FFD2      / a Kód Kiírása/ végrehajtása
271A 78      150      SEI      / a megszakítások tiltása
271B A9 7F      160      LDA #$7F      / a CIA megszakításainak
271D 9D 00 DC      170      STA $0C0D      / tiltása
2720 A9 01      180      LDA #$01      / rasztermegszakítás
2722 9D 1A D0      190      STA $001A      / a VIC maszk regisztere
2725 A9 03      200      LDA #$03      / a ciklus 3 lépéses
2727 85 02      210      STA $02      / a lépésszám tárolása
2729 A9 80      220      LDA #$80      / az első koordináta
272B 8D 12 D0      230      STA $0012      / a raszter R/W regiszterbe
272E A9 1B      240      LDA #$1B
2730 8D 11 D0      250      STA $0011      / VIC 1. ellenőrző-vezérlő regiszter
2733 A9 3F      260      LDA #NIRQ<      / az új rutin címe, alsó byte
2735 8D 14 03      270      STA $0314      / a vektor alsó byte-ja
2738 A9 27      280      LDA #NIRQ>      / az új rutin címe, felső byte
273A 8D 15 03      290      STA $0315      / a vektor felső byte-ja
273D 58      300      CLI      / a megszakítások engedélyezése
273E 60      310      RTS

273F A0 19 D0      320 NIRQ      LDA $D019      / a megszakításkérelem regiszter
2742 8D 19 D0      330      STA $D019      / olvasása és törlése
2745 29 01      340      AND #$01      / rasztermegszakítás ?
2747 F0 19      350      BEQ AA0      / nem
2749 C6 02      360      DEC $02      / a számláló csökkentése
274B 10 04      370      BPL AA1      / még nem futott le
274D A9 02      380      LDA #$02      / a számláló kezdőértékének újratöltése
274F 85 02      390      STA $02
2751 A6 02      400 AA1      LDX $02      / a számláló
2753 8D 69 27      410      LDA COL,X      / az aktuális színkód
2756 8D 20 D0      420      STA $D020      / a színregiszterbe
2759 8D 68 27      430      LDA ROW,X      / az aktuális rasztensor koordináta
275C 8D 12 D0      440      STA $D012      / a raszter R/W regiszterbe
275F 3A      450      TXA      / a számláló lefutott ?
2760 F0 03      460      BEQ AA2      / igen
2762 4C 91 EA      470 AA0      JMP $EA81      / visszatérés a megszakításból

2765 4C 31 EA      480 AA2      JMP $EA31      / az eredeti IRQ rutinra

2768 00 C9 64      490 ROW      .BYTE $00,$C9,$64
276B 05 01 02      500 COL      .BYTE $05,$01,$02
276E      510      .END

```

LEILEN:42 SYMBOLE:6 FEHLER:0

AA0 =2762 AA1 =2751 AA2 =2765 COL =276B NIRQ =273F ROW =2768

```

100 FOR I= 10000 TO 10093 :READ X:POKE I,X:S=S+X:NEXT
110 DATA 169, 0,141, 32,208,169,147, 32,210,255,120,169
120 DATA 127,141, 13,220,169, 1,141, 26,208,169, 3,133
130 DATA 2,169,141,141, 18,208,169, 27,141, 17,208,169
140 DATA 63,141, 20, 3,169, 39,141, 21, 3, 88, 96,173
150 DATA 25,208,141, 25,208, 41, 1,240, 25,198, 2, 16
160 DATA 4,169, 2,133, 2,166, 2,189,107, 39,141, 32
170 DATA 208,129,104, 39,141, 18,208,138,240, 3, 76,129
180 DATA 234, 76, 49,234, 0,201,100, 5, 1, 2
190 IF S <> 9881 THEN PRINT"HIBA A DATASORBAN !":END
200 PRINT"OK !!":END

```

2710			100		*=10000	
0092			110	REG0	=\$92	; az Y táblázat mutatója
0036			120	REG1	=\$96	; sprite koordináta regiszterek mutatója
009B			130	REG2	=\$9B	; a sprite-ok közötti különbség
2710	A9	00	140		LDA #\$00	; a képernyő színe fekete
2712	8D	20 D0	150		STA \$0020	; keret
2715	8D	21 D0	160		STA \$0021	; háttér
2718	A2	FF	170		LDX #\$FF	; 255
271A	86	96	180		STX REG1	
271C	E8		190		INX	; 0
271D	86	92	200		STX REG0	
271F	E8		210		INX	; 1
2720	86	9B	220		STX REG2	
2722	A9	07	230		LDA #\$07	; %00000111, az első három
2724	8D	15 D0	240		STA \$0015	; sprite engedélyezése
2727	A9	0F	250		LDA #\$0F	; a sprite színe szürke
2729	8D	27 D0	260		STA \$0027	; az első
272C	8D	28 D0	270		STA \$0028	; három
272F	8D	29 D0	280		STA \$0029	; sprite színe
2732	20	06 27	290		JSR SR0	; az X pozíciók betöltése
2735	78		300		SEI	; a megszakítások
2736	A9	1B	310		LDA #\$1B	
2738	8D	11 D0	320		STA \$0011	; VIC 1. ellenőrző-vezérlő regiszter
273B	A9	7F	330		LDA #\$7F	; a CIA megszakításának
273D	8D	0D DC	340		STA \$DC0D	; kikapcsolása
2740	A9	51	350		LDA #NIRQ<	; az új rutin címe, alsó byte
2742	A2	27	360		LDX #NIRQ>	; az új rutin címe, felső byte
2744	8D	14 03	370		STA \$0314	; az IRQ-vektor
2747	8E	15 03	380		STX \$0315	; beállítása
274A	A9	01	390		LDA #\$01	; rásztermegszakítás
274C	8D	1A D0	400		STA \$001A	; az IMR beállítása
274F	58		410		CLI	; a megszakítások engedélyezése
2750	60		420		RTS	
2751	A9	01	430	NIRQ	LDA #\$01	
2753	8D	19 D0	440		STA \$0019	; az IRR törlése
2756	8D	1A D0	450		STA \$001A	; az IMR beállítása, rásztermegszakítás
2759	E6	92	460		INC REG0	; az X pozíció táblázatán belüli mutató
275B	A6	92	470		LDX REG0	
275D	8D	01 28	480		LDA SYP,X	; a függőleges koordináta
2760	C9	F0	490		CMP #\$F0	; elérte már az utolsó pozíciót ?
2762	D0	06	500		BNE AA0	; még nem, ugrás
2764	A9	00	510		LDA #\$00	; a mutató
2766	85	92	520		STA REG0	; alaphelyzetre
2768	F0	33	530		BEQ BA0	; feltétel nélküli ugrás
276A	E6	96	540	AA0	INC REG1	; a mutató
276C	E6	96	550		INC REG1	; kettővel növekszik
276E	A5	96	560		LDA REG1	; a mutató
2770	C9	07	570		CMP #\$07	; elérte már a 7-et ?
2772	D0	04	580		BNE AA2	; még nem, ugrás
2774	A9	01	590		LDA #\$01	; elérte, felvenni a
2776	85	96	600		STA REG1	; kezdőértéket
2778	A6	92	610	AA2	LDX REG0	
277A	8D	01 28	620		LDA SYP,X	; a sprite-hoz tartozó Y pozíció
277D	A4	96	630		LDY REG1	; az aktuális mutató
277F	99	00 D0	640		STA \$0000,Y	; az Y pozíciók beállítása
2782	FE	E8 27	650		INC SXP,X	
2785	8D	E8 27	660		LDA SXP,X	; a mutató kiemelése
2788	A8		670		TAY	
2789	89	1B 28	680		LDA SPS,Y	; a pozíció a táblázatból
278C	A4	96	690		LDY REG1	; az aktuális mutató
278E	99	FF CF	700		STA \$CFFF,Y	; az X pozíciók beállítása
2791	8D	01 28	710		LDA SYP,X	; a sprite pozíció Y koordinátája
2794	18		720		CLC	
2795	69	03	730		ADC #\$03	; +3
2797	8D	12 D0	740		STA \$0012	; a rászter R/W regiszterbe
279A	4C	81 EA	750		JMP \$EA81	; befejezni a megszakítást
279D	A9	01	760	BA0	LDA #\$01	
279F	8D	19 D0	770		STA \$0019	; az IRR törlése

27A2	8D	1A	D0	780		STA \$D01A	; maszkbeállítás, rasztermegszakítás
27A5	A9	14		790		LDA #\$14	; kezdőérték
27A7	8D	12	D0	800		STA \$D012	; a raszter R/W beállítása
27AA	A9	A5		810		LDA #\$A5	; a sprite-minta mutatója, 165*64=10560
27AC	8D	F8	07	820		STA \$07F8	; a 0. sprite
27AF	8D	F9	07	830		STA \$07F9	; az 1. sprite
27B2	8D	FA	07	840		STA \$07FA	; a 2. sprite mutatója
27B5	A5	C5		850		LDA \$C5	; a lenyomott billentyű
27B7	C9	04		860		CMP #\$04	; F1 ?
27B9	D0	05		870		BNE AA8	; nem, tovább
27BB	E6	9B		880		INC REG2	; a különbség növelése
27BD	4C	C6	27	890		JMP AB1	
27C0	C9	05		900	AA8	CMP #\$05	; F3 ?
27C2	D0	0F		910		BNE AA9	; nem
27C4	C6	9B		920		DEC REG2	; a különbség csökkentése
27C6	20	D6	27	930	AB1	JSR SR0	; az X pozíciók betöltése
27C9	A2	34		940		LDX #\$34	
27CB	A0	FF		950	AB4	LDY #\$FF	
27CD	88			960	AB3	DEY	; késleltető
27CE	D0	FD		970		BNE AB3	
27D0	CA			980		DEX	; ciklus
27D1	D0	F8		990		BNE AB4	
27D3	4C	31	EA	1000	AA9	JMP \$EA31	; az eredeti IRQ-rutinra
27D6	A2	00		1010	SR0	LDX #\$00	
27D8	A0	00		1020		LDY #\$00	
27DA	8A			1030	AB2	TXA	
27DB	93	E8	27	1040		STA SXP,Y	
27DE	18			1050		CLC	
27DF	65	9B		1060		ADC REG2	; a különbség hozzáadása
27E1	AA			1070		TAX	
27E2	C8			1080		INY	
27E3	C0	19		1090		CPY #\$19	; a 25. művelet ?
27E5	D0	F3		1100		BNE AB2	; még nem, vissza
27E7	60			1110		RTS	
2801				1120	SXP	*=**+25	
2801	28	30	33	1130	SYP	.BYTE 40, 48, 56, 64, 72, 80	
2807	53	60	68	1140		.BYTE 88, 96, 104, 112, 120, 128	
280D	88	90	98	1150		.BYTE 136, 144, 152, 160, 168, 176	
2813	B8	C0	C8	1160		.BYTE 184, 192, 200, 208, 216, 224	
2819	E8	F0		1170		.BYTE 232, 240	
281B	8C	91	96	1180	SP3	.BYTE 140, 145, 150, 156, 161, 166, 171, 177, 182, 187, 191, 196	
2827	C9	CD	D1	1190		.BYTE 201, 205, 209, 213, 217, 221, 225, 228, 231, 234, 237, 239	
2833	F1	F3	F5	1200		.BYTE 241, 243, 245, 246, 247, 248, 249, 249, 250, 249, 249, 248	
283F	F7	F6	F5	1210		.BYTE 247, 246, 245, 243, 241, 239, 237, 234, 231, 228, 225, 221	
284B	D3	D5	D1	1220		.BYTE 217, 213, 209, 205, 201, 196, 191, 187, 182, 177, 171, 166	
2857	A1	9C	96	1230		.BYTE 161, 156, 150, 145, 140, 134, 129, 123, 118, 113, 108, 102	
2863	61	5C	58	1240		.BYTE 97, 92, 88, 83, 78, 74, 70, 66, 62, 58, 54, 51	
286F	30	2D	2A	1250		.BYTE 48, 45, 42, 40, 38, 36, 34, 33, 32, 31, 30, 30	
287B	1E	1E	1E	1260		.BYTE 30, 30, 30, 31, 32, 33, 34, 36, 38, 40, 42, 45	
2887	30	33	36	1270		.BYTE 48, 51, 54, 58, 62, 66, 70, 74, 78, 83, 88, 92	
2893	61	66	6C	1280		.BYTE 97, 102, 108, 113, 118, 123, 129, 134, 140, 145, 150, 156	
289F	A1	A6	AB	1290		.BYTE 161, 166, 171, 177, 182, 187, 191, 196, 201, 205, 209, 213	
28AB	D3	DD	E1	1300		.BYTE 217, 221, 225, 228, 231, 234, 237, 239, 241, 243, 245, 246	
28B7	F7	F3	F9	1310		.BYTE 247, 248, 249, 249, 250, 249, 249, 248, 247, 246, 245, 243	
28C3	F1	EF	ED	1320		.BYTE 241, 239, 237, 234, 231, 228, 225, 221, 217, 213, 209, 205	
28CF	C9	C4	BF	1330		.BYTE 201, 196, 191, 187, 182, 177, 171, 166, 161, 156, 150, 145	
28DB	8C	86	81	1340		.BYTE 140, 134, 129, 123, 118, 113, 108, 102, 97, 92, 88, 83	
28E7	4E	4A	46	1350		.BYTE 78, 74, 70, 66, 62, 58, 54, 51, 48, 45, 42, 40	
28F3	26	24	22	1360		.BYTE 38, 36, 34, 33, 32, 31, 30, 30, 30, 30, 30, 31	
28FF	20	21	22	1370		.BYTE 32, 33, 34, 36, 38, 40, 42, 45, 48, 51, 54, 58	
290B	3E	42	46	1380		.BYTE 62, 66, 70, 74, 78, 83, 88, 92, 97, 102, 108, 113	
2917	76	7B	81	1390		.BYTE 118, 123, 129, 134	
2940				1400		*=10560	
2940	00	00	00	1410		.BYTE 0, 0, 0, 0, 0, 0, 0, 0, 255, 0, 3, 129, 132	


```

234C 0E 7E 70 1420 .BYTE 14,126,112, 25,255,152, 55,255,236,111,255,246
2356 6F FF F6 1430 .BYTE 111,255,246,223,255,251,223,255,251,223,255,251
2364 6F FF F6 1440 .BYTE 111,255,246,111,255,246, 55,255,236, 25,255,152
2370 0E 7E 70 1450 .BYTE 14,126,112, 3,129,192, 0,255, 0, 0, 0, 0
237C 00 00 00 1460 .BYTE 0, 0, 0, 0
2380 1470 .END

```

ZEILEN: 138 SYMBOLE: 17 FEHLER: 0

```

AA0  =276A  AA2  =2778  AA8  =27C0  AA9  =27D3  AB1  =27C6  AB2  =27DA
AB3  =27CD  AB4  =27CB  BA0  =279D  NIRQ  =2751  REG0  =0092  REG1  =0036
REG2  =009B  SPS  =281B  SR0  =27D6  SXP  =27E8  SYP  =2801

```

```

100 FOR I= 10000 TO 10624 :READ X:POKE I,X:S=S+X:NEXT
110 DATA 169, 0,141, 32,208,141, 33,208,162,255,134,150
120 DATA 134,155,232,134,146,232,134,177,169, 7,141, 21
130 DATA 208,169, 15,141, 39,208,141, 40,208,141, 41,208
140 DATA 162,129,160, 0,138,153, 0, 40, 24,101,177,170
150 DATA 200,192, 25,208,243,120,169, 27,141, 17,208,169
160 DATA 127,141, 13,220,169, 97,162, 39,141, 20, 3,142
170 DATA 21, 3,169, 1,141, 26,208, 88, 96,169, 1,141
180 DATA 25,208,141, 26,208,230,146,166,146,189, 25, 40
190 DATA 201,240,208, 6,169, 0,133,146,240, 63,230,150
200 DATA 230,150,165,150,201, 7,208, 4,169, 1,133,150
210 DATA 230,155,165,155,201, 3,208, 4,169, 0,133,155
220 DATA 166,146,189, 25, 40,164,150,153, 0,208,254, 0
230 DATA 40,189, 0, 40,168,185, 51, 40,164,150,153,255
240 DATA 207,189, 25, 40, 24,105, 3,141, 18,208, 76,129
250 DATA 234,163, 1,141, 25,208,141, 26,208,169, 20,141
260 DATA 18,208,169,165,141,248, 7,141,249, 7,141,250
270 DATA 7,165,197,201, 4,208, 5,230,177, 76,226, 39
280 DATA 201, 5,208, 29,198,177,162, 0,160, 0,138,153
290 DATA 0, 40, 24,101,177,170,200,192, 25,208,243,162
300 DATA 52,160,255,136,208,253,202,208,248, 76, 49,234
310 DATA 102,102,102,102,102,102,102,102,102,102,102,102
320 DATA 102,102,102,102,102,102,102,102,102,102,102,102
330 DATA 102, 40, 48, 56, 64, 72, 80, 88, 36,104,112,120
340 DATA 128,136,144,152,160,168,176,184,192,200,208,216
350 DATA 224,232,240,140,145,150,156,161,166,171,177,182
360 DATA 187,191,196,201,205,209,213,217,221,225,228,231
370 DATA 234,237,239,241,243,245,246,247,248,249,249,250
380 DATA 249,249,248,247,246,245,243,241,239,237,234,231
390 DATA 228,225,221,217,213,209,205,201,196,191,187,182
400 DATA 177,171,166,161,156,150,145,140,134,129,123,118
410 DATA 113,108,102, 97, 92, 88, 83, 78, 74, 70, 66, 62
420 DATA 58, 54, 51, 48, 45, 42, 40, 38, 36, 34, 33, 32
430 DATA 31, 30, 30, 30, 30, 30, 31, 32, 33, 34, 36, 38
440 DATA 40, 42, 45, 48, 51, 54, 58, 62, 66, 70, 74, 78
450 DATA 83, 88, 92, 97,102,108,113,118,123,129,134,140
460 DATA 145,150,156,161,166,171,177,182,187,191,196,201
470 DATA 205,209,213,217,221,225,228,231,234,237,239,241
480 DATA 243,245,246,247,248,249,249,250,249,249,248,247
490 DATA 246,245,243,241,239,237,234,231,228,225,221,217
500 DATA 213,209,205,201,196,191,187,182,177,171,166,161
510 DATA 156,150,145,140,134,129,123,118,113,108,102, 97
520 DATA 92, 88, 83, 78, 74, 70, 66, 62, 58, 54, 51, 48
530 DATA 45, 42, 40, 38, 36, 34, 33, 32, 31, 30, 30, 30
540 DATA 30, 30, 31, 32, 33, 34, 36, 38, 40, 42, 45, 48
550 DATA 51, 54, 58, 62, 66, 70, 74, 78, 83, 88, 92, 97
560 DATA 102,108,113,118,123,129,134,102,102,102,102,102
570 DATA 102,102,102,102,102,102,102,102, 0, 0, 0, 0
580 DATA 0, 0, 0,255, 0, 3,129,192, 14,126,112, 25
590 DATA 255,152, 55,255,236,111,255,246,111,255,246,223
600 DATA 255,251,223,255,251,223,255,251,111,255,246,111
610 DATA 255,246, 55,255,236, 25,255,152, 14,126,112, 3
620 DATA 129,192, 0,255, 0, 0, 0, 0, 0, 0, 0, 0
630 DATA 153
640 IF S (>) 82216 THEN PRINT "HIBA A DATASORBAN !":END
650 PRINT "OK !!":END
READY.

```

Sprite-ok a képernyő keretében is elhelyezhetők. Ennek igen szemléletes példája az időpont írás.

A C64-esben két beépített AM/PM óra is van, amelyek hosszú távon meglehetősen pontosak, ha a hálózati frekvencia nem tér el jelentősen 50 Hz-től. Az órák bármelyikét felhasználhatjuk. A probléma abban jelentkezik, hogy az időt látni is szeretnénk. A szerkesztőmenüben megjelenített kijelzés, akárhová tesszük is, munka közben csak bajt okoz. Egyetlen megoldás létezik: a kijelzést a keretre kell helyezni. Ez normál körülmények között nem lehetséges. Segít azonban a rasztermegszakítás.

Példaprogramunk betöltése és elindítása után bemutatja, hogy a VIC-et is be lehet csapni. Amikor az elektronsugár egy meghatározott (a 247.) sorba ér, a VIC majdnem összes regiszterét (ill. azok tartalmát) kicseréljük. Emiatt a VIC azt „hiszi”, hogy az elektronsugár még mindig a szerkesztőmezőben van, ezért a sprite-ok láthatóságát továbbra is engedélyezi. Ez a trükk titka. Amint az elektronsugár az időt mutató sprite-okat kirajzolta, az eredeti helyezésre állítjuk vissza. Így a felhasználót, miközben folyamatosan láthatja az időt, nem zavarja munkájában a kijelzés.

A program SYS 51200, „120813” utasítással indul. A „120813” természetesen bármilyen más időpont is lehet; az óraidőt bármikor módosíthatjuk. Az óra működése közben bármilyen program futhat, amely nem módosítja a megszakítást. A perifériákat problémamentesen használhatjuk (a kazettás egységet kivéve).

C800	100	*=\$C800	; programkezdet
D01A	110	IRMASK =\$D01A	; VIC megszakítás maszkolás
D012	120	RASTER =\$D012	; VIC raszterkoordináta regiszter
D000	130	VIC =\$D000	; VIC báziscím
DC0D	140	ICR1 =\$DC0D	; CIA1 ICR, megszakításregiszter
DD08	150	TODTHS =\$DD08	; TOD tizedmásodpercek
DD00	160	PRA2 =\$DD00	; CIA2 A portregiszter
DD0B	170	TODHRS =\$DD0B	; TOD órák
DD0A	180	TODMIN =\$DD0A	; TOD percek
DD09	190	TODSEC =\$DD09	; TOD másodpercek
DD0E	200	CRA2 =\$DD0E	; CIA2 CRA, A időzítő vezérlő
D013	210	IRR =\$D013	; VIC megszakításkérelem
0073	220	CHRGOT =\$0073	
AEFD	230	CHKCOM =\$AEFD	; vessző ellenőrzése
AD3E	240	FRMEVL =\$AD3E	; a kifejezés kiértékelése
B6A3	250	FRESTR =\$B6A3	; szövegkezelés
C800 20 73 00	260	JSR CHRGOT	; van következő byte ?
C803 F0 6C	270	BEQ END	; nincs, befejezni
C805 20 FD AE	280	JSR CHKCOM	; vessző ellenőrzése
C808 20 3E AD	290	JSR FRMEVL	; a kifejezés kiértékelése
C80B 20 A3 B6	300	JSR FRESTR	; a paraméter hossza
C80E C9 06	310	CMP #\$06	; 6-tal egyenlő ?
C810 D0 60	320	BNE AA0	; nem, hiba
C812 A0 FF	330	LDY #\$FF	; beolvasás offset
C814 A2 03	340	LDX #\$03	; a beolvasandó érték felső határa/10
C816 20 75 C8	350	JSR HMS	; az órák beolvasása
C819 05 02	360	ORA \$02	; nulla óra ?
C81B D0 04	370	BNE AA1	; nem, ugrás
C81D A9 32	380	LDA #\$92	; a nulla óra kódja
C81F D0 0F	390	BNE AA2	; feltétel nélküli ugrás
C821 C9 24	400	CMP #\$24	; nagyobb, mint 24 ?
C823 B0 4D	410	BCS AA0	; igen, hiba
C825 C9 13	420	CMP #\$13	; az óraidő kisebb, mint 13 ?
C827 30 07	430	BCC AA2	; igen, ugrás
C829 38	440	SEC	; a túlcsondulás törlése
C82A F8	450	SED	; a BCD-mód bekapcsolása
C82B F8 10	460	SED	; a BCD-mód bekapcsolása

C82D	D8		470		CLD	/ a BCD-mód megszüntetése
C82E	03	80	480		ORA #\$80	/ a 7. bit beállítása
C830	8D	0B DD	490	AA2	STA TODHRS	/ az óraidő regiszter feltöltése
C833	A2	06	500		LDX #\$06	/ a beolvasandó érték felső határa/10
C835	20	75 C8	510		JSR HMS	/ a percek beolvasása
C838	8D	0A DD	520		STA TODMIN	/ a perc regiszter feltöltése
C83B	20	75 C8	530		JSR HMS	/ a másodpercek beolvasása
C83E	8D	09 DD	540		STA TODSEC	/ a másodperc regiszter feltöltése
C841	A9	00	550		LDA #\$00	
C843	8D	08 DD	560		STA TODTHS	/ a tizedmásodpercek törölve, óra indul
C846	78		570		SEI	/ a megszakítások tiltása
C847	A3	7F	580		LDA #\$7F	/ minden maszk bit törlése
C849	8D	0D DC	590		STA ICR1	/ az ICR-ben
C84C	A3	94	600		LDA #TIME<	/ az új rutin
C84E	8D	14 03	610		STA \$0314	
C851	A3	C8	620		LDA #TIME>	/ vektorának beállítása
C853	8D	15 03	630		STA \$0315	
C856	A9	01	640		LDA #\$01	/ rasztermegszakítás beállítása
C858	8D	1A D0	650		STA IRMASK	/ VIC megszakításregiszter
C85B	A9	F7	660		LDA #\$F7	/ 247
C85D	8D	12 D0	670		STA RASTER	/ raszter R/W regiszter
C860	A2	2F	680		LDX #\$2F	
C862	8D	00 D0	690	A1	LDA VIC,X	/ a VIC regisztereinek
C865	9D	F2 C9	700		STA PFR2,X	/ mentése
C868	CA		710		DEX	
C869	10	F7	720		BPL A1	/ vissza, még nem futott le
C86B	A9	00	730		LDA #\$00	
C86D	8D	08 DD	740		STA TODTHS	/ óra indul
C870	58		750		CLI	/ a megszakítások engedélyezése
C871	60		760	END	RTS	
C872	4C	48 B2	770	AA0	JMP \$B248	/ ILLEGAL QUANTITY ERROR
C875	86	FF	780	HMS	STX \$FF	/ a határ tárolása
C877	C8		790		INY	/ a mutató növelése
C878	B1	22	800		LDA (\$22),Y	/ a következő byte beolvasása
C87A	38		810		SEC	
C87B	E9	30	820		SBC #\$30	/ átalakítás ASCII-kódról
C87D	C5	FF	830		CMP \$FF	/ összehasonlítás a határral
C87F	B0	F1	840		BCS AA0	/ nagyobb, hiba
C881	0A		850		ASL A	
C882	0A		860		ASL A	/ az alsó négy bit
C883	0A		870		ASL A	
C884	0A		880		ASL A	/ áthelyezése a felső félbyte-ba
C885	85	02	890		STA \$02	/ a félbyte tárolása
C887	C8		900		INY	
C888	B1	22	910		LDA (\$22),Y	/ a következő byte beolvasása
C88A	38		920		SEC	
C88B	E9	30	930		SBC #\$30	/ átalakítás ASCII-kódról
C88D	C9	0A	940		CMP #\$0A	/ nagyobb, mint 9 ?
C88F	B0	E1	950		BCS AA0	/ igen, hiba
C891	05	02	960		ORA \$02	/ a felső félbyte átvétele
C893	60		970		RTS	
C894	78		980	TIME	SEI	/ a megszakítások tiltása
C895	AD	19 D0	990		LDA IRR	/ a megszakításregiszter
C898	8D	19 D0	1000		STA IRR	/ visszaírva törlődik
C89B	AD	12 D0	1010		LDA RASTER	/ a VIC raszter R/W regisztere
C89E	C9	F7	1020		CMP #\$F7	/ értéke 247 ?
C8A0	D0	32	1030		BNE A2	/ nem, ugrás
C8A2	A2	18	1040		LDX #\$18	/ 25 regiszter
C8A4	8D	00 D0	1050	A4	LDA VIC,X	/ a VIC regisztere
C8A7	9D	F2 C9	1060		STA PFR2,X	/ a regiszterpufferbe
C8AA	8D	C1 C9	1070		LDA PFR1,X	/ az új értékek
C8AD	9D	00 D0	1080		STA VIC,X	/ betöltése
C8B0	CA		1090		DEX	/ a ciklus lefutott ?
C8B1	10	F1	1100		BPL A4	/ még nem, vissza

C8B8	8D	00	D0	1130		STA	PRA2		
C8BB	A9	00		1140		LDA	#\$00	/	az új rasztermegszakítás
C8BD	8D	12	D0	1150		STA	RASTER	/	sora
C8C0	A2	13		1160		LDX	#\$13	/	újabb 20 regiszter
C8C2	BD	1B	D0	1170	A3	LDA	VIC+27,X		
C8C5	9D	0D	CA	1180		STA	PFR2+27,X		a regiszterpufferbe
C8C8	BD	DC	C9	1190		LDA	PFR1+27,X		új értékek a
C8CB	9D	1B	D0	1200		STA	VIC+27,X		regiszterekbe
C8CE	CA			1210		DEX		/	a ciklus lefutott ?
C8CF	10	F1		1220		BPL	A3	/	még nem, vissza
C8D1	4C	FB	C8	1230		JMP	A5		
C8D4	A2	1B		1240	A2	LDX	#\$1B		
C8D6	BD	F2	C9	1250	A6	LDA	PFR2,X	/	a VIC regisztereinek
C8D9	9D	0D	D0	1260		STA	VIC,X	/	újrátöltése
C8DC	CA			1270		DEX			
C8DD	10	F7		1280		BPL	A6		
C8DF	A9	F7		1290		LDA	#\$F7	/	247
C8E1	8D	12	D0	1300		STA	RASTER	/	a raszter R/W regiszterbe
C8E4	A2	13		1310		LDX	#\$13		
C8E6	BD	DC	C9	1320	A7	LDA	PFR1+27,X		a VIC felső regisztereinek
C8E9	9D	1B	D0	1330		STA	VIC+27,X		újrátöltése
C8EC	CA			1340		DEX			
C8ED	10	F7		1350		BPL	A7		
C8EF	AD	0D	D0	1360		LDA	PRA2	/	a CIA2 PRA
C8F2	09	03		1370		ORA	#\$03	/	visszaállítása
C8F4	8D	0D	D0	1380		STA	PRA2		
C8F7	5B			1390		CLI		/	a megszakítások engedélyezése
C8F8	4C	31	EA	1400		JMP	EA31	/	az eredeti IRQ-ra
C8FB	A0	0D		1410	A5	LDY	#\$0D		
C8FD	AD	0B	D0	1420		LDA	TODHRS	/	az óraregiszter beolvasása
C900	29	10		1430		AND	#\$10	/	a tízes órák maszkolása
C902	1B			1440		CLC			
C903	4A			1450		LSR	A	/	a felső félbyte
C904	4A			1460		LSR	A		
C905	4A			1470		LSR	A	/	áttolása az alsóba
C906	4A			1480		LSR	A		
C907	69	8D		1490		ADC	#\$8D	/	a 7. bit beállítása
C909	4B			1500		PHA		/	és átmeneti tárolás a veremben
C90A	AD	0B	D0	1510		LDA	TODHRS	/	az óraregiszter újrabeolvasása
C90D	C9	12		1520		CMP	#\$12	/	0 óra ?
C90F	D0	09		1530		BNE	A8	/	nem, ugrás
C911	A9	8D		1540		LDA	#\$8D	/	igen, a sprite a 0 számjegy
C913	8D	F8	C7	1550		STA	7F8	/	a 0. sprite mutatója
C916	6B			1560		PLA			
C917	4C	2B	C9	1570		JMP	B0		
C91A	C9	92		1580	A8	CMP	#\$92	/	12 óra ?
C91C	D0	09		1590		BNE	A9	/	nem, ugrás
C91E	A9	81		1600		LDA	#\$81	/	igen, a sprite az 1 számjegy
C920	8D	F8	C7	1610		STA	7F8	/	a 0. sprite mutatója
C923	6B			1620		PLA			
C924	4C	2B	C9	1630		JMP	B0		
C927	6B			1640	A9	PLA		/	az elmentett regiszter
C928	8D	F8	C7	1650		STA	7F8	/	a 0. sprite mutatója
C92B	AD	0B	D0	1660	B0	LDA	TODHRS	/	az óraregiszter ismételt beolvasása
C92E	29	0F		1670		AND	#\$0F	/	a felső félbyte beolvasása
C930	69	8D		1680		ADC	#\$8D	/	a 7. bit beállítása
C932	4B			1690		PHA			
C933	AD	0B	D0	1700		LDA	TODHRS	/	az óraregiszter ismételt beolvasása
C936	C9	12		1710		CMP	#\$12	/	0 óra ?
C938	D0	09		1720		BNE	B1	/	nem
C93A	A9	8D		1730		LDA	#\$8D	/	igen, a sprite a 0 számjegy
C93C	8D	F9	C7	1740		STA	7F9	/	az 1. sprite mutatója
C93F	6B			1750		PLA			
C940	4C	54	C9	1760		JMP	B2	/	folytatni

C943	C9	92		1770	B1	CMP #\$92	; 12 óra ?
C945	D0	09		1780		BNE B3	; nem
C947	A9	82		1790		LDA #\$82	; igen, a sprite a 2. számjegy
C949	8D	F9	C7	1800		STA \$C7F9	; az 1. sprite mutatója
C94C	68			1810		PLA	
C94D	4C	54	C9	1820		JMP B2	; folytatni
C950	68			1830	B3	PLA	
C951	20	9C	C3	1840		JSR B4	
C954	A9	8A		1850	B2	LDA #\$8A	; a Kettőspont mutatója
C956	8D	FA	C7	1860		STA \$C7FA	; a 2. sprite területmutatója
C959	AD	0A	DD	1870		LDA TODMIN	; az óra percei
C95C	29	F0		1880		AND #\$F0	; az alsó négy bit levágása
C95E	4A			1890		LSR A	
C95F	4A			1900		LSR A	; a felső félbyte
C960	4A			1910		LSR A	
C961	4A			1920		LSR A	; áttolása az alsóba
C962	69	80		1930		ADC #\$80	; 128 hozzáadása
C964	8D	FB	C7	1940		STA \$C7FB	; a 3. sprite területmutatója
C967	AD	0A	DD	1950		LDA TODMIN	; a percek ismételt beolvasása
C96A	29	0F		1960		AND #\$0F	; a felső négy bit levágása
C96C	69	80		1970		ADC #\$80	; 128 hozzáadása
C96E	8D	FC	C7	1980		STA \$C7FC	; a 4. sprite területmutatója
C971	A9	8A		1990		LDA #\$8A	; a Kettőspont mutatója
C973	8D	FD	C7	2000		STA \$C7FD	; az 5. sprite területmutatója
C976	AD	09	DD	2010		LDA TODSEC	; az óra másodpercei
C979	29	F0		2020		AND #\$F0	; az alsó négy bit levágása
C97B	4A			2030		LSR A	
C97C	4A			2040		LSR A	; a felső félbyte
C97D	4A			2050		LSR A	
C97E	4A			2060		LSR A	; áttolása az alsóba
C97F	69	80		2070		ADC #\$80	; 128 hozzáadása
C981	8D	FE	C7	2080		STA \$C7FE	; a 6. sprite területmutatója
C984	AD	09	DD	2090		LDA TODSEC	; ismételt beolvasás
C987	29	0F		2100		AND #\$0F	; a felső négy bit levágása
C989	69	80		2110		ADC #\$80	; 128 hozzáadása
C98B	8D	FF	C7	2120		STA \$C7FF	; a 7. sprite területmutatója
C98E	AD	0E	DD	2130		LDA CRA2	; CIA2 CRA, A időzítő vezérlő
C991	09	80		2140		ORA #\$80	; az óra léptéke 50 Hz
C993	8D	0E	DD	2150		STA CRA2	
C996	AD	08	DD	2160		LDA TODTHS	; a tizedmásodpercek beolvasása
C999	4C	81	EA	2170		JMP \$EAS1	; befejezni a megszakítást
C99C	8D	F9	C7	2180	B4	STA \$C7F9	; az 1. sprite mutatója
C99F	AD	08	DD	2190		LDA TODHRS	; az óraregiszter
C9A2	E9	80		2200		SBC #\$80	; 128 levonása, a 'délután' bit törlés
C9A4	10	01		2210		BPL B5	; pozitív, ugrás
C9A6	60			2220		RTS	
C9A7	EE	F8	C7	2230	B5	INC \$C7F8	; a 0. sprite mutatója
C9AA	EE	F9	C7	2240		INC \$C7F9	; az 1. sprite mutatója
C9AD	EE	F9	C7	2250		INC \$C7F9	; növelés 2-vel
C9B0	AD	F9	C7	2260		LDA \$C7F9	; a mutató
C9B3	E9	8A		2270		SBC #\$8A	; nagyobb 138-nál ?
C9B5	10	01		2280		BPL B6	; igen
C9B7	60			2290		RTS	
C9B8	69	7F		2300	B6	ADC #\$7F	; 127 hozzáadása
C9BA	8D	F9	C7	2310		STA \$C7F9	; a mutató újraindítása
C9BD	EE	F8	C7	2320		INC \$C7F8	; a 0. sprite mutatójának növelése
C9C0	60			2330		RTS	
C9C1	57	0B		2340	PFR1	.BYTE 87,11	
C9C3	70	0B		2350		.BYTE 112,11	
C9C5	89	0B		2360		.BYTE 137,11	
C9C7	A2	0B		2370		.BYTE 162,11	
C9C9	BB	0B		2380		.BYTE 187,11	
C9CB	D4	0B		2390		.BYTE 212,11	
C9CD	ED	0B		2400		.BYTE 237,11	

C9CF	07	0B	2410	.BYTE	7,11
C9D1	80		2420	.BYTE	\$80
C9D2	17		2430	.BYTE	\$17
C9D3	59	9A 36	2440	.BYTE	\$59,\$9A,\$36
C9D6	FF		2450	.BYTE	\$FF
C9D7	C8		2460	.BYTE	\$C8
C9D8	00		2470	.BYTE	\$00
C9D9	15		2480	.BYTE	\$15
C9DA	79	F0	2490	.BYTE	\$79,\$F0
C9DC	00	00 00	2500	.BYTE	\$00,\$00,\$00,\$00,\$00
C9E1	F0		2510	.BYTE	\$F0
C9E2	F0		2520	.BYTE	\$F0
C9E3	F1	F2 F3	2530	.BYTE	\$F1,\$F2,\$F3,\$F0,\$F0
C9E8	0F	0F 0F	2540	.BYTE	\$0F,\$0F,\$0F,\$0F
C9EC	0F	0F 0F	2550	.BYTE	\$0F,\$0F,\$0F,\$0F
C9F0	02	FF	2560	.BYTE	\$02,\$FF
C9F2		2570 PFR2	=*		
CA22		2580	*=**+\$30		
E000		2590	*=\$E000		
E000	07	FE 00	2600	.BYTE	7,254, 0, 31,255,128, 63,255,192,120, 1
E008	E0	70 00	2610	.BYTE	224,112, 0,224,112, 0,224,112, 0,224,112, 0
E017	E0	70 00	2620	.BYTE	224,112, 0,224,112, 0,224,112, 0,224,112, 0
E023	E0	70 00	2630	.BYTE	224,112, 0,224,112, 0,224,112, 0,224,112, 0
E02F	E0	70 00	2640	.BYTE	224,112, 0,224,120, 1,224, 63,255,192, 31,255
E038	80	07 FE	2650	.BYTE	128, 7,254, 0, 0, 0, 7,128, 0, 15,128, 0
E047	1F	80 00	2660	.BYTE	31,128, 0, 63,128, 0,123,128, 0,243,128, 1
E053	E3	80 03	2670	.BYTE	227,128, 3,195,128, 0, 3,128, 0, 3,128, 0
E05F	03	80 00	2680	.BYTE	3,128, 0, 3,128, 0, 3,128, 0, 3,128, 0
E06B	03	80 00	2690	.BYTE	3,128, 0, 3,128, 0, 3,128, 0, 3,128, 0
E077	03	80 00	2700	.BYTE	3,128, 0, 3,128, 0, 3,128, 0, 15,255, 0
E083	3F	FF C0	2710	.BYTE	63,255,192,127,255,224,240, 0,240,224, 0,112
E08F	E0	00 70	2720	.BYTE	224, 0,112, 0, 0,112, 0, 0,240, 0, 1,224
E09B	00	07 C0	2730	.BYTE	0, 7,192, 0, 31, 0, 0,124, 0, 1,240, 0
E0A7	07	C0 00	2740	.BYTE	7,192, 0, 31, 0, 0,124, 0, 0,240, 0, 48
E0B3	F0	00 30	2750	.BYTE	240, 0, 48,255,255,240,255,255,240,255,255,240
E0BF	00	0F FF	2760	.BYTE	0, 15,255, 0, 63,255,192,127,255,224,240, 0
E0CB	F0	E0 00	2770	.BYTE	240,224, 0,112, 0, 0,112, 0, 0,112, 0, 0
E0D7	70	00 00	2780	.BYTE	112, 0, 0,240, 0, 15,224, 0, 31,192, 0, 15
E0E3	E0	00 00	2790	.BYTE	224, 0, 0,240, 0, 0,112, 0, 0,112, 0, 0
E0EF	70	E0 00	2800	.BYTE	112,224, 0,112,240, 0,240,127,255,224, 63,255
E0FB	C0	0F FF	2810	.BYTE	192, 15,255, 0, 0, 0, 3,192, 0, 7,192, 0
E107	0F	C0 00	2820	.BYTE	15,192, 0, 31,192, 0, 61,192, 0,121,192, 0
E113	F1	C0 01	2830	.BYTE	241,192, 1,225,192, 3,193,192, 7,129,192, 15
E11F	01	C0 1E	2840	.BYTE	1,192, 30, 1,192, 60, 1,192,127,255,240,255
E12B	FF	F0 FF	2850	.BYTE	255,240,255,255,240, 0, 1,192, 0, 1,192, 0
E137	01	C0 00	2860	.BYTE	1,192, 0, 1,192, 0, 1,192,175,255,255,224
E143	FF	FF E0	2870	.BYTE	255,255,224,255,255,224,224, 0, 0,224, 0, 0
E14F	E0	00 00	2880	.BYTE	224, 0, 0,224, 0, 0,224, 0, 0,239,255, 0
E15B	FF	FF C0	2890	.BYTE	255,255,192,255,255,224,224, 0,240, 0, 0,112
E167	00	00 70	2900	.BYTE	0, 0,112, 0, 0,112, 0, 0,112,224, 0,112
E173	F0	00 F0	2910	.BYTE	240, 0,240,127,255,224, 63,255,192, 15,255, 0
E17F	00	0F FF	2920	.BYTE	0, 15,255, 0, 63,255,192,127,255,224,240, 0
E18B	F0	E0 00	2930	.BYTE	240,224, 0,112,224, 0, 0,224, 0, 0,224, 0
E197	00	E0 00	2940	.BYTE	0,224, 0, 0,239,255, 0,255,255,192,255,255
E1A3	E0	F0 00	2950	.BYTE	224,240, 0,240,224, 0,112,224, 0,112,224, 0
E1AF	70	E0 00	2960	.BYTE	112,224, 0,112,240, 0,240,127,255,224, 63,255
E1BB	C0	0F FF	2970	.BYTE	192, 15,255, 0, 69,255,255,240,255,255,240,255
E1C7	FF	F0 C0	2980	.BYTE	255,240,192, 0,240,192, 0,224, 0, 1,224, 0
E1D3	01	C0 00	2990	.BYTE	1,192, 0, 3,192, 0, 3,128, 0, 7,128, 0
E1DF	07	00 00	3000	.BYTE	7, 0, 0, 15, 0, 0, 14, 0, 0, 30, 0, 0
E1EB	1C	00 00	3010	.BYTE	28, 0, 0, 60, 0, 0, 56, 0, 0,120, 0, 0
E1F7	70	00 00	3020	.BYTE	112, 0, 0,240, 0, 0,224, 0, 0, 15,255, 0
E203	3F	FF C0	3030	.BYTE	63,255,192,127,255,224,240, 0,240,224, 0,112
E20F	E0	00 70	3040	.BYTE	224, 0,112,224, 0,112,224, 0,112,240, 0,240
E21B	7F	FF E0	3050	.BYTE	127,255,224, 63,255,192,127,255,224,240, 0,240
E227	E0	00 70	3060	.BYTE	224, 0,112,224, 0,112,224, 0,112,224, 0,112
E233	F0	00 F0	3070	.BYTE	240, 0,240,127,255,224, 63,255,192, 15,255, 0
E23F	45	0F FF	3080	.BYTE	69, 15,255, 0, 63,255,192,127,255,224,240, 0

```

E24B F0 E0 00 3090 .BYTE240,224, 0,112,224, 0,112,224, 0,112,224, 0
E257 70 F0 00 3100 .BYTE112,240, 0,240,127,255,240, 63,255,240, 15,255
E263 70 00 00 3110 .BYTE112, 0, 0,112, 0, 0,112, 0, 0,112, 0, 0
E26F 70 E0 00 3120 .BYTE112,224, 0,112,240, 0,240,127,255,224, 63,255
E27B C0 0F FF 3130 .BYTE192, 15,255, 0,141, 0, 0, 0, 0, 0, 0, 0, 0
E287 00 00 00 3140 .BYTE 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
E293 00 00 00 3150 .BYTE 0, 0, 0,240, 0, 0,240, 0, 0,240, 0, 0
E29F 00 00 00 3160 .BYTE 0, 0, 0, 0, 0, 0, 0,240, 0, 0,240, 0, 0
E2AB F0 00 00 3170 .BYTE240, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
E2B7 00 00 00 3180 .BYTE 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
E2C0 3190 .END

```

ZEILEN:310 SYMBOLE:39 FEHLER:0

A1	=C862	A2	=C8D4	A3	=C8C2	A4	=C8A4	A5	=C8FB	A6	=C8D6
A7	=C8E6	A8	=C91A	A9	=C927	AA0	=C872	AA1	=C821	AA2	=C830
B0	=C92B	B1	=C943	B2	=C954	B3	=C950	B4	=C99C	B5	=C9A7
B6	=C9B8	CHKCOM=AEFD	CHRGOT=0079	CRA2	=DD0E	END	=C871	FRESTR=B6A3			
FRMEVL=AD9E	HMS	=C875	ICR1	=DC0D	IRMASK=D01A	IRR	=D019	PFR1	=C9C1		
PFR2	=C9F2	PRA2	=DD00	RASTER=D012	TIME	=C894	TODHRS=DD0B	TODMIN=DD0A			
TODSEC=DD09	TODTHS=DD08	VIC	=D000								

```

100 FORI= 51200 TO 51697 :READ X:POKE I,X:S=S+X:NEXT
110 DATA 32,121, 0,240,108, 32,253,174, 32,158,173, 32
120 DATA 163,182,201, 6,208, 96,160,255,162, 3, 32,117
130 DATA 200, 5, 2,208, 4,169,146,208, 15,201, 36,176
140 DATA 77,201, 19,144, 7, 56,248,233, 18,216, 9,128
150 DATA 141, 11,221,162, 6, 32,117,200,141, 10,221, 32
160 DATA 117,200,141, 9,221,169, 0,141, 8,221,120,169
170 DATA 127,141, 13,220,169,148,141, 20, 3,169,200,141
180 DATA 21, 3,169, 1,141, 26,208,169,247,141, 18,208
190 DATA 162, 47,189, 0,208,157,242,201,202, 16,247,169
200 DATA 0,141, 8,221, 88, 96, 76, 72,178,134,255,200
210 DATA 177, 34, 56,233, 48,197,255,176,241, 10, 10, 10
220 DATA 10,133, 2,200,177, 34, 56,233, 48,201, 10,176
230 DATA 225, 5, 2, 96,120,173, 25,208,141, 25,208,173
240 DATA 18,208,201,247,208, 50,162, 24,189, 0,208,157
250 DATA 242,201,189,193,201,157, 0,208,202, 16,241,173
260 DATA 0,221, 41,252,141, 0,221,169, 0,141, 18,208
270 DATA 162, 19,189, 27,208,157, 13,202,189,220,201,157
280 DATA 27,208,202, 16,241, 76,251,200,162, 24,189,242
290 DATA 201,157, 0,208,202, 16,247,169,247,141, 18,208
300 DATA 162, 19,189,220,201,157, 27,208,202, 16,247,173
310 DATA 0,221, 9, 3,141, 0,221, 88, 76, 49,234,160
320 DATA 0,173, 11,221, 41, 16, 24, 74, 74, 74, 74,105
330 DATA 128, 72,173, 11,221,201, 18,208, 9,169,128,141
340 DATA 248,199,104, 76, 43,201,201,146,208, 9,169,129
350 DATA 141,248,199,104, 76, 43,201,104,141,248,199,173
360 DATA 11,221, 41, 15,105,128, 72,173, 11,221,201, 18
370 DATA 208, 9,169,128,141,249,199,104, 76, 84,201,201
380 DATA 146,208, 9,169,130,141,249,199,104, 76, 84,201
390 DATA 104, 32,156,201,169,138,141,250,199,173, 10,221
400 DATA 41,240, 74, 74, 74, 74,105,128,141,251,199,173
410 DATA 10,221, 41, 15,105,128,141,252,199,169,138,141
420 DATA 253,199,173, 9,221, 41,240, 74, 74, 74, 74,105
430 DATA 128,141,254,199,173, 9,221, 41, 15,105,128,141
440 DATA 255,199,173, 14,221, 9,128,141, 14,221,173, 8
450 DATA 221, 76,129,234,141,249,199,173, 11,221,233,128
460 DATA 16, 1, 96,238,248,199,238,249,199,238,249,199
470 DATA 173,249,199,233,138, 16, 1, 96,105,127,141,249
480 DATA 199,238,248,199, 96, 87, 11,112, 11,137, 11,162
490 DATA 11,187, 11,212, 11,237, 11, 7, 11,128, 23, 89
500 DATA 154, 54,255,200, 0, 21,121,240, 0, 0, 0, 0
510 DATA 0,240,240,241,242,243,240,240, 15, 15, 15, 15
520 DATA 15, 15, 15, 15, 2,255
530 IF S <> 64006 THEN PRINT"HIBA A DATASORBAN !":END
540 S=0:FORI=0 TO 702:READ X:POKE 57344+I,X:S=S+X:NEXT I
550 DATA 7,254, 0, 31,255,128, 63,255,192,120, 1
560 DATA 224,112, 0,224,112, 0,224,112, 0,224,112, 0
570 DATA 224,112, 0,224,112, 0,224,112, 0,224,112, 0

```



```

580 DATA 224,112, 0,224,112, 0,224,112, 0,224,112, 0
590 DATA 224,112, 0,224,120, 1,224, 63,255,192, 31,255
600 DATA 128, 7,254, 0, 0, 0, 7,128, 0, 15,128, 0
610 DATA 31,128, 0, 63,128, 0,123,128, 0,243,128, 1
620 DATA 227,128, 3,195,128, 0, 3,128, 0, 3,128, 0
630 DATA 3,128, 0, 3,128, 0, 3,128, 0, 3,128, 0
640 DATA 3,128, 0, 3,128, 0, 3,128, 0, 3,128, 0
650 DATA 3,128, 0, 3,128, 0, 3,128, 0, 15,255, 0
660 DATA 63,255,192,127,255,224,240, 0,240,224, 0,112
670 DATA 224, 0,112, 0, 0,112, 0, 0,240, 0, 1,224
680 DATA 0, 7,192, 0, 31, 0, 0,124, 0, 1,240, 0
690 DATA 7,192, 0, 31, 0, 0,124, 0, 0,240, 0, 48
700 DATA 240, 0, 48,255,255,240,255,255,240,255,255,240
710 DATA 0, 15,255, 0, 63,255,192,127,255,224,240, 0
720 DATA 240,224, 0,112, 0, 0,112, 0, 0,112, 0, 0
730 DATA 112, 0, 0,240, 0, 15,224, 0, 31,192, 0, 15
740 DATA 224, 0, 0,240, 0, 0,112, 0, 0,112, 0, 0
750 DATA 112,224, 0,112,240, 0,240,127,255,224, 63,255
760 DATA 192, 15,255, 0, 0, 0, 3,192, 0, 7,192, 0
770 DATA 15,192, 0, 31,192, 0, 61,192, 0,121,192, 0
780 DATA 241,192, 1,225,192, 3,193,192, 7,129,192, 15
790 DATA 1,192, 30, 1,192, 60, 1,192,127,255,240,255
800 DATA 255,240,255,255,240, 0, 1,192, 0, 1,192, 0
810 DATA 1,192, 0, 1,192, 0, 1,192,175,255,255,224
820 DATA 255,255,224,255,255,224,224, 0, 0,224, 0, 0
830 DATA 224, 0, 0,224, 0, 0,224, 0, 0,239,255, 0
840 DATA 255,255,192,255,255,224,224, 0,240, 0, 0,112
850 DATA 0, 0,112, 0, 0,112, 0, 0,112,224, 0,112
860 DATA 240, 0,240,127,255,224, 63,255,192, 15,255, 0
870 DATA 0, 15,255, 0, 63,255,192,127,255,224,240, 0
880 DATA 240,224, 0,112,224, 0, 0,224, 0, 0,224, 0
890 DATA 0,224, 0, 0,239,255, 0,255,255,192,255,255
900 DATA 224,240, 0,240,224, 0,112,224, 0,112,224, 0
910 DATA 112,224, 0,112,240, 0,240,127,255,224, 63,255
920 DATA 192, 15,255, 0, 69,255,255,240,255,255,240,255
930 DATA 255,240,192, 0,240,192, 0,224, 0, 1,224, 0
940 DATA 1,192, 0, 3,192, 0, 3,128, 0, 7,128, 0
950 DATA 7, 0, 0, 15, 0, 0, 14, 0, 0, 30, 0, 0
960 DATA 28, 0, 0, 60, 0, 0, 56, 0, 0,120, 0, 0
970 DATA 112, 0, 0,240, 0, 0,224, 0, 0, 15,255, 0
980 DATA 63,255,192,127,255,224,240, 0,240,224, 0,112
990 DATA 224, 0,112,224, 0,112,224, 0,112,240, 0,240
1000 DATA 127,255,224, 63,255,192,127,255,224,240, 0,240
1010 DATA 224, 0,112,224, 0,112,224, 0,112,224, 0,112
1020 DATA 240, 0,240,127,255,224, 63,255,192, 15,255, 0
1030 DATA 69, 15,255, 0, 63,255,192,127,255,224,240, 0
1040 DATA 240,224, 0,112,224, 0,112,224, 0,112,224, 0
1050 DATA 112,240, 0,240,127,255,240, 63,255,240, 15,255
1060 DATA 112, 0, 0,112, 0, 0,112, 0, 0,112, 0, 0
1070 DATA 112,224, 0,112,240, 0,240,127,255,224, 63,255
1080 DATA 192, 15,255, 0,141, 0, 0, 0, 0, 0, 0, 0
1090 DATA 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
1100 DATA 0, 0, 0,240, 0, 0,240, 0, 0,240, 0, 0
1110 DATA 0, 0, 0, 0, 0, 0,240, 0, 0,240, 0, 0
1120 DATA 240, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
1130 DATA 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
1140 IF S<>71803 THEN PRINT "HIBA A DATASORBAN !":END
1150 END

```

A VIC igazi szakértői a játékprogramok szerzői. Az említett trükkökön kívül számos speciális effektus valósítható meg. Ezekből mutatunk be kettőt. Az egyik azt a lehetőséget használja ki, hogy a képernyő pontonként, 8 ponttal mozgatható el. A másik, szintén igen egyszerű ötlet a karaktergenerátor mozgatható voltából fakadó lehetőségeket használja fel.

A hullámszó képernyőt előállító példaprogramunk olyan illúziót kelt a monitoron, amelyet a hajdani „jobb” televíziók minden trükk nélkül képesek voltak produkálni. A program a BASIC és a gépi nyelv „szimbiózisa”.

A trükk elve lényegében a következő: a VIC megengedi a képernyő háttérének képpontkénti eltolását. Az eltolás mértékét a VIC 2. ellenőrző-vezérlő regiszterének (\$D016) alsó három bitje szabályozza. Mint tudjuk, a monitor elektronsugara kb. 20 ms ciklusidővel ér vissza ugyanabba a képernyősorba. A C64-es gépi programjainak sebessége lehetővé teszi, hogy megváltoztassuk az eltolás mértékét még az előtt, hogy az elektronsugár új képkocka felépítését kezdené meg. Sőt, olyan gyors is lehet, hogy az elektronsugár még a rastersort sem képes elhagyni, mialatt a program megváltoztatja az eltolás mértékét.

A BASIC program egy kezdőértéket kér; az itt megadott szám nagysága befolyásolja a gépi cikluson belüli időzítést, de fordított módon. A nagyobb szám rövidebb futást eredményez, és szaporább hullámozást. A BASIC program felépít egy keretet, amelyen a hullámozó képernyő szemléletesen látható. A keret alatt egyfajta menü található. Az F1 billentyűre növeli eggyel a paramétert, ha az kisebb 255-nél. Az F3 billentyűre eggyel csökkenti a paramétert, ha az nagyobb, mint 0. Az F5 billentyű újraindítja a programot, ezáltal a paramétert újra megadhatjuk. Az F7 billentyű hatására a program véget ér. A gépi kódú program működéséből következik, hogy a BASIC nem reagál azonnal a lenyomott billentyűkre.

A paraméter értéke 0 és 255 közé eshet. Szemléletes effektust kaphatunk több paraméterrel, pl. 113, 162, 189, 212, 223, 236, 239, 245 stb.

```

100 POKE 53280,0:POKE 53281,0:PRINT CHR$(30)
110 GOSUB 1000
120 PRINT CHR$(147):INPUT " KEZDOERTEK:";A
130 IF A>255 OR A<0 THEN 120
140 PRINT CHR$(147):PRINT " |-----|"
150 FOR I=1 TO 20:PRINT " |         |         |         |         |" :NEXT
160 PRINT " |-----|"
170 PRINT "      F1: +   F3: -   F5: NEW   F7: END"
180 PRINT CHR$(13):PRINT " PARAMETER : ";A
190 POKE 10035,A AND 255
200 SYS 10000
210 GET X$:IF X$="" THEN 210
220 IF X$=CHR$(133) THEN IF A<255 THEN A=A+1
230 IF X$=CHR$(134) THEN IF A>0 THEN A=A-1
240 IF X$=CHR$(135) THEN 120
250 IF X$=CHR$(136) THEN POKE 53270,200:PRINT CHR$(147):END
260 GOTO 180
1000 FOR I= 10000 TO 10038 :READ X:POKE I,X:S=S+X:NEXT
1010 DATA 162, 0,160, 0,140, 22,208, 32, 46, 39,200,132
1020 DATA 8,208,245,160, 7,140, 22,208, 32, 46, 39,136
1030 DATA 16,247,232,208,229, 96,230, 2,208,252,163,255
1040 DATA 133, 2, 96
1050 IF S <> 4827 THEN PRINT "HIBA A DATASORBAN !":END
1060 RETURN
READY.

```

A *karaktergenerátor módosítását* a játékprogramok alkotói gyakran alkalmazzák; azaz háttérként a nagyfelbontású grafika helyett módosított bitmintájú karaktereket szerepeltetnek (rendszerint többszínű üzemmódban). Az itt bemutatott program ugyan egyszínű üzemmóddal dolgozik, de a megoldás elvét jól szemlélteti. Jó példa arra is, hogyan lehet csekély programozási munkával látványos eredményre jutni.


```

2710          100      *=10000      ; belépési pont

2710 A2 00      110 SR0      LDX #00      ; kezdőértékek
2712 A0 00      120 AA2      LDY #00
2714 8C 16 D0   130 AA0      STY $D016    ; a VIC 2. ellenőrző-vezérlő regisztere
2717 20 2E 27   140          JSR SR1      ; kitartóciklus
271A C3         150          INY          ; növelni az eltolást
271B C0 08      160          CPY #08      ; az eltolás értéke elérte a 8-at ?
271D D0 F5      170          BNE AA0      ; még nem, vissza
271F A0 07      180          LDY #07      ; lefelé számlálás - a hullám alsó fele
2721 8C 16 D0   190 AA1      STY $D016    ; a pillanatnyi érték a regiszterbe
2724 20 2E 27   200          JSR SR1      ; kitartóciklus
2727 83         210          DEY          ; csökkenteni az eltolás értékét
2728 10 F7      220          BPL AA1      ; vissza
272A E3         230          INX          ; a külső ciklus változójának növelése
272B D0 E5      240          BNE AA2      ; még nem futott le, vissza
272D 60         250          RTS

272E E6 02      260 SR1      INC $02      ; a számláló növelése
2730 D0 FC      270          BNE SR1
2732 A9 FF      280 PRM      LDA #$FF      ; új kezdőérték betöltése
2734 85 02      290          STA $02      ; a számlálóba
2736 60         300          RTS
2737          310          .END

```

ZEILEN:22 SYMBOLE:6 FEHLER:0

AA0 =2714 AA1 =2721 AA2 =2712 PRM =2732 SR0 =2710 SR1 =272E

A teljes képernyő mozgásban levő hátteret mutat, pedig valójában csak a módosított karaktergenerátor egy elemét változtatjuk folyamatosan a megszakítási rutin alatt. A gépi program először megváltoztatja a karaktergenerátor címezését, majd lemásolja azt a RAM-ba, a \$3000 címtől kezdődően. Ezt követően megváltoztatja a karaktergenerátor első elemét, amely eredetileg a @ karakter bitmintáját tárolta. Ezután a képernyőtárt és a színtárt tölti fel a megfelelő byte-okkal. Be kell még állítani a megszakítórutin új kezdőcímét, a megfelelő összehatáshoz módosítani kell a megszakítások gyakoriságát.

A tulajdonképpeni trükköt a megszakítási rutin végzi, mert annak keretében változik meg a kiválasztott karakter bitmintája. A kis szubrutin lényegében ciklikusan forgatja a karakter kontraszterének sorait. A program nem használ paramétereket. Indítása a SYS 10000 utasítással, megállítása a STOP és RESTORE billentyűk együttes lenyomásával történhet. A program tulajdonképpen nem ér véget, mert végtelen ciklusba fut.

```

2710          100      *=10000      ; programkezdet
2710 A9 00      110          LDA #00      ; a szín fekete
2712 8D 20 D0   120          STA $D020    ; a keret
2715 8D 21 D0   130          STA $D021    ; a háttér
2718 AD 18 D0   140          LDA $D018    ; a karaktergenerátor
271B 29 F0      150          AND #$F0      ; a karaktergenerátor
271D 09 0C      160          ORA #0C      ; címezésének megváltoztatása
271F 8D 18 D0   170          STA $D018
2722 A5 01      180          LDA $01
2724 48         190          PHA
2725 29 FB      200          AND #$FB      ; hozzáférés a CHAREN-hez

```

2727	85	01		210		STA	\$01	
2729	A2	00		220		LDX	#00	
272B	8D	00	D0	230	AA3	LDA	\$D000,X	; a CHAREN
272E	9D	00	30	240		STA	\$3000,X	
2731	8D	00	D1	250		LDA	\$D100,X	; lemásolása a RAM-ba
2734	9D	00	31	260		STA	\$3100,X	
2737	CA			270		DEX		
2738	D0	F1		280		BNE	AA3	
273A	68			290		PLA		
273B	35	01		300		STA	\$01	
273D	A2	07		310		LDX	#07	
273F	8D	AB	27	320	AA4	LDA	CHAR,X	; az első karakter
2742	9D	00	30	330		STA	\$3000,X	; bitmintájának módosítása
2745	CA			340		DEX		
2746	10	F7		350		BPL	AA4	
2748	A2	00		360		LDX	#00	
274A	A9	00		370	AA5	LDA	#00	
274C	9D	00	04	380		STA	\$0400,X	; a videoram feltöltése
274F	9D	00	05	390		STA	\$0500,X	
2752	9D	00	06	400		STA	\$0600,X	; a módosított karakterrel
2755	9D	00	07	410		STA	\$0700,X	
2758	A9	0F		420		LDA	#0F	
275A	9D	00	08	430		STA	\$D800,X	; a színram
275D	9D	00	09	440		STA	\$D900,X	
2760	9D	00	0A	450		STA	\$DA00,X	; feltöltése
2763	9D	00	0B	460		STA	\$DB00,X	
2766	CA			470		DEX		
2767	D0	E1		480		BNE	AA5	
2769	78			490		SEI		; a megszakítások tiltása
276A	A9	4C		500		LDA	#4C	; a megszakítások
276C	8D	05	DC	510		STA	\$DC05	
276F	A9	C7		520		LDA	#C7	; gyakoriságának megváltoztatása
2771	8D	04	DC	530		STA	\$DC04	
2774	A9	82		540		LDA	#NIRQ<	; az új IRQ rutin címe
2776	8D	14	03	550		STA	\$0314	; a vektor alsó byte-ja
2779	A9	27		560		LDA	#NIRQ>	; az új IRQ rutin címe
277B	8D	15	03	570		STA	\$0315	; a vektor felső byte-ja
277E	58			580		CLI		; a megszakítások engedélyezése
277F	4C	7F	27	590	AA6	JMP	AA6	; végtelen ciklus
2782	A2	07		600	NIRQ	LDX	#07	
2784	8D	00	30	610	AA0	LDA	\$3000,X	
2787	18			620		CLC		
2788	2A			630		ROL	A	; a karakter bitmintájának balratolása
2789	9D	02		640		BCC	AA1	
278B	09	01		650		ORA	#01	
278D	9D	00	30	660	AA1	STA	\$3000,X	
2790	CA			670		DEX		
2791	10	F1		680		BPL	AA0	
2793	A2	07		690		LDX	#07	
2795	8D	00	30	700		LDA	\$3000,X	
2798	48			710		PHA		; az első sor elmentése
2799	CA			720	AA2	DEX		
279A	8D	00	30	730		LDA	\$3000,X	; a bitminta sorainak mozgatása
279D	E8			740		INX		
279E	9D	00	30	750		STA	\$3000,X	
27A1	CA			760		DEX		
27A2	D0	F5		770		BNE	AA2	
27A4	68			780		PLA		; az elmentett sor
27A5	9D	00	30	790		STA	\$3000,X	; most az utolsó
27A8	4C	31	EA	800		JMP	\$EA31	; az eredeti IRQ rutinra
27AB	03	03	03	810	CHAR	,BYTE	3,3,3,3,3,3,3,255	
27B3				820		,END		

ZEILEN:73 SYMBOLE:9 FEHLER:0

AA0	=2784	AA1	=278D	AA2	=2799	AA3	=272B	AA4	=273F	AA5	=274A
AA6	=277F	CHAR	=27AB	NIRQ	=2782						


```

100 FOR I= 10000 TO 10162 :READ X:POKE I,X:S=S+X:NEXT
110 DATA 169, 0,141, 32,208,141, 33,208,173, 24,208, 41
120 DATA 240, 9, 12,141, 24,208,165, 1, 72, 41,251,133
130 DATA 1,162, 0,189, 0,208,157, 0, 48,189, 0,209
140 DATA 157, 0, 49,202,208,241,104,133, 1,162, 7,189
150 DATA 171, 39,157, 0, 48,202, 16,247,162, 0,169, 0
160 DATA 157, 0, 4,157, 0, 5,157, 0, 6,157, 0, 7
170 DATA 169, 15,157, 0,216,157, 0,217,157, 0,218,157
180 DATA 0,219,202,208,225,120,169, 76,141, 5,220,169
190 DATA 199,141, 4,220,169,130,141, 20, 3,169, 39,141
200 DATA 21, 3, 88, 76,127, 39,162, 7,189, 0, 48, 24
210 DATA 42,144, 2, 9, 1,157, 0, 48,202, 16,241,162
220 DATA 7,189, 0, 48, 72,202,189, 0, 48,232,157, 0
230 DATA 48,202,208,245,104,157, 0, 48, 76, 49,234, 3
240 DATA 3, 3, 3, 3, 3, 3, 3,255
250 IF S <> 16344 THEN PRINT"HIBA A DATASORBAN !":END
260 PRINT"OK !!":END

```

Nézzünk most egy példát a *fényceruza kezelésére*! A fényceruza y koordinátáját közvetlenül is kiolvashatjuk a VIC megfelelő regiszteréből, de az x koordináta olyan szóródást mutat, hogy ha pontosabb értéket akarunk a fényceruza pillanatnyi x koordinátájáról, akkor legálábbis átlagot kell számolni az egy időintervallum alatt kapott adatokból. Az alábbi program megadja a fényceruza aktuális koordinátáit, a gépi program pedig elvégzi az imént említett átlagolást is (a BASIC igen lassúnak bizonyulna ebben az esetben). Az x koordináta értékét a megszakítási rutinba ágyazott program folyamatosan aktualizálja.

```

10 IF PEEK(10000)<>120 OR PEEK(10031)<>153 THEN GOSUB200
100 POKE53280,11:POKE53281,12:PRINTCHR$(155)
110 SYS10000
120 PRINTCHR$(147)PEEK(10070),PEEK(53268)
130 GOTO120
200 FOR I= 10000 TO 10069 :READ X:POKE I,X:S=S+X:NEXT
210 DATA 120,169, 29,141, 20, 3,169, 39,141, 21, 3, 88
220 DATA 96,173, 25,208,141, 25,208, 41, 8,208, 3, 76
230 DATA 49,234,160, 0,173, 19,208,153, 86, 39,200,208
240 DATA 247,169,127,133, 2, 24,162, 0,160, 0,185, 86
250 DATA 39,200,121, 86, 39,106,157, 86, 39,200,232,228
260 DATA 2,208,239, 70, 2,208,231, 76,129,234
270 IF S <> 7911 THEN PRINT"HIBA A DATASORBAN !":END
280 RETURN
READY.

```

2710	100	*=10000	
2710 78	110	SEI	; az IRQ letiltása
2711 A9 10	120	LDA #NIRQ<	; a megszakítási vektor
2713 8D 14 03	130	STA \$0314	
2716 A9 27	140	LDA #NIRQ>	; módosítása
2718 8D 15 03	150	STA \$0315	
271B 58	160	CLI	; az IRQ engedélyezése
271C 60	170	RTS	
271D AD 19 D0	180 NIRQ	LDA \$D019	; VIC IRR
2720 8D 19 D0	190	STA \$D019	; visszaírva törlődik
2723 29 08	200	AND #\$08	; %00001000, fényceruza generálta ?
2725 D0 03	210	BNE AA4	; igen, ugrás
2727 4C 31 EA	220	JMP \$EA31	; az eredeti IRQ
272A A0 00	230 AA4	LDY #\$00	
272C AD 13 D0	240 AA3	LDA \$D013	; fényceruza X koordináta
272F 99 56 27	250	STA BUF,Y	; a puffer feltöltése
2732 C8	260	INY	
2733 D0 F7	270	BNE AA3	
2735 A9 7F	280	LDA #\$7F	; átlagot képezni 7-szer (%01111111)

```

2737 85 02      290      STA $02      ; tárolni a ciklusszámlálót
2739 18        300      CLC
273A A2 00      310 AA2    LDX #$00
273C A0 00      320      LDY #$00
273E 89 56 27   330 AA0    LDA BUF,Y    ; a minta beolvasása
2741 C8        340      INY
2742 79 56 27   350      ADC BUF,Y    ; a következő minta hozzáadása
2745 6A        360      ROR A        ; és osztás kettővel
2746 9D 56 27   370      STA BUF,X    ; és vissza az első minta helyére
2749 C8        380      INY
274A E8        390      INX        ; a mutatók növelése
274B E4 02      400      CPX $02      ; az aktuális ciklusnak vége ?
274D 00 EF      410      BNE AA0      ; még nem, ugrás
274F 46 02      420      LSR $02      ; ciklusmutató csökkentése, lefutott ?
2751 00 E7      430      BNE AA2      ; még nem, ugrás
2753 4C 81 EA    440      JMP $EA81    ; befejezni az IRQ-t
2756          450 BUF    =*          ; az eredmény a puffer első eleme lesz
2756          460      .END

```

ZEILEN:37 SYMBOLE:6 FEHLER:0

AA0 =273E AA2 =273A AA3 =272C AA4 =272A BUF =2756 NIRQ =271D

Végül lássunk egy egyszerű *grafikai segédletet*, amely a fontosabb grafikai parancsokat végrehajtja, és emellett rugalmasan kezelhető. Beépíthető és felhasználható saját programjainkhoz, legyenek azok akár BASIC, akár gépi nyelvű programok.

A program törli a HIRES képernyőt, megadja annak színét, a pontot bekapcsolja, törli, invertálja. A gépi rutinok úgy lettek kialakítva, hogy BASIC programból legyenek hívhatók, de a paraméter átvételének programrészeit módosítva akár gépi programok is hívhatják őket.

A közölt BASIC program bemutatja a felhasználás néhány lehetőségét. A program gépi része lényegében szubrutinyűjtemény, ezek a szubrutinok a következő feladatokat végezhetik:

- | | | |
|---------------------------|--------|--------------------|
| 1. inicializálás | \$C000 | SYS 49152 |
| 2. a grafika bekapcsolása | \$C003 | SYS 49155 |
| 3. a szín beállítása | \$C00F | SYS 49167, színkód |
| 4. a pont bekapcsolása | \$C012 | SYS 49170, X, Y |
| 5. a pont törlése | \$C018 | SYS 49176, X, Y |
| 6. a pont invertálása | \$C015 | SYS 49173, X, Y |
| 7. a képernyő invertálása | \$C00C | SYS 49164 |
| 8. a képernyő törlése | \$C009 | SYS 49161 |
| 9. a grafika kikapcsolása | \$C006 | SYS 49158 |

```

100 IF PEEK(49152)<>76 OR PEEK(49437)<>35 THEN GOSUB 500
110 POKE 53280,2:POKE 53281,2
120 SYS 49152      :REM INICIALIZALAS
130 SYS 49155      :REM GRAFIKA BE
140 SYS 49167,2    :REM SZIN VOROS
150 A=49170        :REM SET
160 GOSUB 320
170 A=49176        :REM RESET
180 GOSUB 320
190 A=49173        :REM INV
200 GOSUB 320
210 SYS 49167,10   :REM ROZSASZ IN
220 POKE 53280,10

```



```

SUB 300
S 49164          :REM INVERZ
KE 53280,0
SUB 300
S 49158          :REM GRAFIKA KI
ID

```

```
IT193,1:GETX#:RETURN
```

```

R I=0 TO 319
S A,I,0
S A,319-I,199
XT I
R I=0 TO 99
S A,319-INT(I*1.6+.5),I
S A,INT(I*1.6+.5),199-I
XT I
R I=99 TO 0 STEP-1
S A,319-INT(I*1.6+.5),199-I
S A,INT(I*1.6+.5),I
XT I
R I=0 TO 199
S A,0,I
S A,319,199-I
XT I
TURN

```

```

RI= 49152 TO 49553 :READ X:POKE I,X:S=S+X:NEXT
TA 76, 27,192, 76, 61,192, 76, 85,192, 76,106,192
TA 76,129,192, 76,160,192, 76,186,192, 76,199,192
TA 76,192,192,173, 17,208,141,140,193,173, 24,208
TA 141,141,193,173, 0,221,141,142,193,173, 22,208
TA 41,239,141, 22,208, 32,106,192,162, 16, 76,163
TA 192,173, 2,221, 9, 3,141, 2,221,169,196,141
TA 0,221,169, 56,141, 24,208,169, 59,141, 17,208
TA 96,173,142,193,141, 0,221,173,141,193,141, 24
TA 208,173,140,193,141, 17,208, 76, 68,229,160, 0
TA 169,224,132, 34,133, 35,152,145, 34,200,208,251
TA 230, 35,165, 35,201, 0,208,242, 96,169, 0,169
TA 224,192, 34,133, 35, 32, 74,193,177, 34, 73,255
TA 145, 34,200,208,247,230, 35,165, 35,201, 0,208
TA 239, 76, 82,193, 32,241,183,160, 0,169,204,132
TA 34,133, 35,138,145, 34,200,208,251,230, 35,165
TA 35,201,208,208,242, 96,169, 0, 44,169, 64, 44
TA 169,129,133, 2, 32,253,174, 32,235,183,224,200
TA 144, 6, 32, 85,192, 76, 72,178,165, 21,201, 1
TA 144, 8,208,242,165, 20,201, 64,176,236, 32, 74
TA 193,138, 74, 74, 74,168,185,115,193,141,143,193
TA 185, 90,193,141,144,193,138, 41, 7, 24,109,143
TA 193,141,143,193,165, 20, 41,248,141,145,193, 24
TA 173,143,193,133, 34,169,224,109,144,193,133, 35
TA 165, 34, 24,109,145,193,133, 34,165, 35,101, 21
TA 133, 35,165, 20, 41, 7, 73, 7,170,169, 1,202
TA 48, 3, 10,208,250,160, 0, 36, 2, 80, 5, 81
TA 34, 76, 69,193, 16, 5, 73,255, 49, 34, 44, 17
TA 34,145, 34, 76, 32,193,120,165, 1, 41,249,133
TA 1, 96,165, 1, 3, 5,133, 1, 88, 96, 0, 1
TA 2, 3, 5, 6, 7, 8, 10, 11, 12, 13, 15, 16
TA 17, 18, 20, 21, 22, 23, 25, 26, 27, 28, 30, 0
TA 64,128,192, 0, 64,128,192, 0, 64,128,192, 0
TA 64,128,192, 0, 64,128,192, 0, 64,129,192, 0
TA 0, 0, 0, 0, 0, 0
S (> 44671 THEN PRINT"HIBA A DATASORBAN !":END
TURN

```

C000				100		*= \$C000	
C000	4C	1B	C0	110		JMP INIT	; alapbeállítások
C003	4C	3D	C0	120		JMP HIRES	; HIRES mód bekapcsolása
C006	4C	55	C0	130		JMP CHAR	; vissza Karakteres módba
C009	4C	6A	C0	140		JMP CLEAR	; a bittérkép törlése
C00C	4C	81	C0	150		JMP INVERZ	; a bittérkép invertálása
C00F	4C	A0	C0	160		JMP COLOR	; a szín beállítása
C012	4C	BA	C0	170		JMP SET	; pont bekapcsolása
C015	4C	BD	C0	180		JMP INV	; pont invertálása
C018	4C	C0	C0	190		JMP RESET	; pont törlése
C01B	AD	11	D0	200	INIT	LDA \$D011	; VIC ellenőrző-vezérlő regiszter #1
C01E	8D	8C	C1	210		STA PUF1	; átmeneti tárolása
C021	AD	18	D0	220		LDA \$D018	; a CHAREN és a videoram báziscíme
C024	8D	8D	C1	230		STA PUF2	; átmeneti tárolása
C027	AD	00	D0	240		LDA \$D000	; a CHAREN báziscíme, 14-15. bit
C02A	8D	8E	C1	250		STA PUF3	; átmeneti tárolása
C02D	AD	16	D0	260		LDA \$D016	; VIC ellenőrző-vezérlő regiszter #2
C030	29	EF		270		AND #\$EF	; %11101111, a 4. bit törlése, HIRES
C032	8D	16	D0	280		STA \$D016	
C035	20	6A	C0	290		JSR CLEAR	; a HIRES tár törlése
C038	A2	10		300		LDX #\$10	; a szín fekete
C03A	4C	A3	C0	310		JMP COL1	; a szín beállítása
C03D	AD	02	D0	320	HIRES	LDA \$D002	; CIA2 DDRA regiszter
C040	03	03		330		ORA #\$03	; a 0. és az 1. bit beállítása, kimene
C042	8D	02	D0	340		STA \$D002	
C045	A9	C4		350		LDA #\$C4	; %11000100
C047	8D	00	D0	360		STA \$D000	; a 4. 16K-s lap címezhető
C04A	A9	38		370		LDA #\$38	; %00111000, a videoram offsete 3072
C04C	8D	18	D0	380		STA \$D018	; a CHAREN offsete 8192
C04F	A9	3B		390		LDA #\$3B	; %00111011
C051	8D	11	D0	400		STA \$D011	; VIC ellenőrző-vezérlő regiszter #1
C054	60			410		RTS	
C055	AD	8E	C1	420	CHAR	LDA PUF3	
C058	8D	00	D0	430		STA \$D000	; CIA2, PRA
C05B	AD	8D	C1	440		LDA PUF2	
C05E	8D	18	D0	450		STA \$D018	; a videoram és a CHAREN báziscíme
C061	AD	8C	C1	460		LDA PUF1	
C064	8D	11	D0	470		STA \$D011	; VIC ellenőrző-vezérlő regiszter #1
C067	4C	44	E5	480		JMP \$E544	; képernyőtörlés
C06A	A0	00		490	CLEAR	LDY #\$00	; a HIRES-tár
C06C	A9	E0		500		LDA #\$E0	; Kezdőcíme, \$E000
C06E	84	22		510		STY \$22	
C070	85	23		520		STA \$23	
C072	98			530	AA0	TYA	
C073	91	22		540	AA1	STA (\$22),Y	; a byte törlése
C075	C8			550		INY	; a belső ciklus lefutott ?
C076	D0	FB		560		BNE AA1	; még nem, vissza
C078	E6	23		570		INC \$23	
C07A	A5	23		580		LDA \$23	
C07C	C9	00		590		CMP #\$00	; a külső ciklus lefutott ?
C07E	D0	F2		600		BNE AA0	; még nem, vissza
C080	60			610		RTS	
C081	A0	00		620	INVERZ	LDY #\$00	; a HIRES-tár
C083	A9	E0		630		LDA #\$E0	; Kezdőcíme, \$E000
C085	84	22		640		STY \$22	
C087	85	23		650		STA \$23	
C089	20	4A	C1	660		JSR ROMOFF	; a ROM kikapcsolása
C08C	B1	22		670	AA2	LDA (\$22),Y	; olvasás a ROM alól
C08E	49	FF		680		EOR #\$FF	; minden bit megfordítása
C090	91	22		690		STA (\$22),Y	; a byte visszairása
C092	C8			700		INY	; a belső ciklus lefutott ?
C093	D0	F7		710		BNE AA2	; még nem, vissza
C095	E6	23		720		INC \$23	
C097	A5	23		730		LDA \$23	
C099	C9	00		740		CMP #\$00	; a külső ciklus lefutott ?
C09B	D0	EF		750		BNE AA2	; még nem, vissza

C09D	4C	52	C1	760		JMP ROMON	; a ROM visszakapcsolása
C0A0	20	F1	B7	770	COLOR	JSR \$B7F1	; a vessző és a paraméter beolvasása
C0A3	A0	00		780	COL1	LDY #\$00	; a Karakteres videoram
C0A5	A9	CC		790		LDA #\$CC	; Kezdőcíme, \$CC00
C0A7	84	22		800		STY \$22	
C0A9	85	23		810		STA \$23	
C0AB	8A			820	AA3	TXA	; a színkód az X regiszterbe
C0AC	91	22		830	AA4	STA (\$22),Y	; a képernyő feltöltése
C0AE	C8			840		INY	; a belső ciklus lefutott ?
C0AF	D0	FB		850		BNE AA4	; még nem, vissza
C0B1	E6	23		860		INC \$23	
C0B3	A5	23		870		LDA \$23	
C0B5	C9	D0		880		CMP #\$00	; a külső ciklus lefutott ?
C0B7	D0	F2		890		BNE AA3	; még nem, vissza
C0B9	60			900		RTS	
C0BA	A9	00		910	SET	LDA #\$00	; a pont bekapcsolása
C0BC	2C			920		.BYTE \$2C	; rejtett LDA
C0BD	A9	40		930	INV	LDA #\$40	; a pont invertálása
C0BF	2C			940		.BYTE \$2C	; rejtett LDA
C0C0	A9	80		950	RESET	LDA #\$80	; a pont törlése
C0C2	85	02		960		STA \$02	; a módjelző tárolása
C0C4	20	FD	AE	970		JSR \$AEFD	; CHKCOM, vessző keresése
C0C7	20	EB	B7	980		JSR \$B7EB	; a paraméterek beolvasása
C0CA	E0	C8		990		CPX #\$C8	; nagyobb, mint 199 ?
C0CC	90	06		1000		BCC AA5	; nem, folytatni
C0CE	20	55	C0	1010	ERR	JSR CHAR	; visszaváltás karakteres módra
C0D1	4C	48	B2	1020		JMP \$B248	; ILLEGAL QUANTITY ERROR
C0D4	A5	15		1030	AA5	LDA \$15	; X koordináta felső byte
C0D6	C9	01		1040		CMP #\$01	; nagyobb, mint 1 ?
C0D8	90	08		1050		BCC AA6	; nem, folytatni
C0DA	D0	F2		1060		BNE ERR	; nem egyenlő és nagyobb, hiba
C0DC	A5	14		1070		LDA \$14	; X koordináta alsó byte
C0DE	C9	40		1080		CMP #\$40	; nagyobb, mint 63 ?
C0E0	80	EC		1090		BCS ERR	; igen, hiba
C0E2	20	4A	C1	1100	AA6	JSR ROMOFF	; a ROM kikapcsolása
C0E5	8A			1110		TXA	; Y koordináta
C0E6	4A			1120		LSR A	
C0E7	4A			1130		LSR A	; osztás 8-cal
C0E8	4A			1140		LSR A	
C0E9	A8			1150		TAY	
C0EA	B9	73	C1	1160		LDA TABL1,Y	
C0ED	8D	8F	C1	1170		STA PUF4	; cím offset, alsó byte
C0F0	B9	5A	C1	1180		LDA TABL2,Y	
C0F3	8D	90	C1	1190		STA PUF5	; cím offset, felső byte
C0F6	8A			1200		TXA	; Y koordináta
C0F7	29	07		1210		AND #\$07	; %00000111
C0F9	18			1220		CLC	
C0FA	6D	8F	C1	1230		ADC PUF4	
C0FD	8D	8F	C1	1240		STA PUF4	
C100	A5	14		1250		LDA \$14	; X koordináta alsó byte
C102	29	F8		1260		AND #\$F8	; %11111000
C104	8D	91	C1	1270		STA PUF6	; segédregiszter
C107	18			1280		CLC	
C108	AD	8F	C1	1290		LDA PUF4	
C10B	85	22		1300		STA \$22	
C10D	A9	E0		1310		LDA #\$E0	
C10F	6D	90	C1	1320		ADC PUF5	
C112	85	23		1330		STA \$23	
C114	A5	22		1340		LDA \$22	; a tényleges cím kiszámítása
C116	18			1350		CLC	
C117	6D	91	C1	1360		ADC PUF6	; a segédregiszter hozzáadása
C11A	85	22		1370		STA \$22	; az alsó byte kész
C11C	A5	23		1380		LDA \$23	
C11E	65	15		1390		ADC \$15	
C120	05	22		1400		STA \$22	; a felső byte kész

C122 A5 14	1410	LDA \$14	; X paraméter, alsó byte
C124 29 07	1420	AND #\$07	; a számláló
C126 49 07	1430	EOR #\$07	; Kiszámítása
C128 AA	1440	TAX	; számláló a maszk Kiszámításához
C129 A3 01	1450	LDA #\$01	; egy magas bit
C12B CA	1460 AA7	DEX	
C12C 30 03	1470	BMI AA8	
C12E 0A	1480	ASL A	; a maszk bit Kiszámítása
C12F D0 FA	1490	BNE AA7	
C131 A0 00	1500 AA8	LDY #\$00	
C133 24 02	1510	BIT \$02	; a módjelző vizsgálata
C135 50 05	1520	BVC AA3	
C137 51 22	1530	EOR (\$22),Y	; a bit invertálása
C139 4C 45 C1	1540	JMP AB1	; majd újrainása
C13C 10 05	1550 AA3	BPL AB0	
C13E 49 FF	1560	EOR \$FF	; a maszk invertálása
C140 31 22	1570	AND (\$22),Y	; a bit kikapcsolása
C142 2C	1580	.BYTE \$2C	; rejtett ORA
C143 11 22	1590 AB0	ORA (\$22),Y	; a bit beállítása
C145 91 22	1600 AB1	STA (\$22),Y	; a bit beírása a HIRES-tárba
C147 4C 52 C1	1610	JMP ROMON	; a ROM visszakapcsolása
C14A 78	1620 ROMOFF	SEI	; a megszakítások tiltása
C14B A5 01	1630	LDA \$01	; processzorport
C14D 29 F3	1640	AND #\$F3	; %11111001, az 1-2. bitek törölve
C14F 85 01	1650	STA \$01	
C151 60	1660	RTS	
C152 A5 01	1670 ROMON	LDA \$01	; processzorport
C154 09 06	1680	ORA #\$06	; %00000110, az 1-2. bitek beállítva
C156 85 01	1690	STA \$01	
C158 58	1700	CLI	; a megszakítások engedélyezése
C159 60	1710	RTS	
C15A 00 01 02	1720 TABL2	.BYTE \$00,\$01,\$02,\$03	
C15E 05 06 07	1730	.BYTE \$05,\$06,\$07,\$08	
C162 0A 0B 0C	1740	.BYTE \$0A,\$0B,\$0C,\$0D	
C166 0F 10 11	1750	.BYTE \$0F,\$10,\$11,\$12	
C16A 14 15 16	1760	.BYTE \$14,\$15,\$16,\$17	
C16E 19 1A 1B	1770	.BYTE \$19,\$1A,\$1B,\$1C	
C172 1E	1780	.BYTE \$1E	
C173 00 40 80	1790 TABL1	.BYTE \$00,\$40,\$80,\$C0	
C177 00 40 80	1800	.BYTE \$00,\$40,\$80,\$C0	
C17B 00 40 80	1810	.BYTE \$00,\$40,\$80,\$C0	
C17F 00 40 80	1820	.BYTE \$00,\$40,\$80,\$C0	
C183 00 40 80	1830	.BYTE \$00,\$40,\$80,\$C0	
C187 00 40 80	1840	.BYTE \$00,\$40,\$80,\$C0	
C18B 00	1850	.BYTE \$00	
C18C 00	1860 PUF1	.BYTE \$00	
C18D 00	1870 PUF2	.BYTE \$00	
C18E 00	1880 PUF3	.BYTE \$00	
C18F 00	1890 PUF4	.BYTE \$00	
C190 00	1900 PUF5	.BYTE \$00	
C191 00	1910 PUF6	.BYTE \$00	
C192	1920	.END	

ZEILEN:183 SYMBOLE:33 FEHLER:0

AA0	=C072	AA1	=C073	AA2	=C08C	AA3	=C0AB	AA4	=C0AC	AA5	=C0D4
AA6	=C0E2	AA7	=C12B	AA8	=C131	AA9	=C13C	AB0	=C143	AB1	=C145
CHAR	=C055	CLEAR	=C06A	COL1	=C0A3	COLOR	=C0A0	ERR	=C0CE	HIRES	=C03D
INIT	=C01B	INV	=C0BD	INVERZ	=C081	PUF1	=C18C	PUF2	=C18D	PUF3	=C18E
PUF4	=C18F	PUF5	=C190	PUF6	=C191	RESET	=C0C0	ROMOFF	=C14A	ROMON	=C152
SET	=C0BA	TABL1	=C173	TABL2	=C15A						

1.3. Complex Interface Adapter (CIA)

A C64-es egyik legfontosabb és legtevékenyebb része a CIA. A számítógépen belül a VIC „beszél”, a CIA pedig „lát” és „hall”. Hasonlóan a VIC-hez, a CIA is a processzor és a külvilág közötti kapcsolat létrehozására és fenntartására készült. Neve is ez: Complex Interface Adapter. Adapter, azaz kiegészítő egység, mert a processzor nélküle is működőképes, más kérdés, hogy mire jó. Interface, mert adatokat küldhet és fogadhat. Komplex, mert a tárgykörhöz tartozó összes feladatot önállóan képes ellátni. A CIA chip MOS gyártmányú félvezető áramkör, gyári kódja 6526. Ez az input/output illesztő a MOS 6522 továbbfejlesztett változata.

A C64-esben MOS 6526-os integrált áramkör végzi a billentyűzet lekérdezését, az RS232 port kezelését, a soros adatátvitelt, a joystick lekérdezését, a számítógépen belüli időzítési feladatok egy részét. Feladatainak végrehajtásához

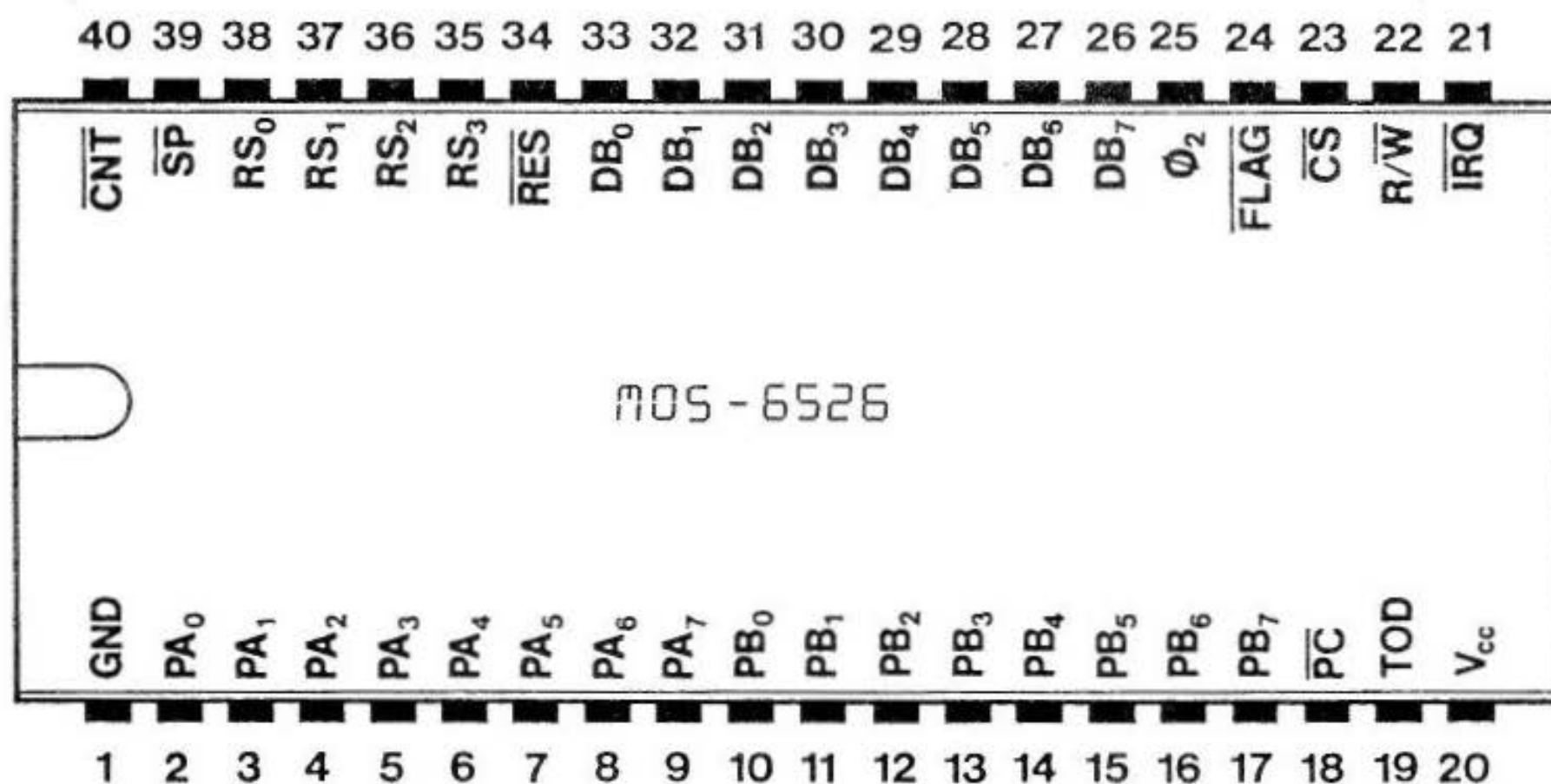
- 16 párhuzamos input/output vonallal,
- két független, 16 bites kaszkádolható időzítővel,
- 8 bites léptetőregiszterrel,
- 24 órás AM/PM órával,
- két handshake (kézfogó) bemenettel

rendelkezik. A C64-esben két CIA chip található, mert egy áramkör nem lenne képes ellátni minden feladatot.

A 6526 tokozása a 9. ábrán látható; tűinek megnevezése és jelentése:

1. GND: Ground, rendszerföld
- 2—9. PA0-PA7: Port A, A I/O port, 8 párhuzamos vonal
- 10—17. PB0-PB7: Port B, B I/O port, 8 párhuzamos vonal
18. \overline{PC} : Port Control, handshake vonal, amelyre rövid alacsony impulzus érkezik, ha a processzor elérte a B regisztert
19. TOD: Time Of Day, a valós idejű óra vezérlőüteme, 50 Hz. Az ütem a hálózati (220 V) feszültség frekvenciájától függ
20. Vcc: tápfeszültség, +5 V
21. \overline{IRQ} : Interrupt Request, megszakításkérelem. Állapota alacsony, ha az előre beállított megszakításmódhoz tartozó feltétel teljesült
22. R/ \overline{W} : Read/Write, bemenet. Ha alacsony, a regiszterekbe adat érkezik a processzortól
23. \overline{CS} : Chip Select, bemenet. Ha alacsony, a CIA regisztereihez a processzor hozzá kíván férni. Alacsony szintet csak a rendszerütem magas fázisában vehet fel
24. \overline{FLAG} : handshake vonal. Alacsony állapota a B porton megjelent adatok érvényességét jelzi

- 25. Φ_2 : rendszerütem
- 26—33. DB7-DB0: Data Bus, adatbusz
- 34. \overline{RES} : Reset, ha alacsony, a chip inicializálja magát
- 35—38. RS3-RS0: Register Select, a 16 regiszter címzése ezeken a vonalakon történik, ha a \overline{CS} alacsony
- 39. SP: Serial Port, a léptetőregiszter be- vagy kimenete
- 40. CNT: Counter, számlálóbemenet. Külső impulzusok triggerbemenete. Kimenet, a soros léptetőregiszterrel összekapcsolva ütemadó.



9. ábra. A CIA kiosztása

1.3.1. A CIA regiszterei

A CIA 6526 16 belső regiszterének többsége írható és olvasható. Bár a két CIA chip belső felépítését tekintve azonos, regiszterei egy része más-más funkciót lát el. A CIA2 chip eltérő regisztereit külön is felsoroljuk. A CIA1 báziscíme \$DC00, a CIA2 azonos sorszámú regiszterei egy lappal feljebb, a \$DD00 címtől érhetők el. A regiszterek sorszáma mögött a megfelelő tárcímet is megadjuk.

0. regiszter (\$DC00) PRA: Port Register A

- 0—4. bit JOY0, a 0. botkormány (Control Port 1) vezérlése. A bitek rendre: fel, le, bal, jobb, tűz
- 0—7. bit: PRA, alapállapotban a billentyűzetmátrix sorkiválasztása
- 6—7. bit: POT, a paddle potenciométerek kiválasztása

1. regiszter (\$DC01) PRB, Port Register B
0—4. bit: JOY1, az 1. botkormány (Control Port 2) vezérlése
0—7. bit: PRB, alapállapotban a billentyűzetmátrix oszlop kiválasztása
2. regiszter (\$DC02) DDRA: Data Direction Register A
0—7. bit: adatirány-regiszter. Ezek a bitek határozzák meg a PRA megfelelő bitjének adatirányát. 0 = input, 1 = output
3. regiszter (\$DC03) DDRB: Data Direction Register B
0—7. bit: adatirány-regiszter. Mint DDRA, de a B portra
4. regiszter (\$DC04) TA LO: Timer A Low
0—7. bit: az A időzítő alsó byte-ja. *Olvasás*: az A időzítő alsó byte aktuális értéke. *Írás*: a visszaszámlálás kezdőértéke, alsó byte
5. regiszter (\$DC05) TA HI: Timer A High
0—7. bit: az A időzítő felső byte-ja. *Olvasás*: az A időzítő felső byte aktuális értéke. *Írás*: a visszaszámlálás kezdőértéke, felső byte
6. regiszter (\$DC06) TB LO: Timer B Low
0—7. bit: Mint az A időzítő
7. regiszter (\$DC07) TB HI: Timer B High
0—7. bit: Mint az A időzítő
8. regiszter (\$DC08) TODTHS: Time Of Day 10TH Seconds
0—3. bit: valós idejű óra, tized másodpercek. *Olvasás*: a valós idejű óra tized másodpercei. *Írás*: ha CRB7 = 1: a riasztási idő, CRB7 = 0: az óraidő tized másodpercei. Minden érték BCD-kódban értendő
4—7. bit: nem használt
9. regiszter (\$DC09) TODSEC: Time Of Day Seconds
0—3. bit: valós idejű óra, a másodpercek „egyes” helyiértéke. *Olvasás*: a másodpercek BCD kódban. *Írás*: ha CRB7 = 1: a riasztási idő, CRB7 = 0: az óraidő
4—6. bit: valós idejű óra, másodpercek „tízes” helyiértéke. *Olvasás*: a másodpercek BCD kódban. *Írás*: ha CRB7 = 1: riasztási idő, CRB7 = 0: óraidő
7. bit: nem használt, mindig 0
10. regiszter (\$DC0A) TODMIN: Time Of Day Minutes
0—3. bit: a valós idejű óra perceinek „egyes” helyiértéke. *Olvasás*, *írás*: mint TODSEC
4—6. bit: a valós idejű óra perceinek „tízes” helyiértéke. *Olvasás*, *írás*: mint TODSEC
7. bit: nem használt, mindig 0

11. regiszter (\$DC0B) TODHRS: Time Of Day Hours
 - 0—3. bit: valós idejű óra, az órák „egyes” helyiértéke. Olvasás, írás: mint TODSEC
 4. bit: valós idejű óra, az órák „tízes” helyiértéke
 - 5—6. bit: nem használt, mindig 0
 7. bit: délelőtt/délután jelző, 1 = délután
12. regiszter (\$DC0C) SDR: Serial Data Register
 - 0—7. bit: soros léptetőregiszter, amelybe sorban egymás után tolhatók be a bitek. Az olvasás hasonló, de fordított folyamat. Az SP lábba az adatok innen érkeznek és megfordítva
13. regiszter (\$DC0D) ICR: Interrupt Control Register. Megszakítást ellenőrző-vezérlő regiszter. Írásra és olvasásra különböző funkciót lát el.
Olvasás: INT DATA
 0. bit: magas, ha az A időzítő alulcsordult
 1. bit: magas, ha a B időzítő alulcsordult
 2. bit: magas, ha az óraidő azonos a riasztási idővel
 3. bit: magas, ha az SDR megtelt vagy üres, az üzemmódtól függően
 4. bit: (1) magas, ha a $\overline{\text{FLAG}}$ lábon lefutó él jelent meg.
 (2) kazetta vagy soros busz kiszolgáláskérése (SRQ)
 - 5—6. bit: nem használt, mindig 0
 7. bit: magas, ha az INT DATA és az INT MASK legalább egy bitje páronként magas (pl. INT DATA 2 és INT MASK 2)*Írás: INT MASK*
 - 0—4. bit: funkcióik a fentivel megegyeznek
 - 5—6. bit: nem használt, mindig 0
 7. bit: ha 1, minden magas bitet átvesz a maszkregiszter. 0: minden magas bit törli a maszkbitet, a többi bit nem változik. Az INT DATA regiszter minden bitje törlődik olvasás után
14. regiszter (\$DC0E) CRA: Control Register A; A vezérlőregiszter
 0. bit: ha 0, az A időzítő áll; 1: az A időzítő fut
 1. bit: ha magas, az A időzítő alulcsordulásait a PB₆ láb jelzi
 2. bit: ha 0, az A időzítő alulcsordulása a PB₆ lábon rendszerütemnyi magas jelet állít elő. 1: az A időzítő alulcsordulása invertálja a PB₆ lábat
 3. bit: ha 0, az A időzítő alulcsordulás után újra betölti a kezdőértéket és folytatja a számlálást. 1: az A időzítő a megadott kezdőértéktől csak egyszer számol vissza, azután megáll
 4. bit: ha magas, az A időzítőbe új értéket tölthetünk. A bit beírás után azonnal törlődik
 5. bit: ha 0, az A időzítő a rendszerütemeket számlálja 1: az A időzítő a CNT bemenetre érkező felfutó éleket számlálja,
 6. bit: ha 0, az SP láb (és ezáltal az SDR) soros kimenet. 1: az SP láb soros bemenet

7. bit: ha 0, a valós idejű óra 60 Hz frekvencia szerint számlál. 1: a valós idejű óra 50 Hz frekvencia szerint számlál
15. regiszter (\$DC0F) CRB: Control Register B. B vezérlőregiszter
- 0. bit: ha 0, a B időzítő áll. 1: a B időzítő fut
 - 1. bit: ha magas, a B időzítő alulcsordulásait a PB7 láb jelzi
 - 2. bit: ha 0, a B időzítő alulcsordulása rendszerütemnyi magas jelet állít elő a PB7 lábon. 1: a B időzítő alulcsordulása invertálja a PB7 lábat
 - 3. bit: ha 0, a B időzítő alulcsordulás után újra betölti a kezdőértéket, és folytatja a számlálást. 1: a B időzítő csak egyszer számlál vissza, azután megáll
 - 4. bit: ha magas, a B időzítőbe új értéket tölthetünk. A bit beírás után azonnal törlődik
 - 5—6. bit: a B időzítő léptetőütemének forrását határozza meg. 00: a rendszerütem számlálása. 01: a felfutó CNT élek számlálása. 10: az A időzítő alulcsordulásainak számlálása. 11: az A időzítő alulcsordulásainak számlálása, ha a CNT magas
 - 7. bit: ha 0, a TOD óraidejének beállítása. 1: a TOD riasztási idejének beállítása

A CIA2 eltérő regiszterei

0. regiszter (\$DD00 tárcímen) PRA: Port Register A. Az A portregiszter
- 0. bit: VA₁₄, a videoram 14. címbitje
 - 1. bit: VA₁₅, a videoram 15. címbitje
 - 2. bit: Tx_D, RS232 adatkimenet, Transmitted Data
 - 3. bit: ATN, ATN jel, kimenet a soros buszhoz
 - 4. bit: CLOCK, a soros busz órajelkimenete
 - 5. bit: DATA, a soros busz adatvonala, kimenet
 - 6. bit: CLOCK, a soros busz órajelbemenete
 - 7. bit: DATA, a soros busz órajelbemenete
1. regiszter (\$DD01 tárcímen) PRB: Port Register B. A B portregiszter. Csak RS232 esetén foglalt
- 0. bit: Rx_D, Receive Data, adatfogadás
 - 1. bit: RTS, Request To Send, adatküldési kérelem
 - 2. bit: DTR, Data Terminal Ready, az adat továbbításra kész
 - 3. bit: RI, Ring Indicator
 - 4. bit: DCD, Data Carrier Detect, az adatfogadás nyugtázása
 - 6. bit: CTS, Clear To Send
 - 7. bit: DSR, Data Set Ready
13. regiszter: (\$DD0D tárcímen) ICR: Interrupt Control Register, megszakításvezérlő regiszter
- 4. bit: Rx_D, RS232 esetén Receive Data
 - 7. bit: nem az $\overline{\text{IRQ}}$, vezetékre, hanem az $\overline{\text{NMI}}$ vezetékre van kötve

1.3.2. A CIA1 feladatai

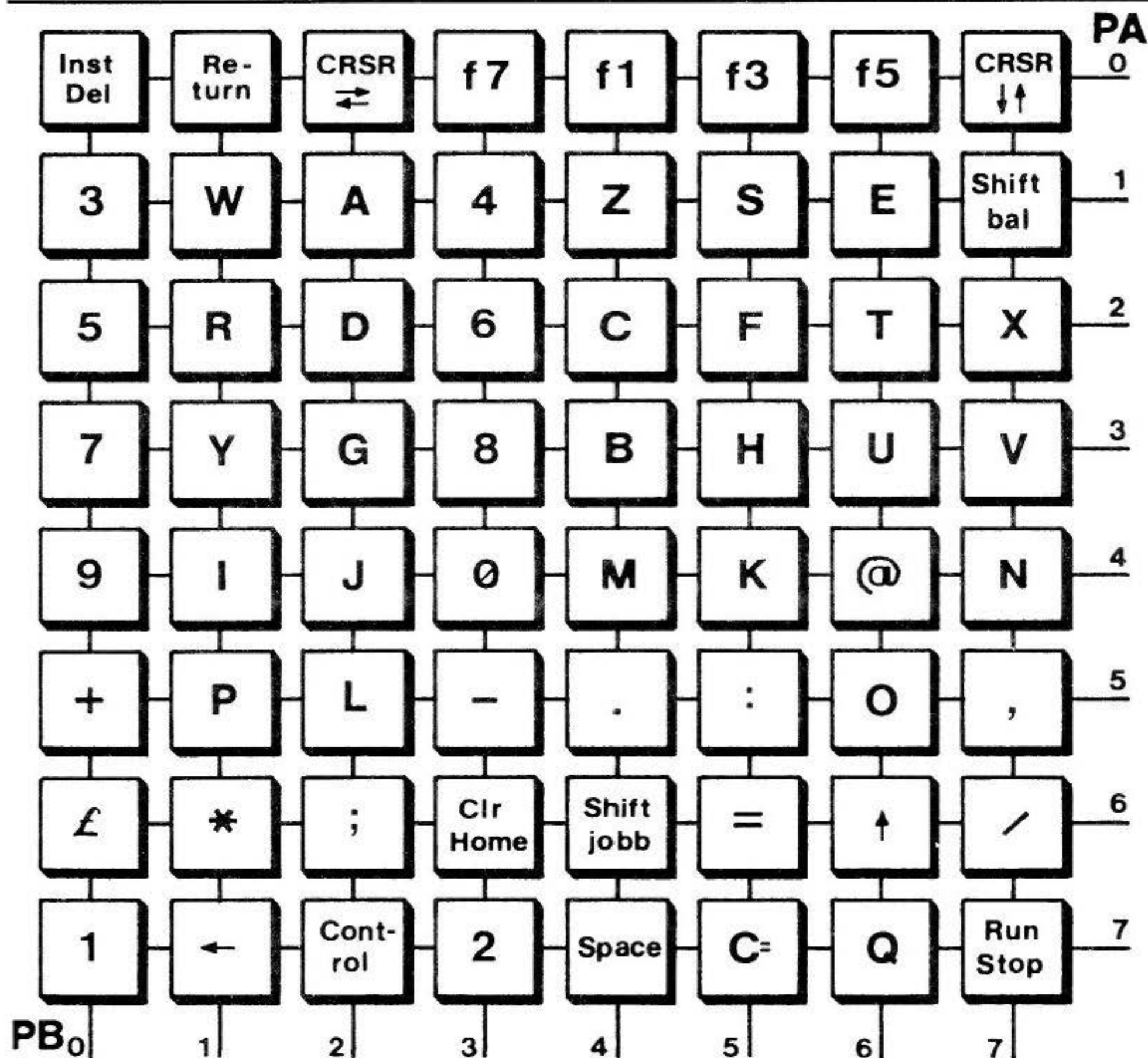
A CIA1 legfontosabb feladata a *billentyűzet illesztése* a processzorhoz. Végül soron ennek a chipnek köszönhető, hogy a processzor értesül szándékainkról. A billentyűzeten lévő 66 gomb valójában csak 65 billentyűt jelent, mert a SHIFT LOCK és a bal SHIFT ugyanazon az áramkörön van. Ha a CIA 16 adatvonalát mint sor- és oszlopindexeket két nyolcas egységre bontjuk, 64 elemű mátrixot kapunk. Akkor hogyan lehetséges 65 különböző billentyű? A megoldás a RESTORE billentyű bekötésében van. Ez a billentyű nem tartozik a CIA hatósugarába, mert közvetlenül a processzor NMI vonalára van kötve, lenyomása tehát megszakítást generál. A többi 64 billentyűt viszont valóban a CIA érzékeli.

Azt gondolhatjuk, a lekérdezés egyszerű: a processzor megvizsgálja, hogy az A port (\$DC00) bitjei között van-e alacsony. Ha van, akkor megvizsgálja a B port bitjeit is, és ha ott is van alacsony, akkor a billentyű sor- és oszlopindexe már meg is van. A valóságban ez annyiban más, hogy a billentyűzetet az operációs rendszer \$EA87 címén kezdődő gépi program alapján kérdezi le a processzor.

A program alapján a billentyű felismerése a következőképpen történik. Alapállapotban az A port (\$DC00) kimenet, a B port (\$DC01) pedig bemenet. Amikor a program meg akarja vizsgálni, hogy van-e lenyomott billentyű, az A port minden vonalát alacsonyra állítja, és amennyiben van lenyomott billentyű, akkor a B port vonalai között is van alacsony állapotú. Azt, hogy melyik ez a billentyű, ezután számítja ki. Az A port vonalait sorban egymás után alacsonyra állítja, és mindegyik sornál megnézi, hogy a B port bitjei között van-e alacsony. Ha nincs, akkor tovább keres, ha van, akkor a billentyűt azonosította. Az itt ismertetett program másodpercenként 60-szor fut le a megszakítórutin részeként, így még egy nagyon gyors gépiró sem tudja zavarba hozni a programot.

Az A és B portok másik feladata a *joystick-portok* illesztése. A 0. joystick az A porthoz, az 1. joystick a B porthoz csatlakozik. A botkormány csak az alsó öt bitet (0–4) foglalja le. A bitek sorban a következő irányoknak felelnek meg: fel, le, bal, jobb, tűz. A joystickok többnyire nyolcirányúak. A le-balra irány az 1-2. bitek alacsony állapotának felel meg. A botkormányok érzékeléséhez tartozik egy érdekesség: a 0. joystick szimulálható az 1, ←, CONTROL, 2, SPACE billentyűkkel. Itt csak egy változat létezik. Az 1. joystick szimulálása több billentyűcsoporttal is történhet, de egy billentyűt folyamatosan lenyomva kell tartani. Ez a lenyomva tartandó billentyű szerepel a sor elején:

1	INST-DEL, 3, 5, 7, 9
←	RETURN, W, R, Y, I
CONTROL	CRSR↔ A, D, G, J
2	f7, 4, 6, 8, 0
SPACE	f1, Z, C, B, M
C=	f3, S, F, H, K
Q	f5, E, T, U, @
RUN/STOP	CRSR↓↑, SHIFT bal, X, V, N



10. ábra. A billentyűzet mátrixelrendezése

A legtöbb játékprogram a billentyűzetről is irányítható. Az újabb keletű programok viszont már az A port 7. bitjét is magasan tartják. Ennek köszönhető ugyanis, hogy a billentyűzetről nem lehet szimulálni a joystickot. Magyarországon sajnos a 2 billentyűhöz tartozó variáció terjedt el; így éppen az egyik legtöbbet használt billentyű megy tönkre idő előtt.

A CIA1 A portja vezérli a *paddle-portok* kiválasztását is. Az A port (\$DC00) 6. és 7. bitje választja ki, hogy az analóg/digitális átalakító melyik portra (Control Port 1 vagy 2) csatlakoztatott paddle ellenállását méri. A 6. bit a B, a 7. bit az A paddle kiválasztását vezérli.

A CIA1 nem kevésbé fontos feladata a *hardvermegszakítás* időzítésének biztosítása is. Erre az A időzítő szolgál. A CIA1-et az operációs rendszer \$FDA3 címen kezdődő gépi rutinja úgy állítja be, hogy minden 16421. ütemciklus után IRQ-t generáljon. Az A időzítő folyamatos

üzem módban működik. Ha figyelembe vesszük az ütemfrekvenciát (985,25 kHz), 16421 ciklus kb. 0,0167 másodperc alatt zajlik le. Ez majdnem pontosan 1/60 másodperc.

Az ICR regiszter (\$DC0D) 0. és 7. bitje, valamint a CRA regiszter (\$DC0E) 0. és 7. bitje vesz részt a folyamatban. Az A időzítő lefutásakor a CIA1 magasra állítja az ICR 0. és 7. bitjét. A 7. bit egy inverteren keresztül össze van kötve a chip $\overline{\text{IRQ}}$ lábával. Azon ilyenkor alacsony szintű jel jelenik meg, ami a processzort arra készíti, hogy megszakítván addigi munkáját, végrehajtsa a megszakítási rutint.

A CIA1 B időzítője a kazettás egység kezelésekor jut szerephez, egyébként nem használatos.

1.3.3. A CIA2 feladatai

A CIA2 a külvilág más oldalaival tart fenn összeköttetést. Feladata a soros busz és az RS232 port kezelése, tehát a C64-es és egyéb mikroprocesszoros egységek közötti kommunikáció megteremtése és lebonyolítása.

A Commodore cég saját gépeinél rendszeresített *soros busz* a nemzetközi szabványként is elfogadott és IEC-625 néven ismert soros adatátviteli rendszer egy változata. A Commodore valószínűleg azért módosította az eredeti buszrendszert, mert az viszonylag költséges. Az eredetileg 24 tűs csatlakozó így 5 tűsre csökkent, ennek megfelelően az átviteli kábelnek is csak 5 eresnek kell lennie. (A valóságban 6 tűs és 6 eres, mert a csatlakozón van egy RESET tű is.) Ez egy olcsó, házi számítógép szempontjából nagyon jelentős árcsökkenést tesz lehetővé. A módosításnak természetesen ára is van; az adatátvitel sebessége lényegesen alacsonyabb. A módosított busz időegység alatt nagyságrenddel kevesebb adatot képes továbbítani.

A soros busz vonalai egy 6 tűs DIN szabványú csatlakozó aljzatba futnak. A csatlakozó lábkiosztását és szemléltető ábráját *A csatlakozók* című fejezet megfelelő része ismerteti.

A vonalak feladata:

1. $\overline{\text{SRQ}}$: Service Request In; a külső egység számára fenntartott jelzővonal. Ezen jelezheti, hogy pl. adatra vár. A vonal a CIA1 $\overline{\text{FLAG}}$ bemenetére van kötve, és összeköttetésben áll a kazettás egység CASS RD vonalával is.
2. $\overline{\text{GND}}$: Ground; rendszerföld.
3. $\overline{\text{ATN}}$: Attention In/Out. A vonalon a buszvezérlő figyelmeztető jelzést küld a külső egységek felé, hogy igénybe kívánja venni a buszt. A külső egység szintén ezen a vonalon jelzi hasonló szándékát.
4. $\overline{\text{CLK}}$: Clock In/Out. Az adatokat küldő egység (TALK) egy impulzussal ezen a vonalon jelzi egy bit érvényességét a fogadó egység (LISTEN) számára. Az átvitelt szinkronizálja.
5. $\overline{\text{DATA}}$: Data In/Out. E vonalon zajlik le az adatforgalom. A byte-ok bitjei a legalsótól kezdve sorban jelennek meg a vonalon, érvényességüket a $\overline{\text{CLK}}$ impulzusa jelzi.

6. $\overline{\text{RESET}}$: Reset Out. A processzor $\overline{\text{RESET}}$ vonala. Külső inicializálást végez.

A buszon végzett adatátvitel esetén rendkívül fontos az időzítés. Végző soron ezen múlik az adatvesztés elkerülése. A DATA és CLK vonalak által vezérelt buszon az adatokat a vezérlőjelektől a vonalakon megjelenő impulzusok hossza különbözteti meg.

Kövessük végig a buszvezérlés és az adatátvitel folyamatát.

Címzés

- (1) az $\overline{\text{ATN}}$ jelszintje alacsonyra vált, utasítás következik
- (2) röviddel ezután a $\overline{\text{CLK}}$ is alacsonyra vált, és mindaddig alacsony marad, amíg a külső egység az adatvonalat alacsonyra nem állítja. Ezt, ha működik, 1 ms-on belül köteles megtenni
- (3) a $\overline{\text{DATA}}$ vonal mindaddig alacsony marad, amíg a külső egység nincs készen az adat fogadására
- (4) ezután az adatvonalon sorban megjelennek az egységszám bitjei, mindegyik egy magas $\overline{\text{CLK}}$ impulzus kíséretében. Ezek az impulzusok lényegesen rövidebbek, mint az adatfogadás előkészítésében részt vevő jelek
- (5) amint a külső egység fogadta az utolsó bitet is, és a $\overline{\text{CLK}}$ hosszabb idő óta alacsony, a külső egység max. 1 ms alatt alacsonyra állítja a $\overline{\text{DATA}}$ vonalat, jelezve, hogy fogadta az adatbyte-ot
- (6) a vezérlő ezt nyugtázva hamarosan magasra váltja az $\overline{\text{ATN}}$ vonalat

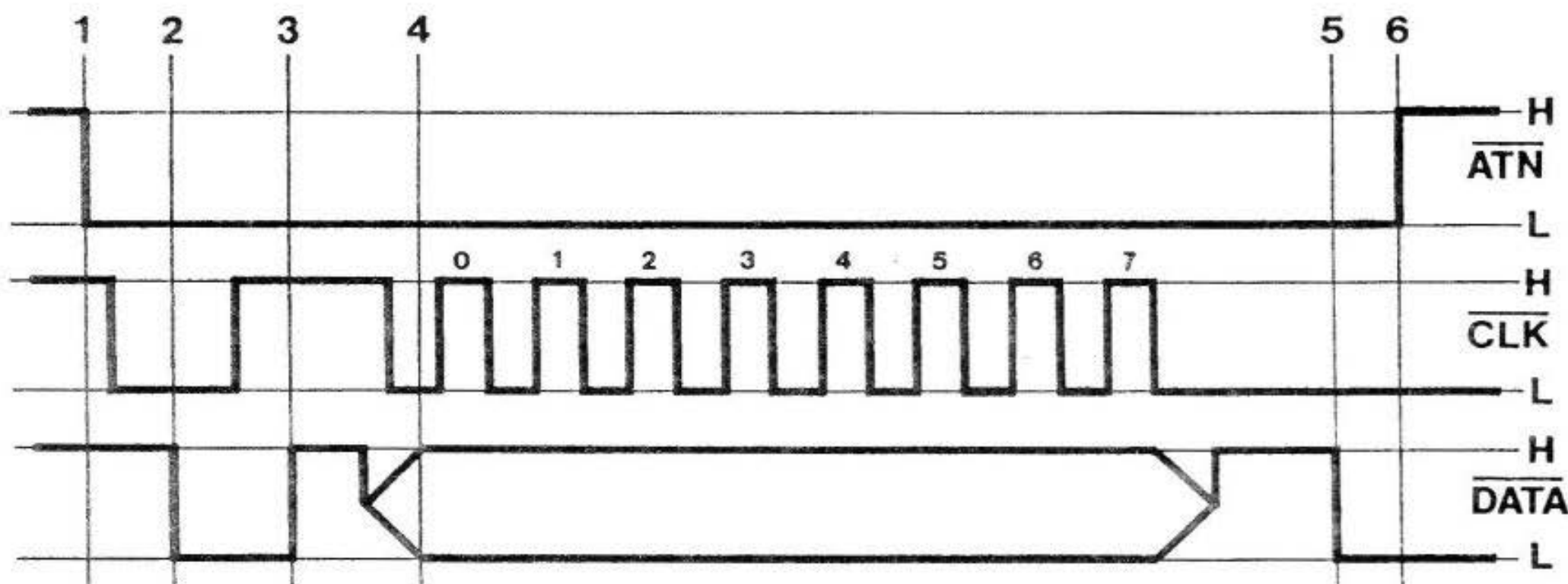
Adatátvitel

Adatátvitel esetén meg kell különböztetni TALK (beszélő) és LISTEN (hallgató) egységet. A kapcsolat mindig TALK és LISTEN egység kapcsolata. E két egység közösen, összehangolt módon vezérli a buszt.

- (7) a TALK egység a $\overline{\text{CLK}}$ vonalat magasra állítva jelzi, hogy az adatbyte továbbításra készen áll
- (8) ezután a LISTEN egység a $\overline{\text{DATA}}$ vonal magasra állításával jelzi készenlétét
- (9) ezt követően minden magas $\overline{\text{CLK}}$ impulzus egy adatbit érvényességét jelzi. Ez itt is pontosan úgy zajlik le, mint a címzésnél
- (10) a LISTEN egység az adatbyte átvételét a $\overline{\text{DATA}}$ vonal alacsonyra állításával jelzi.

Az $\overline{\text{ATN}}$ vonal az adatátvitel alatt folyamatosan magas. A további adatbyte-ok a 7—10 pontok alatt felsorolt műveletek ciklikus ismétlődése közben továbbítódnak. Az utolsó adatbyte után még egy egyezményes lezáró jel is a buszra kerül (ez a CHR\$(13) vagy Carriage Return).

- (11) Mielőtt a következő byte-ot továbbítaná, a TALK egység a $\overline{\text{CLK}}$ vonal alacsonyra állításával max. 0,2 ms-ig vár. Ha a $\overline{\text{CLK}}$ 0,2 ms-nál hosszabb ideig magas, ez a LISTEN egység számára az adatátvitel végét jelenti.
- (12) Ekkor az adatvonalat alacsonyra állítja, ezzel nyugtázza a végjelet.
- (13) A busz ezután a szokásos adatátvitellel egy $\text{CHR}\$(0)$ -t továbbít.
- (14) Végül a busz nyugalmi állapotba kerül; $\overline{\text{ATN}}$ és $\overline{\text{DATA}}$ magas, $\overline{\text{CLK}}$ alacsony.

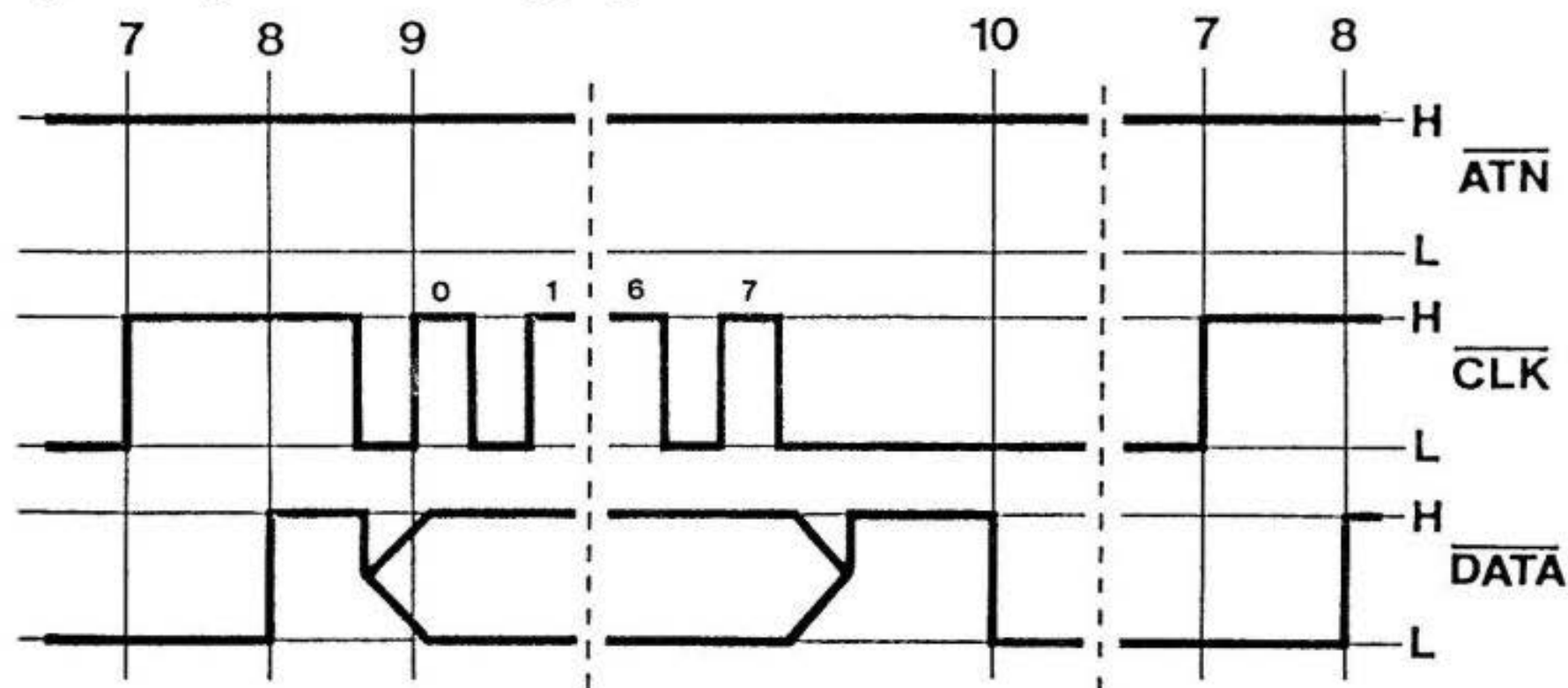


11. ábra. A címzés folyamata

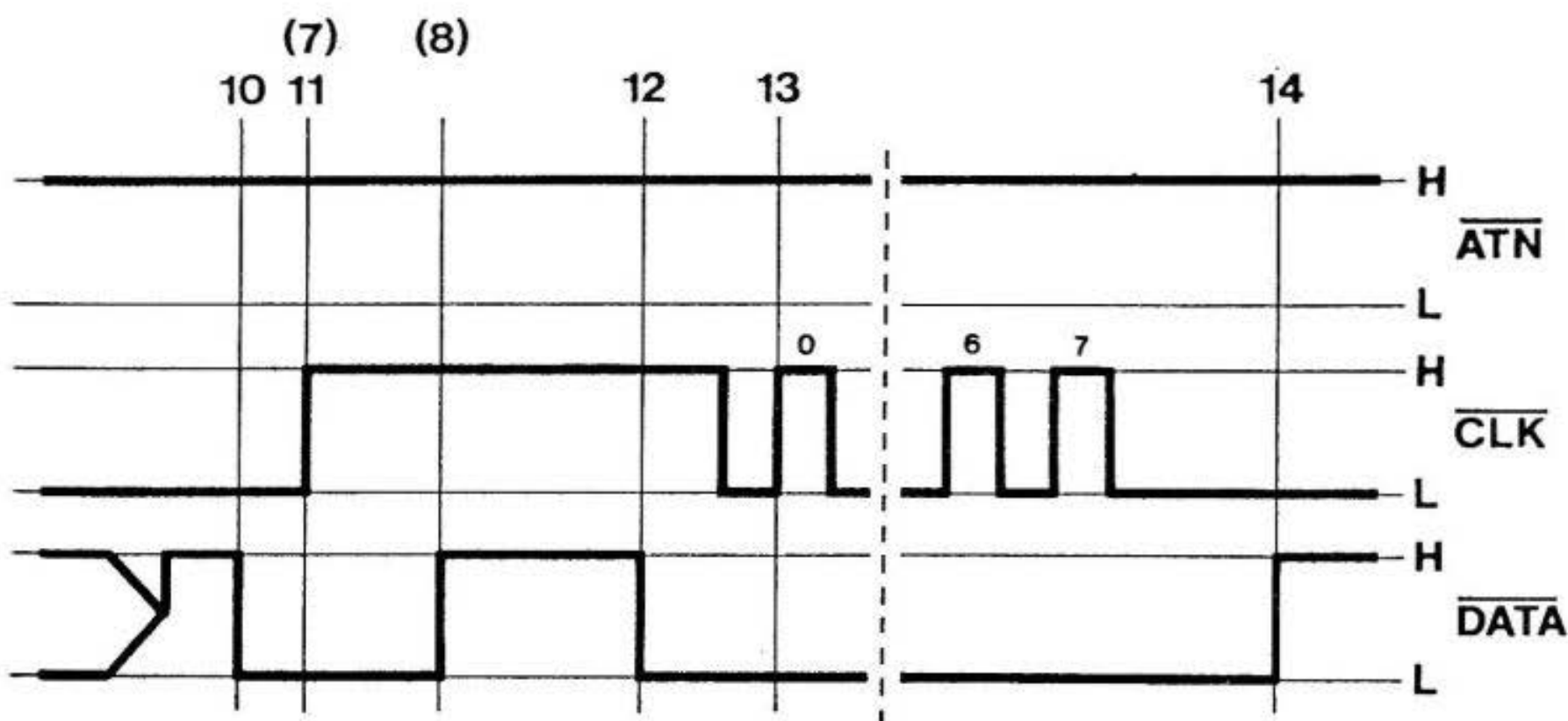
A soros busz bonyolult vezérlése ellenére biztonságos. A busz hátránya viszonylagos lassúsága. A C64-es értékét az bizonyítja a legjobban, hogy még hardvere is rugalmas, vannak benne még fejlesztési tartalékok. Például megoldható a soros busz átvitelének jelentős gyorsítása anélkül, hogy ez a biztonság feladásához vezetne. Az ún. TURBO üzemmód persze csak a lemezegység (VC—1541, 1570, 1571) és a számítógép között valósítható meg, mert a buszt gyorsító programot mindkét egységben el kell helyezni, és egy nyomtatóban általában nincs címezhető RAM. A buszgyorsító programok többnyire egészlemez (BACKUP) másolóprogramokban fordulnak elő, de megjelentek már más jellegű programokban is. 5—7-szeres sebességnövelésük rendkívül előnyös nagy adatállományok mozgatása esetén. Hátrányuk azonban, hogy az adatvesztés kockázati tényezője jóval nagyobb, és működésük közben a gép gyakorlatilag „merev”. További hátrányként jelentkezik, hogy ahol a gyorsító program helyezkedne el a memóriában (a drive memóriájában), ott relatív file-kezelés esetén oldalszektor blokkok vannak. A gyorsító modul csak szekvenciális és user file-ok esetén alkalmazható.

RS232-es illesztőjű külső egység a C64-es USER-portjára csatlakoztatható. Az USER-port kivezetéseinek jelentős része közvetlen kapcsolatban van a CIA2-vel. Az USER-port 24 kivezetését a *A csatlakozók* című fejezet megfelelő része mutatja be, ezért ezzel nem foglalkozunk részletesen. A csatlakozó kivezetésein megtalálható a CIA2 B portjának (\$DD01) minden bitje, a CNT, SP, $\overline{\text{PC}}$ és $\overline{\text{FLAG}}$ vonalak, valamint az A port (\$DD00) PA2 vonala. Ugyanígy a CIA1 CNT és SP vonala is megtalálható.

Az RS232 interfész a soros adatátvitel szabványos illesztőegysége. A kezeléséhez szükséges programot a C64-es operációs rendszere tartalmazza. Az RS232-est a C64-es szoftver szabályos külső egységként kezeli, ennek következtében az egységen be-, illetve kimenetre egyaránt nyitható file. Az egység száma 2.



12. ábra. Az adatátvitel folyamata



13. ábra. Az adatátvitel vége

A kezelő szoftver file-nyitáskor kijelöl egy input és egy output puffert. Ez a két puffer a BASIC RAM utolsó két lapján helyezkedik el. Mindkettő 256 byte hosszú. Az operációs rendszer mind a file megnyitásakor, mind lezárásakor végrehajt egy CLR BASIC-utasítást.

Az RS232 használatakor meg kell adni egy vezérlőregisztert és egy parancsregisztert. A *vezérlőregiszter* (Control register) feladata az átviteli sebesség és az adat- illetve stopbitek számának meghatározása.

A vezérlőregiszter alsó négy (0—3) bitje határozza meg az átviteli sebességet baud (bit/s) egységben.

%0000	#0	programozott	%1000	#8	1200 baud
%0001	#1	50 baud	%1001	#9	1800 baud
%0010	#2	75 baud	%0110	#10	2400 baud
%0011	#3	110 baud	%1011	#11	3600 baud
%0100	#4	134,5 baud	%1100	#12	4800 baud
%0101	#5	150 baud	%1101	#13	7200 baud
%0110	#6	300 baud	%1110	#14	9600 baud
%0111	#7	600 baud	%1111	#15	19200 baud

A C64-esen csak az 50 — 2400 baudos sebességek érhetők el. A vezérlőregiszter 4. bitje nem definiált. Az 5 — 6. bit határozza meg az egy szóként átvitt bitek számát:

%00	#0	8 bit
%01	#32	7 bit
%10	#64	6 bit
%11	#96	5 bit

A regiszter 7. bitje a stopbitek számát adja meg. Az átvitel során minden szó után 1 vagy 2 stopbit továbbítódik.

%0	#0	1 stopbit
%1	#128	2 stopbit

A *parancsregiszter* (Command register) határozza meg az átvitel módját, a handshake vezetékek számát és a paritás-ellenőrzést.

A 0. bit szabja meg a handshake vezetékek számát:

%0	#0	3 vonalas handshake
%1	#1	határozatlan

A parancsregiszter 1 — 3. bitjei nem definiáltak. A 4. bit az átvitel módját adja meg:

%0	#0	duplex
%1	#16	fél duplex

Az 5 — 7. bitek a paritás-ellenőrzés módját szabják meg:

%000	#0	nincs ellenőrzés, nincs 8. adatbit
%001	#32	páratlan paritás
%010	#64	nincs ellenőrzés, nincs 8. adatbit
%011	#96	páros paritás
%100	#128	nincs ellenőrzés, nincs 8. adatbit
%101	#160	nincs ellenőrzés, a 8. adatbit mindig 1
%110	#192	nincs ellenőrzés, nincs 8. adatbit
%111	#224	nincs ellenőrzés, a 8. adatbit mindig 0

A vezérlő- és a parancsregiszter tartalmát az OPEN utasítás közvetíti az operációs rendszerhez. Az utasítást a következő módon kell megadni:

OPEN 1,2,1,CHR\$(7+32+128)+CHR\$(0+0+224)

Ez 600 baudos átviteli sebességet, 7 bites adatszót, 2 stopbitet, 3 vonalas handshake kapcsolatot, duplex átvitelt, zéró 8. adatbitet és kikapcsolt paritás-ellenőrzést jelent.

A táblázatban megadott átviteli sebességektől eltérhetünk. A file-nyitó utasítást ekkor így kell megadni:

OPEN 1,2,1,CHR\$(160)+CHR\$(224)+CHR\$(90)+CHR\$(2)

Ez 700 baudos átviteli sebességet jelent. Az átviteli sebesség az alábbiak szerint számítható.

Ha a sebesség 700 baud ($X = 492\,662/\text{baud} - 101$):

$$X = 492\,662/700 - 101 = 602,4 \text{ } 602$$

Felbontva alsó és felső byte-ra ($XH = \text{INT}(X/256)$, $XL = X - XH*256$):

$$XL = 90, XH = 2$$

A vezérlőregiszter 0—3. bitjei ilyenkor alacsonyak legyenek. A programozott átviteli sebesség 8 baud és 2400 baud közé eshet.

Az operációs rendszer RS232 üzemben az alábbi tárhelyeket foglalja le az adatátviteli feladatok megoldásához.

A nulláslapon:

- \$A7: az input bit ideiglenes tárolása
- \$A8: az input bitek számlálása
- \$A9: START-bit ellenőrzése
- \$AA: INPUT-puffer szerkesztési célokra
- \$AB: INPUT-paritásbit
- \$B4: az output bitek számlálása
- \$B5: az output bit ideiglenes tárolása
- \$B6: output-byte puffer
- \$BD: OUTPUT-paritásbit
- \$F7—F8: input-puffer mutató
- \$F9—FA: output-puffer mutató

A RAM-ban:

- \$0293: az RS232 vezérlőregisztere
- \$0294: az RS232 parancsregisztere
- \$0295—96: bit timing, átviteli időzítő
- \$0297: az RS232 állapotregisztere
- \$0298: az adatbitek száma

- \$0299: — 9A: output átviteli sebesség
- \$029B: input-puffer mutató írás közben
- \$029C: input-puffer mutató olvasás közben
- \$029D: output-puffer mutató olvasás közben
- \$029E: output-puffer mutató írás közben

Az adatátvitel során fellépő hibákat az RS232 *állapotregiszterében* vizsgálhatjuk. Az állapotregiszter biteinek jelentése:

- 0 ha magas, paritáshiba
- 1 ha magas, kerethiba
- 2 ha magas, az input puffer megtelt
- 3 nem használt
- 4 ha magas, nem érkezett CTS (Clear To Send) jel
- 5 nem használt
- 6 ha magas, nem érkezett DSR (Data Set Ready) jel
- 7 ha magas, a BREAK jelet az illesztő érzékelte (STOP)

1.3.4. A CIA programozása

A két CIA *megszakításkezelés* szempontjából némileg eltér egymástól. Ez az eltérés nem a két áramkör különbözőségéből ered, hanem az áramkörök eltérő bekötéséből. A CIA1 IRQ vonala a processzor IRQ vonalára, a CIA2 IRQ vonala pedig a processzor NMI vonalára csatlakozik. Emiatt a CIA1 csak IRQ típusú, azaz letiltható megszakítást, míg a CIA2 csak NMI típusú, azaz nem tiltható megszakítást hozhat létre.

A CIA-val létrehozható megszakítások fajtáit a chip ICR (\$Dx0D) regiszterének funkcióival azonosíthatjuk. Az ICR öt bite jelez megszakítást:

- 0: magas, ha az A időzítő csordult alul,
- 1: magas, ha a B időzítő csordult alul,
- 2: magas, ha a TOD óra elérte a beállított riasztási időt
- 3: magas, ha az SDR (\$Dx0C) megtelt vagy üres, az üzemmódtól függően
- 4: magas, ha a FLAG lábon lefutó él jelent meg (kézfogó-üzemmód)

Az A és B időzítő alulcsordulása, illetve az általuk kiváltott megszakítás ütemezési feladatok megoldását teszi lehetővé. A bekapcsolt számítógép kurzora is egy ilyen megszakítás miatt villoghat. Az időzítők beállításával biztosíthatjuk a ciklikusságot olyankor is, ha egy program futása közben pl. az analóg/digitális átalakító SID-ben lévő regisztereit (\$D419 és \$D41A) kell bizonyos időnként kiolvasni és tartalmukat megjeleníteni. Ugyanígy a megszakításkére-

lemmel jelezheti a CIA, ha az óraidő a riasztási időt elérte. A soros adatátvitel is lehetetlenné válna, ha a CIA nem jelezné, hogy a léptetőregiszter megtelt vagy kiürült.

A két CIA az ICR (\$Dx0D) regiszter 4. bitjét más-más célra használja. A CIA1 $\overline{\text{FLAG}}$ vezetéke a soros busz $\overline{\text{SRQIN}}$ vonalára csatlakozik. Tehát ha a soros buszra kötött külső egység kiszolgálási kérelmet jelez ezen a vonalon, a CIA1 megszakítást generálhat. (Szalagos üzemmódban ez a megszakítás aktivizálódik, mert a kazettás egység CASS RD vonala a CIA1 $\overline{\text{FLAG}}$ vonalára van kötve. Más perifériát, ami ezt a megszakítást kihasználná, a Commodore nem gyárt.) A CIA2 $\overline{\text{FLAG}}$ vezetéke a felhasználói portra csatlakozik, és csak az RS232 használata esetén van jelentősége: az adat fogadását (RxD — Receive Data) jelzi.

Multiplexelt regiszterek

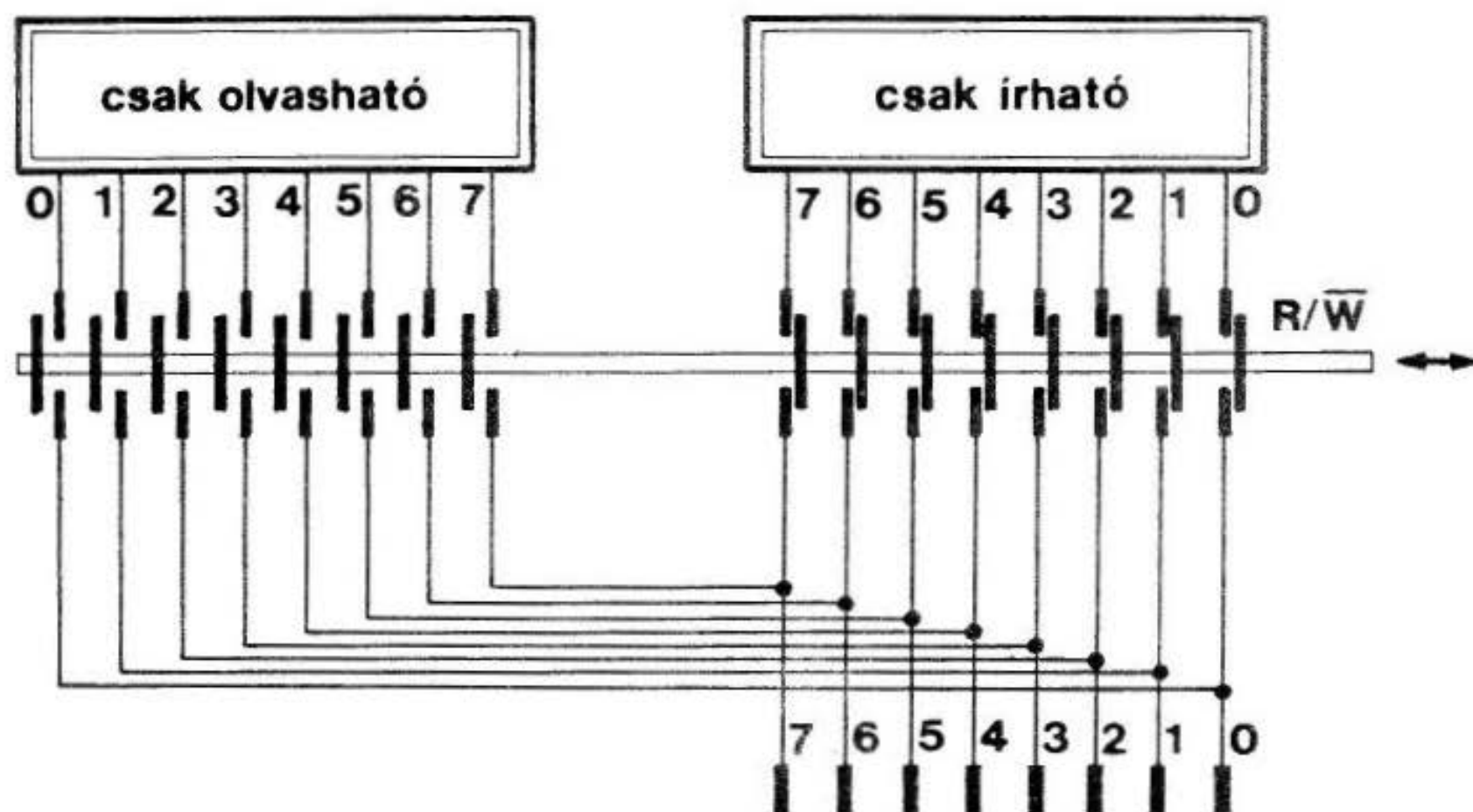
A VIC egyes regisztereinél már említettük, hogy az I/O chipekbe olyan regisztereket építettek, amelyeknek „egyszerre” több funkciójuk is van. A CIA regisztereinek egy része is ilyen. Ilyenek az AM/PM óra és az időzítő regiszterei, de az ICR regiszter is kettős.

A laikus számára furcsa lehet, hogy pl. az ICR regiszternek (\$DC0D vagy \$DD0D) hogyan lehetséges két elnevezése (INT DATA és INT MASK) és két különböző funkciója. A megoldást az adja, hogy az I/O chip érzékelni képes, hogy egy regiszteréhez írás vagy olvasás céljából fordul a processzor. A jelzés a chip $\text{R}/\overline{\text{W}}$ lábán jelenik meg. A regiszter a chipen belül kétfelé oszlik: az egyik a kívülről, a másik a belülről érkező adatok tárolására szolgál. Nevezük őket írás- és olvasásregiszternek. Az írásregisztert kívülről csak írni, az olvasásregisztert pedig csak olvasni lehet. A megoldást a 14. ábra szemlélteti.

A kapcsolás éppen írásra van beállítva. A $\text{R}/\overline{\text{W}}$ vezetékek pontosan azt végzi, amit a kezünk tesz a mechanikus kapcsolóval.

Az időzítő és az AM/PM óra regiszterei azonban még tovább bonthatók. Ez főként az óra és az időzítő írásából és olvasásából következik.

Az órák BASIC programból (sőt akár parancsmódból) is indíthatók, írhatók és olvashatók, illetve megállíthatók. A BASIC közismerten nem túl gyors, de még a gépi program sem biztos, hogy egy adott időpontban (néhány μs) képes lenne pontosan kiolvasni az óraidőt vagy az időzítő értékét. Ez különösen az időzítő esetében vezethetne pontatlansághoz, amelyek legkisebb léptetési ideje 1 μs . Ez a sebesség messze felülmúlja (15-20-szorosan) a szoftver olvasási sebességét. Persze ez a probléma csak akkor állna fenn, ha az óráknak (időzítőknak) nem lenne „részdő” üzemmódjuk. Az olvasás ezért így zajlik: a chip észlelte, hogy az óra (vagy az időzítő) felső regisztereire olvasáskérelem érkezett. Ekkor az éppen aktuális időt (számlálóértéket) rögzíti az olvasásregiszterekben. Ezt követően a regiszterek értékét egészen addig nem frissíti, amíg a legalacsonyabb regisztert nem olvassuk. Az óra (ill. időzítő) alapjául szolgáló számláló természetesen az olvasás egész ideje alatt változatlanul futott tovább, ezért a következő olvasás is pontos és objektív eredményt ad. Ha a legalsó regisztert is olvastuk, az állónak látszó óra újra „elindul”. Írás esetén a dolog hasonló módon működik, az óra az értéket mindaddig nem veszi át, amíg az alsó regisztert nem írtuk át.



14. ábra. Írás-olvasás multiplexeléssel

Csak hogy itt nem egyszerűen olvasásról és írásról van szó! A regiszterekbe a CRB (\$DC0F vagy \$DD0F) 7. bitjének állapotától függően óraidőt, illetve riasztási időt is írhatunk. Tehát íráskor a chip megkülönbözteti és külön tárolja az óraidőt és a riasztási időt. Emiatt címenként még egy álregiszterre van szükség, de ez már a chip „belügye”.

A TOD órák regiszterei tehát, amelyek egy adott címen érhetőek el, a CIA-ban háromfelé oszlanak, de még ezenfelül is van egy számláló, amelyből a hardver az óraidőt kiszámítja. Az ilyen többcélú felhasználás, azaz a multiplexelés a takarékoskosságot segíti.

A 16 regiszterhez négy címvezeték szükséges, de mi lenne, ha a TOD óra nem négy, hanem tizenkét regiszterből állna? Akkor egy újabb címbit kellene. Ez persze így egyszerűen hangzik, de a valóságban ez egy 44 lábú chipet, nagyobb áramköri lapot (és bonyolultabbat) és 3 kihasználatlan lábat jelentene. Egy ilyen apróság messzemenő következményekkel jár: több százezer áramkör esetén tonnákban mérhető fémet és műanyagot, költségemelkedést, profitcsökkenést jelent. Egy ilyen apróság megingathatja egy egyébként jól menő cég piaci helyzetét is, ha a konkurencia kihasználja a kihagyott lehetőséget. Végül soron ez már igazán indokolja a takarékoskosságot.

Az időzítők programozása

A C64-esben három szabadon használható 16 bites időzítő található. Azért csak három, mert az operációs rendszer a megszakításhoz használja a CIA1 A időzítőjét. Persze gépi programot írhatunk olyat is, amelyik felfüggeszti az IRQ megszakítást, ezáltal a foglalt időzítő is felszabadulhat. Ezt azonban lehetőleg ne tegyük, mert a megszakítás hiánya meglehetősen megkeseríti a programozó életét.

Az időzítők vezérelhetősége többféle felhasználást is lehetővé tesz:

- μ s-ra kiszámított ütemvezérlés,
- számlálhatunk külső impulzusokat (frekvenciamérés),
- előállíthatunk négyszögjeleket és impulzusokat (frekvenciagenerátor),
- vezérelhetünk külső, digitális alapú egységeket.

Az időzítő a megadott kezdőértéktől (ez 1 és 65535 között változhat) folyamatosan számlál vissza zéróig. Amikor elérte a zéró értéket, újra betölti a kezdőértéket vagy pedig leáll. Mindkét esetben impulzussal jelzi, hogy alulcsordulás történt.

Az A időzítő léptetési ütemének forrása lehet a rendszerütem (Φ_2) vagy külső jel (felfutó CNT élek). A B időzítő vezérlése bonyolultabb: szintén számlálhat a rendszerütem vagy a CNT impulzusai alapján, de emellett számlálhatja az A időzítő alulcsordulásait is. Sőt, lehet még egy szempont: az A időzítő alulcsordulásait számlálja, de csak akkor, ha a CNT magas. Ez a külső, illetve belső jelek számlálására vonatkozik.

Az időzítők segítségével magunk is állíthatunk elő impulzusokat. Ezek az impulzusok a PB6 vagy a PB7 lábon érzékelhetők. A négyszögjelek magas szintjei között tetszőleges hosszúságú alacsony szint állítható be. A magas szint hossza egy rendszerütem. Az időzítők nyújtotta gazdag lehetőségek széles felhasználási skálát biztosítanak külső egységek vezérlésére. Emiatt használható a C64-es pl. CAM (Computer Aided Manufacturing — számítógéppel vezérelt gyártás) feladatok ellátására is, még ha ezek a feladatok csak szerények és egyszerűek lehetnek. Egy jó elektronikai szakember némi fantáziával és több-kevesebb kiegészítő elektronikával képessé teheti a C64-est szalagszerű munkafolyamat ütemezésére.

Az alábbiakban bemutatunk egy példát az időzítők segítségével végzett frekvenciaszámlálásra. Mindkét CIA chipen van számlálóbemenet (CNT1 és CNT2). A bemeneteken megjelenő impulzusokat az időzítők számlálhatják meg. Mivel az időzítők maximális léptetési üteme 985 kHz körüli, ezért a gyakorlatilag még mérhető frekvencia is ennyi kellene hogy legyen. A tapasztalat azonban azt mutatja, hogy a CIA időzítők ennek csak a felét (kb. 400-450 kHz) képesek biztonságosan megszámlálni. Ez olyan gyakorlati korlát, amelyet programozással nem léphetünk át. Ha nagyobb frekvenciákat akarunk mérni, hardveres előosztót kell alkalmaznunk.

A program előosztó nélkül működik, ezért csak kb. 400 kHz frekvenciáig alkalmazható. A mérni kívánt frekvencia jelét a felhasználói (user) port 6-os (CNT2) kivezetésére kell kötnünk. Nagy körültekintéssel járjunk el a mérni kívánt jel kiválasztásánál. A CNT2 vonalra nem lehet 5 V-nál nagyobb feszültségű frekvenciaforrást csatlakoztatni, mert a CIA2 ekkor szinte biztosan tönkremegy.

A BASIC program egy rövid gépi szubrutint használ, mert csak gépi program tudja biztosítani a mérés idejének állandóságát. A mérési idő itt $18 \cdot 65536$ ciklus, kb. 1,16 másodperc. A program mindkét CIA A és B időzítőjét kaszkád üzemmódban működteti (az A időzítő alulcsordulása lépteti a B időzítőt). A CIA1 A időzítője a rendszerütemeket számlálja, a CIA2 A időzítője a CNT2 vonalon megjelenő impulzusokat.

Ahhoz, hogy mindkét CIA A-B időzítőjét egyszerre használhassuk, a megszakításokat fel

kell függeszteni. A CIA2 időzítői mindaddig számlálnak, amíg a CIA1 időzítői le nem futottak. Ha ez megtörtént, a CIA2 A és B időzítőjének regisztereit (\$DD04—DD07) a nulláslap \$FC—FF tárcímeire mentjük. A gépi szubrutin a megszakítások inicializálásával ér véget (\$FDA3).

A BASIC program folyamatosan méri a frekvenciát. A mért érték felfrissítése kb. 1,5 másodpercenként következik be. A frekvenciamérő természetesen nem hitelesített, ezért nem teljesen pontos értéket ad. A hitelesítést pontos és ismert frekvenciaforrás esetén magunk is elvégezhetjük. Ehhez a 220-as sorban levő állandót kell megváltoztatni (0.835202). Ez az érték a rendszerütem és a mérési hossz értékének viszonyából származtatható.

$$0,835202 = \frac{985\,244,4}{18 \cdot 65536}$$

A paraméter megváltoztatására azért lehet szükség, mert a rendszerütem nem minden számítógépben 985 244,4 Hz.

A 180, 230 és 250 sorokban szereplő utasítások a kurzor pozicionálását végzik. A gépi program a SYS 49152 utasítással hívható. A program a 270-es sorában levő GOTO 200 miatt végtelen ciklusban fut, ezért csak a RUN/STOP billentyűvel állítható meg.

```

100 POKE 53280,6:POKE 53281,0
110 GOSUB 1000
120 PRINT CHR$(147);CHR$(30)
130 PRINT SPC(10);"FREKVENCIASZÁMLALO":PRINT:PRINT
140 PRINT "    MAXIMALIS FREKVENCIA KB. 400 KHZ":PRINT
150 PRINT "    BEMENET : AZ USER-PORT CNT2 TUJE":PRINT
160 PRINT "    MAXIMALIS BEMENETI FESZULTSEG : +5V"
170 PRINT:PRINT:PRINT
180 POKE 211,4:POKE 214,15:SYS 58732
190 PRINT "MERESI EREDMENY:"
200 SYS 49152
210 X=PEEK(252):Y=PEEK(253):Z=PEEK(254)
220 F=INT(-.835202*((Z*65536+Y*256+X+1)-2124)*100)/100
230 POKE 211,21:POKE 214,15:SYS 58732
240 PRINT "    "
250 POKE 211,21:POKE 214,15:SYS 58732
260 PRINT F;"    HZ"
270 GOTO 200
1000 FOR I= 49152 TO 49237 :READ X:POKE I,X:S=S+X:NEXT
1010 DATA 120,169,255,162, 18,160,  0,141,  4,221,141,  5
1020 DATA 221,141,  6,221,141,  7,221,141,  4,220,141,  5
1030 DATA 220,142,  6,220,140,  7,220,162, 17,160, 89,169
1040 DATA  81,141, 15,221,169, 49,141, 14,221,142, 14,220
1050 DATA 140, 15,220,173,  6,220,208,251,141, 14,221,141
1060 DATA  15,221,173,  4,221,133,252,173,  5,221,133,253
1070 DATA 173,  6,221,133,254,173,  7,221,133,255, 88, 76
1080 DATA 163,253
1090 IF S <> 11555 THEN PRINT"HIBA A DATASORBAN !":END
1100 RETURN

```


C000		100	*=C000	
C000	78	110	SEI	; A megszakítások letiltása
C001	A9 FF	120	LDA #\$FF	; Az időzítők kezdőértéke
C003	A2 12	130	LDX #\$12	; 18*65536 ciklus - kb. 1.16 sec.
C005	A0 00	140	LDY #\$00	
C007	8D 04 DD	150	STA \$DD04	; A CIA2 időzítőinek beállítása
C00A	8D 05 DD	160	STA \$DD05	
C00D	8D 06 DD	170	STA \$DD06	
C010	8D 07 DD	180	STA \$DD07	
C013	8D 04 DC	190	STA \$DC04	; A CIA1 időzítőinek beállítása
C016	8D 05 DC	200	STA \$DC05	
C019	8E 06 DC	210	STX \$DC06	
C01C	8C 07 DC	220	STY \$DC07	
C01F	A2 11	230	LDX #\$11	; %00010001 - CIA1 A időzítő
C021	A0 59	240	LDY #\$59	; %01011001 - CIA1 B időzítő
C023	A9 51	250	LDA #\$51	; %01010001 - CIA2 B időzítő
C025	8D 0F DD	260	STA \$DD0F	; indítása
C028	A9 31	270	LDA #\$31	; %00110001 - CIA2 A időzítő
C02A	8D 0E DD	280	STA \$DD0E	; indítása
C02D	8E 0E DC	290	STX \$DC0E	; CIA1 A időzítő start
C030	8C 0F DC	300	STY \$DC0F	; CIA1 B időzítő start
C033	AD 06 DC	310	LDA \$DC06	; Kivárni az 1.16 másodpercet
C036	D0 FB	320	BNE AA0	; Még nem futott le, vissza
C038	8D 0E DD	330	STA \$DD0E	; CIA2 A időzítő stop
C03B	8D 0F DD	340	STA \$DD0F	; CIA2 B időzítő stop
C03E	AD 04 DD	350	LDA \$DD04	; A megszámlált
C041	85 FC	360	STA \$FC	
C043	AD 05 DD	370	LDA \$DD05	; impulzusok áttöltése
C046	85 FD	380	STA \$FD	
C048	AD 06 DD	390	LDA \$DD06	; a nulláslapra
C04B	85 FE	400	STA \$FE	
C04D	AD 07 DD	410	LDA \$DD07	
C050	85 FF	420	STA \$FF	
C052	58	430	CLI	; A megszakítások engedélyezése
C053	4C A3 FD	440	JMP \$FDA3	; A CIA-K inicializálása
C056		450	.END	

ZEILEN:36 SYMBOLE:1 FEHLER:0

AA0 =C033

A C64-eshez még két *kiegészítő CIA 6526-os chipet* is csatlakoztathatunk. A processzor a memória \$DE00 és \$DF00 címeken kezdődő I/O₁ és az I/O₂ címterületeken érheti el a választható CIA chipet. A chipet tartalmazó kártyát a bővítőportba kell helyezni. Ezek a CIA chipok további vezérlési feladatokat ellátását teszik lehetővé. E chipok pontosan úgy kezelhetők, mint a gépbe építettek, de ezeknek egyetlen regiszterük sem foglalt.

Az AM/PM óra

A CIA beépített TOD (valósídejű) órája úgy működik, mint a karunkon viselt digitális óra. Az AM/PM megnevezés az angol nyelvterületen szokásos délelőtt/délután jelzés megfelelője. AM = ante meridiem, PM = post meridiem. A TOD óra négy regiszterből áll. Ezek sorban: a tizedmásodpercek, a másodpercek, a percek és az órák regiszterei. A regiszterek — mint a multiplexelésnél láttuk — több belső regiszterre bonthatók. Feladatuk kettős: belőlük olvasható ki az aktuális óraidő, de ezekben tárolható a riasztási idő is. A riasztási idő nem más, mint egy időpont. Amikor a TOD óra eléri ezt az időpontot, a CIA a processzortól

megszakítást kér. Megfelelő figyelőrutinnal biztosítható, hogy az időpontok egyezése kívülről is észrevehető legyen (hang- és/vagy képjelzés).

Az óra bármikor megállítható és újraindítható. Vezérlőimpulzusait nem a kvarcstabilizált belső ütemjelről kapja, hanem a hálózati (220 V) frekvenciáról. Ezzel a megoldással hosszú távon kielégítő pontosságot lehet elérni, mivel a hálózati frekvencia jó közelítéssel 50 Hz.

Az óraregisztereket normál tárhelyekként olvashatjuk ki. Amikor az óra legfelső regiszterét olvassuk, az olvasható regiszterekben az óra pillanatnyi állapota rögzül, és mindaddig nem változik, amíg a tizedmásodperceket is ki nem olvastuk. Az óra közben természetesen továbbra is jár. Írásnál mindez hasonló: az óra mindaddig nem veszi át az új értéket, amíg a tizedmásodperceket is felül nem írtuk. Amikor az óraidőn kívánunk változtatni, a CIA 15. regiszterének (\$DC0F vagy \$DD0F) 7. bitje alacsony kell legyen. Ha ez a bit magas, az új értéket a CIA a riasztás időpontjaként fogja tárolni.

A regiszterek írásánál és olvasásánál fontos tudni, hogy a regiszterek BCD kódban tárolják az időegységek értékeit. A tizedmásodperceket tároló regiszter alsó, a perceket és másodperceket tároló regiszterek alsó és felső félbyte-ja foglalt. Az óraregiszter alsó félbyte-ja teljesen, felső félbyte-ja csak részben foglalt. Ennek a regiszternek a 7. bitje mutatja meg, hogy a nap melyik szakát jelzi ki az óra. Ha a bit magas, az óra szerint délután van. Lássunk néhány példát az idő ábrázolására!

Délelőtt 9 óra 43 perc 56 másodperc:

\$Dx0B	TODHRS	0 0 0 0	1 0 0 1	\$09
\$Dx0A	TODMIN	0 1 0 0	0 0 1 1	\$43
\$DX09	TODSEC	0 1 0 1	0 1 1 0	\$56

Délután 6 óra 12 perc 24 másodperc:

\$Dx0B	TODHRS	1 0 0 0	0 1 1 0	\$86
\$Dx0A	TODMIN	0 0 0 1	0 0 1 0	\$12
\$Dx09	TODSEC	0 0 1 0	0 1 0 0	\$24

A TOD óra felhasználható ahhoz, hogy a hosszasan önállóan dolgozó programok futása közben a képernyő mozdulatlanságát megtörjük. Következő programunk erre mutat példát: az idő múlását karakteres módban jelezzük a képernyőn. A kezdőpozíciót (sor, oszlop) a program tőlünk várja. Mivel az órát az IRQ rutin írja ki a képernyőre, ezért mellette nyugodtan futhat bármilyen program.

A gépi szubrutint a SYS 49152, „120000”, X,Y utasítással indíthatjuk el. Az óra a 12 óra 0 perc 0 másodperc értékről indul. A képernyőn az X. sorban az Y. oszloptól kezdődően a következő sor lesz látható:

12 : 00 : 00 : 0

A gépi program ellenőrzi a paramétereket. Ha azok valamilyen oknál fogva hibásak, ?ILLEGAL QUANTITY ERROR hibaüzenetet fogunk kapni. Ilyenkor felül kell vizsgálni,

hogy a paraméterek értékei helyesek-e. A sor és az oszlop megadása csak az első indításkor kötelező, a program ezután minden indításkor az eredetileg megadott helyre teszi a kijelzést.

AD9E		140	FRMEVL	=\$AD9E	/ a Kifejezés Kiértékelése
0079		240	CHRGOT	=\$0079	/ az utolsó Karakter újraolvasása
AEFD		250	CHKCOM	=\$AEFD	/ vessző ellenőrzése
B7F1		260	CHKGET	=\$B7F1	/ a paraméter beolvasása
B6A3		270	FRESTR	=\$B6A3	/ szövegkezelés
DC0E		360	CRA1	=\$DC0E	/ CIA1 A vezérlőregiszter
DC0F		370	CRB1	=CRA1+1	/ CIA1 B vezérlőregiszter
DC08		380	TODTHS	=\$DC08	/ CIA1 TOD tizedmásodpercek
DC09		390	TODSEC	=TODTHS+1	/ CIA1 TOD másodpercek
DC0A		400	TODMIN	=TODTHS+2	/ CIA1 TOD percek
DC0B		410	TODHRS	=TODTHS+3	/ CIA1 TOD órák
C000		430		*\$C000	
C000	AD 0E DC	440	TIME	LDA CRA1	/ az ütemlépték beállítása
C003	09 80	450		ORA #\$80	/ a lépték 50 Hz
C005	8D 0E DC	460		STA CRA1	
C008	AD 0F DC	470		LDA CRB1	/ a B vezérlőregiszter beállítása
C00B	29 7F	480		AND #\$7F	/ az óraidő beállítása következik
C00D	8D 0F DC	490		STA CRB1	
C010	20 79 00	500		JSR CHRGOT	/ van még Karakter ?
C013	F0 70	510		BEQ BA4	/ nincs, Kikapcsolni a rutint
C015	20 FD AE	520		JSR CHKCOM	/ vessző ellenőrzése
C018	20 9E AD	530		JSR FRMEVL	/ a Kifejezés Kiértékelése
C01B	20 A3 B6	540		JSR FRESTR	/ a Kifejezés hossza
C01E	C9 06	550		CMP #\$06	/ 6-tal egyenlő ?
C020	D0 69	560		BNE ILLQ	/ nem, hiba
C022	A0 FF	570		LDY #\$FF	/ Kezdőpozíció
C024	20 A0 C0	580		JSR BA5	/ a Következő Karakter beolvasása
C027	C9 03	590		CMP #\$03	/ 2-nél nagyobb ?
C029	20 A7 C0	600		JSR BA7	/ ellenőrzés és feldolgozás
C02C	20 A0 C0	610		JSR BA5	/ a Következő Karakter beolvasása
C02F	C9 0A	620		CMP #\$0A	/ 9-nél nagyobb ?
C031	B0 58	630		BCS ILLQ	/ igen, hiba
C033	05 FB	640		ORA \$FB	/ a felső félbyte átvétele
C035	D0 04	650		BNE BA0	/ nem 0 óra, ugrás
C037	A9 92	660		LDA #\$92	/ 0 óra helyettesítése
C039	D0 0F	670		BNE BA1	/ feltétel nélküli ugrás
C03B	C9 24	680	BA0	CMP #\$24	/ nagyobb, mint 23 ?
C03D	B0 4C	690		BCS ILLQ	/ igen, hiba
C03F	C9 13	700		CMP #\$13	/ nagyobb, mint 12 ?
C041	90 07	710		BCC BA1	/ igen, ugrás
C043	38	720		SEC	/ túlcsoordulás törölve
C044	F8	730		SED	/ BCD-mód beállítva
C045	E9 12	740		SBC #\$12	/ 12 levonása
C047	D8	750		CLD	/ BCD-mód törölve
C048	09 80	760		ORA #\$80	/ a 7. bit beállítása
C04A	8D 0B DC	770	BA1	STA TODHRS	/ az óraidő regiszter beállítása
C04D	20 8E C0	780		JSR BA6	/ a percek beolvasása
C050	8D 0A DC	790		STA TODMIN	/ a perc regiszter beállítása
C053	20 8E C0	800		JSR BA6	/ a másodpercek beolvasása
C056	8D 09 DC	810		STA TODSEC	/ a másodperc regiszter beállítása
C059	A9 00	820		LDA #\$00	/ a tizedmásodperc 0
C05B	8D 08 DC	830		STA TODTHS	/ a regiszter beállítása, az óra indul
C05E	20 79 00	840		JSR CHRGOT	/ van még Karakter ?
C061	F0 15	850		BEQ BA2	/ nincs, ugrás
C063	20 F1 B7	860		JSR CHKGET	/ vessző ellenőrzése, a byte beolvasása
C066	E0 18	870		CPX #\$18	/ nagyobb, mint 24 ?
C068	B0 21	880		BCS ILLQ	/ igen, hiba
C06A	86 FD	890		STX \$FD	/ a Koordináta tárolása
C06C	20 F1 B7	900		JSR CHKGET	/ vessző ellenőrzése, a byte beolvasása
C06F	E0 1F	910		CPX #31	/ nagyobb, mint 30 ?
C071	B0 18	920		BCS ILLQ	/ igen, hiba
C073	86 FE	930		STX \$FE	/ a Koordináta tárolása
C075	20 B0 C0	940		JSR BB5	/ a mutató Kiszámítása
C078	A9 E1	950	BA2	LDA #TIMIRQ<	/ a megszakítás rutin új címe
C07A	A0 C0	960		LDY #TIMIRQ>	

C07C	78			970	BA3	SEI		; a megszakítások tiltása
C07D	8D	14	03	980		STA	\$0314	; a megszakítás-vektor
C080	8C	15	03	990		STY	\$0315	; beállítása
C083	58			1000		CLI		; a megszakítások engedélyezése
C084	60			1010		RTS		
C085	A9	31		1020	BA4	LDA	#\$31	; az eredeti
C087	A0	EA		1030		LDY	#\$EA	; vektor
C089	D0	F1		1040		BNE	BA3	; feltétel nélküli ugrás
C08B	4C	48	B2	1050	ILLQ	JMP	\$B248	; ILLEGAL QUANTITY ERROR
C08E	20	A0	C0	1060	BA6	JSR	BA5	; átalakítás ASCII-ról
C091	C9	06		1070		CMP	#\$06	; nagyobb, mint 5 ?
C093	20	A7	C0	1080		JSR	BA7	; ellenőrzés és feldolgozás
C096	20	A0	C0	1090		JSR	BA5	; a következő karakter feldolgozása
C099	C9	0A		1100		CMP	#\$0A	; nagyobb, mint 9 ?
C09B	B0	EE		1110		BCS	ILLQ	; igen, hiba
C09D	05	FB		1120		ORA	\$FB	; a felső félbyte átvétele
C09F	60			1130		RTS		
C0A0	C8			1140	BA5	INY		; átalakítás ASCII kódról
C0A1	B1	22		1150		LDA	(\$22),Y	; a beolvasott byte-ból
C0A3	38			1160		SEC		
C0A4	E9	30		1170		SBC	#\$30	; 48 levonása
C0A6	60			1180		RTS		
C0A7	B0	E2		1190	BA7	BCS	ILLQ	; nagyobb, mint a megadott határ, hiba
C0A9	0A			1200		ASL	A	
C0AA	0A			1210		ASL	A	; az alsó félbyte
C0AB	0A			1220		ASL	A	
C0AC	0A			1230		ASL	A	; feltolása a felsőbe
C0AD	85	FB		1240		STA	\$FB	; és átmeneti tárolása
C0AF	60			1250		RTS		
C0B0	AD	88	02	1260	BB5	LDA	\$0288	; a képernyőmemória kezdőcíme
C0B3	8D	A8	02	1270		STA	\$02A8	; tárolva
C0B6	A2	00		1280		LDX	#\$00	; az alsó byte 0
C0B8	8E	A7	02	1290		STX	\$02A7	
C0BB	A5	FD		1300		LDA	\$FD	; a sorok száma
C0BD	F0	13		1310		BEQ	BB6	; nulla, a számítást átugrani
C0BF	AD	A7	02	1320	BB3	LDA	\$02A7	; az alsó byte
C0C2	18			1330		CLC		
C0C3	69	28		1340		ADC	#40	; egy sor hossza, 40 hozzáadása
C0C5	8D	A7	02	1350		STA	\$02A7	; az alsó byte új értéke
C0C8	90	03		1360		BCC	BB2	; nincs túlcsordulás, ugrás
C0CA	EE	A8	02	1370		INC	\$02A8	
C0CD	E8			1380	BB2	INX		; a számláló
C0CE	E4	FD		1390		CPX	\$FD	; egyenlő a sorok számával ?
C0D0	D0	ED		1400		BNE	BB3	; még nem, vissza
C0D2	AD	A7	02	1410	BB6	LDA	\$02A7	; az alsó byte
C0D5	18			1420		CLC		
C0D6	65	FE		1430		ADC	\$FE	; az oszlopok számának hozzáadása
C0D8	8D	A7	02	1440		STA	\$02A7	; az új érték tárolása
C0DB	90	03		1450		BCC	BB4	; nincs túlcsordulás, ugrás
C0DD	EE	A8	02	1460		INC	\$02A8	
C0E0	60			1470	BB4	RTS		
C0E1	A5	FB		1480	TIMIRQ	LDA	\$FB	; a felhasznált regiszterek
C0E3	48			1490		PHA		
C0E4	A5	FC		1500		LDA	\$FC	; elmentése a verembe
C0E6	48			1510		PHA		
C0E7	AD	A7	02	1520		LDA	\$02A7	; a kezdőcím
C0EA	AC	A8	02	1530		LDY	\$02A8	
C0ED	85	FB		1540		STA	\$FB	; a nulláslapra
C0EF	84	FC		1550		STY	\$FC	
C0F1	A0	00		1560		LDY	#\$00	
C0F3	AD	0B	DC	1570		LDA	TODHRS	; az óráidő regiszter
C0F6	C9	12		1580		CMP	#\$12	; 12-vel egyenlő ?
C0F8	F0	11		1590		BEQ	BA8	; igen, ugrás
C0FA	C9	80		1600		CMP	#\$80	; délután ?

C0FC	90	0F	1610	BCC	BA9	; nem, ugrás
C0FE	29	7F	1620	AND	#\$7F	; a 7. bit levágása
C100	C9	12	1630	CMP	#\$12	; 24 óra ?
C102	F0	09	1640	BEQ	BA9	; igen
C104	F8		1650	SED		; BCD-mód beállítva
C105	18		1660	CLC		; a túlcscordulás törölve
C106	69	12	1670	ADC	#\$12	; 12 hozzáadása a számjegyhez
C108	D8		1680	CLD		; BCD-mód törölve
C109	D0	02	1690	BNE	BA9	; feltétel nélküli ugrás
C10B	A9	00	1700	LDA	#\$00	
C10D	20	2B	1710	JSR	BB0	; Kiírás
C110	AD	0A	1720	LDA	TODMIN	; percek
C113	20	2B	1730	JSR	BB0	; Kiírás
C116	AD	09	1740	LDA	TODSEC	; másodpercek
C119	20	2B	1750	JSR	BB0	; Kiírás
C11C	AD	08	1760	LDA	TODTHS	; tizedmásodpercek
C11F	20	3D	1770	JSR	BB1	; Kiírás
C122	68		1780	PLA		; a regiszterek
C123	85	FC	1790	STA	\$FC	
C125	68		1800	PLA		; visszatöltése a veremből
C126	85	FB	1810	STA	\$FB	
C128	4C	31	1820	JMP	\$EA31	; az eredeti IRQ-ra
C12B	48		1830	PHA		; a byte elmentése
C12C	29	F0	1840	AND	#\$F0	; az alsó négy bit levágása
C12E	4A		1850	LSR	A	
C12F	4A		1860	LSR	A	; a felső félbyte
C130	4A		1870	LSR	A	
C131	4A		1880	LSR	A	; áttolása az alsóba
C132	20	3D	1890	JSR	BB1	; Kiírás
C135	68		1900	PLA		; a byte visszatöltése
C136	29	0F	1910	AND	#\$0F	; a felső félbyte levágása
C138	20	3D	1920	JSR	BB1	; Kiírás
C13B	A9	0A	1930	LDA	#\$0A	; Kettőspont
C13D	09	30	1940	ORA	#\$30	; átalakítás KépernyőKódra
C13F	91	FB	1950	STA	(\$FB),Y	; a byte elhelyezése a Képernyőn
C141	C8		1960	INY		; mutató növelése
C142	60		1970	RTS		
C143			1980	.END		

ZEILEN:167 SYMBOLE:31 FEHLER:0

BA0	=C03B	BA1	=C04A	BA2	=C078	BA3	=C07C	BA4	=C085	BA5	=C0A0
BA6	=C08E	BA7	=C0A7	BA8	=C10B	BA9	=C10D	BB0	=C12B	BB1	=C13D
BB2	=C0CD	BB3	=C0BF	BB4	=C0E0	BB5	=C0B0	BB6	=C0D2	CHKCOM	=AEFD
CHKGET	=B7F1	CHRGOT	=0079	CRA1	=DC0E	CRB1	=DC0F	FRESTR	=B6A3	FRMEVL	=AD9E
ILLQ	=C08B	TIME	=C000	TIMIRQ	=C0E1	TODHRS	=DC0B	TODMIN	=DC0A	TODSEC	=DC09
TODTHS	=DC08										

100 IF PEEK(49152)<>173 OR PEEK(49324)<>10 THEN GOSUB 180

110 PRINT CHR\$(147)

120 SYS 49152,"235956",12,15

130 FOR I=1 TO 3000: NEXT I

140 SYS 49152

150 FOR I=1 TO 1000: NEXT I

160 PRINT CHR\$(147)

170 END

180 FOR I= 49152 TO 49474 :READ X:POKE I,X:S=S+X:NEXT

190 DATA 173, 14,220, 9,128,141, 14,220,173, 15,220, 41

200 DATA 127,141, 15,220, 32,121, 0,240,112, 32,253,174

210 DATA 32,158,173, 32,163,182,201, 6,208,105,160,255

220 DATA 32,160,192,201, 3, 32,167,192, 32,160,192,201

230 DATA 10,176, 88, 5,251,208, 4,169,146,208, 15,201

240 DATA 36,176, 76,201, 19,144, 7, 56,248,233, 18,216

250 DATA 9,128,141, 11,220, 32,142,192,141, 10,220, 32

260 DATA 142,192,141, 9,220,169, 0,141, 8,220, 32,121

270 DATA 0,240, 21, 32,241,183,224, 24,176, 33,134,253

```

280 DATA 32,241,183,224, 31,176, 24,134,254, 32,176,192
290 DATA 169,225,160,192,120,141, 20, 3,140, 21, 3, 88
300 DATA 96,169, 49,160,234,208,241, 76, 72,178, 32,160
310 DATA 192,201, 6, 32,167,192, 32,160,192,201, 10,176
320 DATA 238, 5,251, 96,200,177, 34, 56,233, 48, 96,176
330 DATA 226, 10, 10, 10, 10,133,251, 96,173,136, 2,141
340 DATA 168, 2,162, 0,142,167, 2,165,253,240, 19,173
350 DATA 167, 2, 24,105, 40,141,167, 2,144, 3,238,168
360 DATA 2,232,228,253,208,237,173,167, 2, 24,101,254
370 DATA 141,167, 2,144, 3,238,168, 2, 96,165,251, 72
380 DATA 165,252, 72,173,167, 2,172,168, 2,133,251,132
390 DATA 252,160, 0,173, 11,220,201, 18,240, 17,201,128
400 DATA 144, 15, 41,127,201, 18,240, 9,248, 24,105, 18
410 DATA 216,208, 2,169, 0, 32, 43,193,173, 10,220, 32
420 DATA 43,193,173, 9,220, 32, 43,193,173, 8,220, 32
430 DATA 61,193,104,133,252,104,133,251, 76, 49,234, 72
440 DATA 41,240, 74, 74, 74, 74, 32, 61,193,104, 41, 15
450 DATA 32, 61,193,169, 10, 9, 48,145,251,200, 96
460 IF S <> 39304 THEN PRINT"HIBA A DATASORBAN !":END
470 RETURN

```

A soros léptetőregiszter olyan 8 bites regiszter, amelybe az adatok egyenként érkeznek, majd egyenként távoznak. A regisztert a processzor normál tárhelyként kezeli, azaz olvasó- és íróműveletei is 8 bitesek. Ha a léptetőregiszter kimenet, a tárolt bitek az A időzítő alulcsordulásakor egyenként az SP lábra kerülnek. Ha a regiszter bemenet, az SP lábón megjelent jelszintnek megfelelő bit akkor kerül a regiszterbe, amikor a külső egység a CNT bemeneten magas szintet hoz létre. A CNT impulzus hatására a regiszter tartalma eggyel balra lép. A nyolcadik impulzus után az ICR (\$Dx0D) 3. bitje magasra vált, azt jelezve, hogy az SDR (\$Dx0C) megtelt, a processzor olvashatja a byte-ot. Ha a folyamatot megfordítjuk, a bitek érvényességét az jelzi, hogy az A időzítő alulcsordult. A nyolcadik impulzus után az ICR (\$Dx0D) 3. bitje magas lesz, most azt jelezve, hogy az SDR kiürült. A regiszter segítségével a soros adatátvitel minden különösebb hardver nélkül megvalósítható.

Ha a C64-esen létrehozott adatállományt Amigán szeretnénk felhasználni, akkor az adatokat háttértárolóra mentjük, és az Amigával onnan olvassuk be. De mit tegyünk akkor, ha a másik számítógép nem Amiga, és sehogyan sem képes a C64-es háttértárolójáról olvasni?

Ekkor meg kell oldani a közvetlen kapcsolat létrehozásakor felmerülő problémákat. Szerencsére a C64-es hardvere egyszerű megoldást kínál. A két CIA chip mindegyike rendelkezik olyan vezetékekkel (kettővel), amelyek adatok fogadását és küldését vezérelhetik. Az angol terminológia ezekre a handshake (kézfogó) elnevezést használja.

Ez a két vonal — \overline{PC} és \overline{FLAG} — két számítógép közötti adatforgalom összehangolásában pl. úgy vehet részt, hogy a \overline{FLAG} -re érkező alacsony szintű jel a B porton (\$DD01) olvasható adatok érvényességét jelzi. Ha külső egység elérte a B portot, azaz az ott jelentkező jeleket értelmezte és tárolta, a \overline{PC} vonalon egy alacsony impulzus formájában nyugtázhatja, hogy az adatokat átvette.

Ezzel a megoldással közvetlenül össze lehet kötni két C64-es számítógépet is, de a másik fél minden olyan számítógép lehet, amely a \overline{PC} és \overline{FLAG} vonalaknak megfelelő vezérlővezeték tud biztosítani. Bár mindkét CIA rendelkezik az említett vonalakkal, csak a CIA2 vonalai vezetnek a felhasználói portra, ezért a kapcsolat csak CIA2 programozásával képzelhető el.

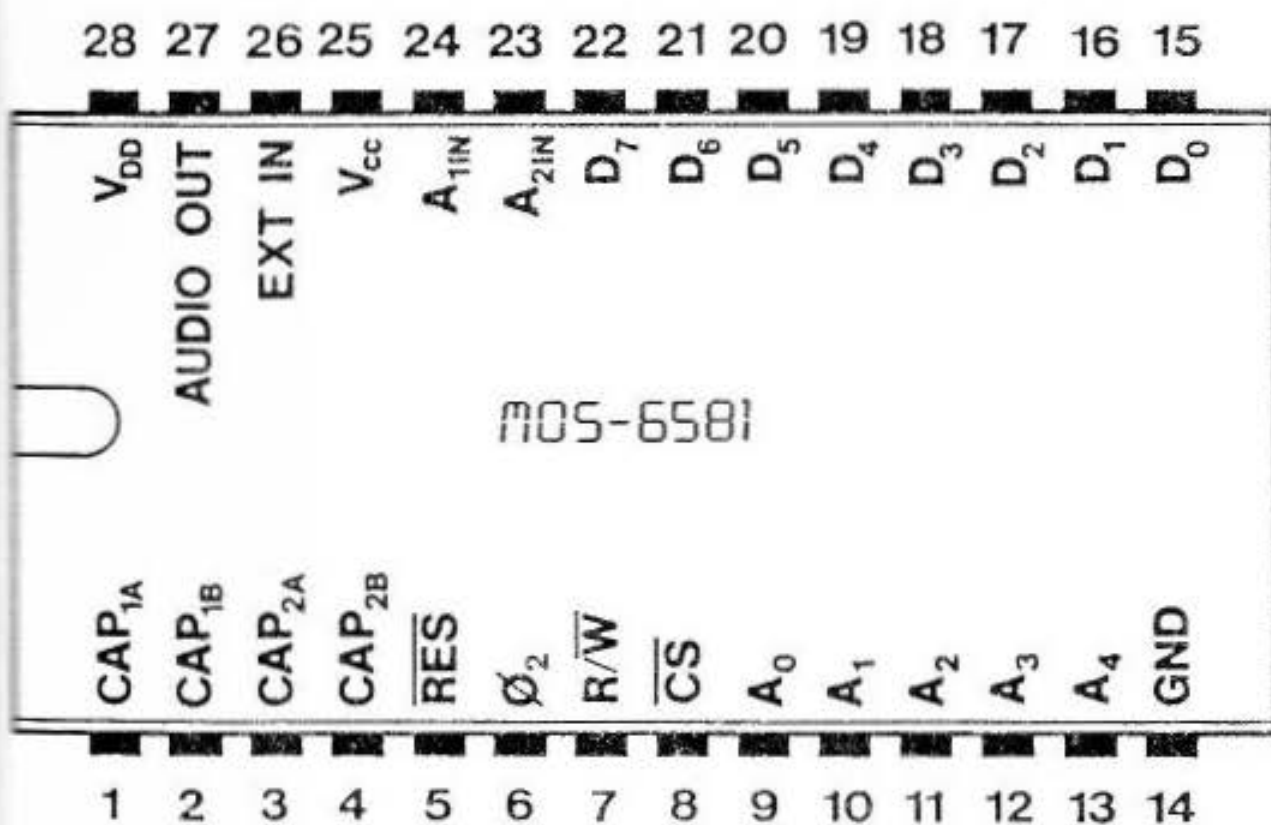
1.4. A hanggenerátor (SID)

A SID (Sound Interface Device) a chipkészlet olyan eleme, amely a rendszer bekapcsolás utáni működésében még nem vesz részt. De amint hangjelzésre van szükség, az előállításához szükséges összes feladatot magára vállalja.

A SID integrált áramköre magában foglal:

- három külön-külön vezérelhető oszcillátort (szólamot);
- szólamonként három keverhető szűrőt;
- szólamonként burkológörbe-generátort;
- két, kaszkádmódon kapcsolható gyűrűs modulátort, ami a hangképzést végzi, valamint
- két független, 8 bites A/D átalakítót.

A chip egy 28 lábú DIL-tokban kerül forgalomba.



15. ábra. A SID lábkiosztása

A hanggenerátor kivezetéseinek jelentése:

- 1—2. CAP_{1A}-CAP_{1B}: szűrőkondenzátor-csatlakozás
- 3—4. CAP_{2A}-CAP_{2B}: szűrőkondenzátor-csatlakozás
5. RES: Reset bemenet, alacsony állapota inicializálja a chipet
6. Φ₂: ütembemenet, a processzor órajele
7. R/W: Read/Write választó, 0: íráshozzáférés, 1: olvasás
8. CS: Chip Select, 0: a chip regiszterei elérhetők, ha Φ₂ alacsony
- 9—13. A₀-A₄: a regiszterek címbitjei
14. GND: Ground, rendszerföld
- 15—22. D₀-D₇: adatbusz
23. A_{2IN}: POTY, az egyik analóg-digitális átalakító bemenete
24. A_{1IN}: POTX, a másik A/D átalakító bemenete
25. VCC: tápfeszültség, +5 V

- 26. EXT IN: External input, külső hangforrás jelének fogadása
- 27. AUDIO OUT: a SID hangfrekvenciás jeleinek kimenete
- 28. VDD: tápfeszültség, +12 V

1.4.1. A SID regiszterei

A regiszterek jelentős része csak írható, ezeket olvasni nem lehet. Van azonban néhány regiszter, amelyek csak olvashatók, a regiszterekbe irányuló írásműveletet a SID nem hajtja végre.

A SID 29 címezhető belső regisztere négy csoportba sorolható: három csoportot tesznek ki azok a regiszterek, amelyek az egyes szólamok vezérlését végzik. A negyedik csoport vegyes: olyan regiszterek is előfordulnak, amelyeknek nincs közükhöz a hangképzéshez, illetve olyanok, amelyek mindhárom szólamhoz tartoznak. Az alábbi felsorolás tagolása ennek megfelelő. (A regiszterek tárcímét a sorszám után zárójelben adtuk meg.)

1. szólam

- 0. (\$D400) a frekvencia alsó byte-ja
- 1. (\$D401) a frekvencia felső byte-ja
- 2. (\$D402) a négyszög hullámforma kitöltési tényezője, alsó byte
- 3. (\$D403)
 - 0—3. bit: a négyszög hullámforma kitöltési tényezője, felső bitek
 - 4—7. bit: nem használt
- 4. (\$D404) vezérlőregiszter
 - 0. bit: KEY, a burkológörbe-generátor vezérlőbitje
 - 1. bit: SYNC, szinkronizáló vezérlőbit
 - 2. bit: RING, a gyűrűsmódulátor kapcsolóbitje
 - 3. bit: TEST, ha magas, a hang nem hallatszik
 - 4. bit: TRI, Triangle, 1 = a hullámforma háromszög
 - 5. bit: SAW, Sawthorne, 1 = fűrészfog hullámforma
 - 6. bit: PUL, Pulse, 1 = négyszög hullámforma
 - 7. bit: NSE, Noise, 1 = fehérzaj
- 5. (\$D405) burkológörbe-generátor vezérlőbyte-ja, felfutás-lecsengés (Attack-Decay, AD)
 - 0—3. bit: lecsengés; 6 ms — 24 s
 - 4—7. bit: felfutás; 2 ms — 8 s
- 6. (\$D406) a burkológörbe-generátor vezérlőbyte-ja, tartás-elengedés (Sustain-Release, SR)
 - 0—3. bit: elengedés; 6 ms — 24 s
 - 4—7. bit: tartás; hosszát a KEY bit vezérli, értéke relatív hangerősséget határoz meg a maximális hangerőhöz viszonyítva

2. szólam

- 7. (\$D407) a frekvencia alsó byte-ja
- 8. (\$D408) a frekvencia felső byte-ja
- 9. (\$D409) a négyszög hullámforma kitöltési tényezője, alsó byte
- 10. (\$D40A)
 - 0—3. bit: a négyszög hullámforma kitöltési tényezője, felső bitek
 - 4—7. bit: nem használt
- 11. (\$D40B) vezérlőregiszter. Bitjeinek kiosztása és funkciója azonos az 1. szólam hasonló regiszterének jellemzőivel
- 12. (\$D40C) a burkológörbe-generátor vezérlőbyte-ja, felfutás-lecsengés
- 13. (\$D40D) a burkológörbe-generátor vezérlőbyte-ja, tartás-elengedés

3. szólam

- 14. (\$D40E) a frekvencia alsó byte-ja
- 15. (\$D40F) a frekvencia felső byte-ja
- 16. (\$D410) a négyszög hullámforma kitöltési tényezője, alsó byte
- 17. (\$D411)
 - 0—3. bit: a négyszög hullámforma kitöltési tényezője, felső bitek
 - 4—7. bit: nem használt
- 18. (\$D412) vezérlőregiszter. Bitjeinek kiosztása és funkciója azonos az 1. szólam hasonló regiszterének jellemzőivel
- 19. (\$D413) a burkológörbe-generátor vezérlőbyte-ja, felfutás-lecsengés
- 20. (\$D414) a burkológörbe-generátor vezérlőbyte-ja, tartás-elengedés

Közös regiszterek

- 21. (\$D415) a szűrőfrekvencia értéke
 - 0—2. bit: alsó bitek (0—2)
 - 3—7. bit: nem használt
- 22. (\$D416) a szűrőfrekvencia értéke, felső bitek (3—10)
- 23. (\$D417) szűrőrezonancia és szűrőkapcsoló
 - 0. bit: 1 = az 1. szólam szűrése
 - 1. bit: 1 = a 2. szólam szűrése
 - 2. bit: 1 = a 3. szólam szűrése
 - 3. bit: 1 = külső hangforrás szűrése
 - 4—7. bit: a szűrő rezonanciafrekvenciája
- 24. (\$D418) hangerő és szűrőválasztó kapcsoló
 - 0—3. bit: összhangerő (0—15)
 - 4. bit: 1 = aluláteresztő szűrő, meredekség: 12 dB/oktáv
 - 5. bit: 1 = sávszűrő, meredekség: 6 dB/oktáv
 - 6. bit: 1 = feluláteresztő szűrő, meredekség: 12 dB/oktáv
 - 7. bit: 1 = a 3. szólam nem hallható

4. táblázat. A hangskála

Oktáv	Hang	Frekvencia, Hz	Paraméter	Alsó	Felső byte
0	0 C	16.4	268	12	1
	1 C#	17.3	284	28	1
	2 D	18.4	301	45	1
	3 D#	19.4	318	62	1
	4 E	20.6	337	71	1
	5 F	21.8	358	102	1
	6 F#	23.1	379	123	1
	7 G	24.5	401	145	1
	8 G#	26.0	425	169	1
	9 A	27.5	451	195	1
	10 A#	29.1	477	221	1
	11 H	30.9	506	250	1
1	0 C	32.7	536	24	2
	1 C#	34.6	568	56	2
	2 D	36.7	602	90	2
	3 D#	38.9	637	125	2
	4 E	41.2	675	163	2
	5 F	43.7	716	204	2
	6 F#	46.2	758	246	2
	7 G	49.0	803	35	3
	8 G#	51.9	851	83	3
	9 A	55.0	902	134	3
	10 A#	58.3	955	187	3
	11 H	61.7	1012	244	3
2	0 C	65.4	1072	48	4
	1 C#	69.3	1136	112	4
	2 D	73.4	1204	180	4
	3 D#	77.8	1275	251	4
	4 E	82.4	1351	71	5
	5 F	87.3	1432	152	5
	6 F#	92.5	1517	237	5
	7 G	98.0	1607	71	6
	8 G#	103.8	1703	167	6
	9 A	110.0	1804	12	7
	10 A#	116.5	1911	119	7
	11 H	123.5	2025	233	7
3	0 C	130.8	2145	97	8
	1 C#	138.6	2273	225	8
	2 D	146.8	2408	104	9
	3 D#	155.6	2551	247	9
	4 E	164.8	2703	143	10
	5 F	174.6	2864	48	11
	6 F#	185.0	3034	218	11
	7 G	196.0	3215	143	12
	8 G#	207.7	3406	78	13
	9 A	220.0	3608	24	14
	10 A#	233.1	3823	239	14
	11 H	246.9	4050	210	15

Az eddig felsorolt regisztereket csak írásra érheti el a processzor, az alábbiakat pedig csak olvasásra.

- 25. (\$D419) analóg/digitális átalakító — Paddle 1
- 26. (\$D41A) analóg/digitális átalakító — Paddle 2
- 27. (\$D41B) a 3. oszcillátor (szólam) frekvenciájának felső byte-ja
- 28. (\$D41C) a 3. szólam burkológörbe-generátorának felső byte-ja

Oktáv	Hang	Frekvencia, Hz	Paraméter	Alsó	Felső byte
4	0	C	261.6	4291	195 16
	1	C#	277.2	4547	195 17
	2	D	293.7	4817	203 18
	3	D#	311.1	5103	239 19
	4	E	329.6	5407	31 21
	5	F	349.2	5728	96 22
	6	F#	370.0	6069	181 23
	7	G	392.0	6430	30 25
	8	G#	415.3	6812	156 26
	9	A	440.0	7217	49 28
	10	A#	466.2	7647	223 29
	11	H	493.9	8101	165 31
5	0	C	523.2	8583	135 33
	1	C#	554.4	9094	134 35
	2	D	587.3	9634	162 37
	3	D#	622.3	10207	223 39
	4	E	659.3	10814	62 42
	5	F	698.5	11457	193 44
	6	F#	740.0	12139	107 47
	7	G	784.0	12860	60 50
	8	G#	830.6	13625	57 53
	9	A	880.0	14435	99 56
	10	A#	932.3	15294	190 59
	11	H	987.8	16203	75 63
6	0	C	1046.5	17167	15 67
	1	C#	1108.7	18188	12 71
	2	D	1174.7	19269	69 75
	3	D#	1244.5	20415	191 79
	4	E	1318.5	21629	125 84
	5	F	1396.9	22915	131 89
	6	F#	1480.0	24278	214 94
	7	G	1568.0	25721	121 100
	8	G#	1661.2	27251	115 106
	9	A	1760.0	28871	199 112
	10	A#	1864.7	30588	124 119
	11	H	1975.5	32407	151 126
7	0	C	2093.0	34334	30 134
	1	C#	2217.5	36376	24 142
	2	D	2349.3	38539	139 150
	3	D#	2489.0	40830	126 159
	4	E	2637.0	43258	250 168
	5	F	2793.8	45830	6 179
	6	F#	2959.9	48556	172 189
	7	G	3136.0	51443	243 200
	8	G#	3322.4	54502	230 212
	9	A	3520.0	57743	143 225
	10	A#	3729.3	61176	248 238
	11	H	3951.1	64814	46 253

1.4.2. A SID programozása

Az oszcillátor frekvenciáját a szólamhoz tartozó első két regiszter adja meg. A regiszterpár tartalma és a frekvencia értéke közötti összefüggés:

$$F = 0,06097 * R$$

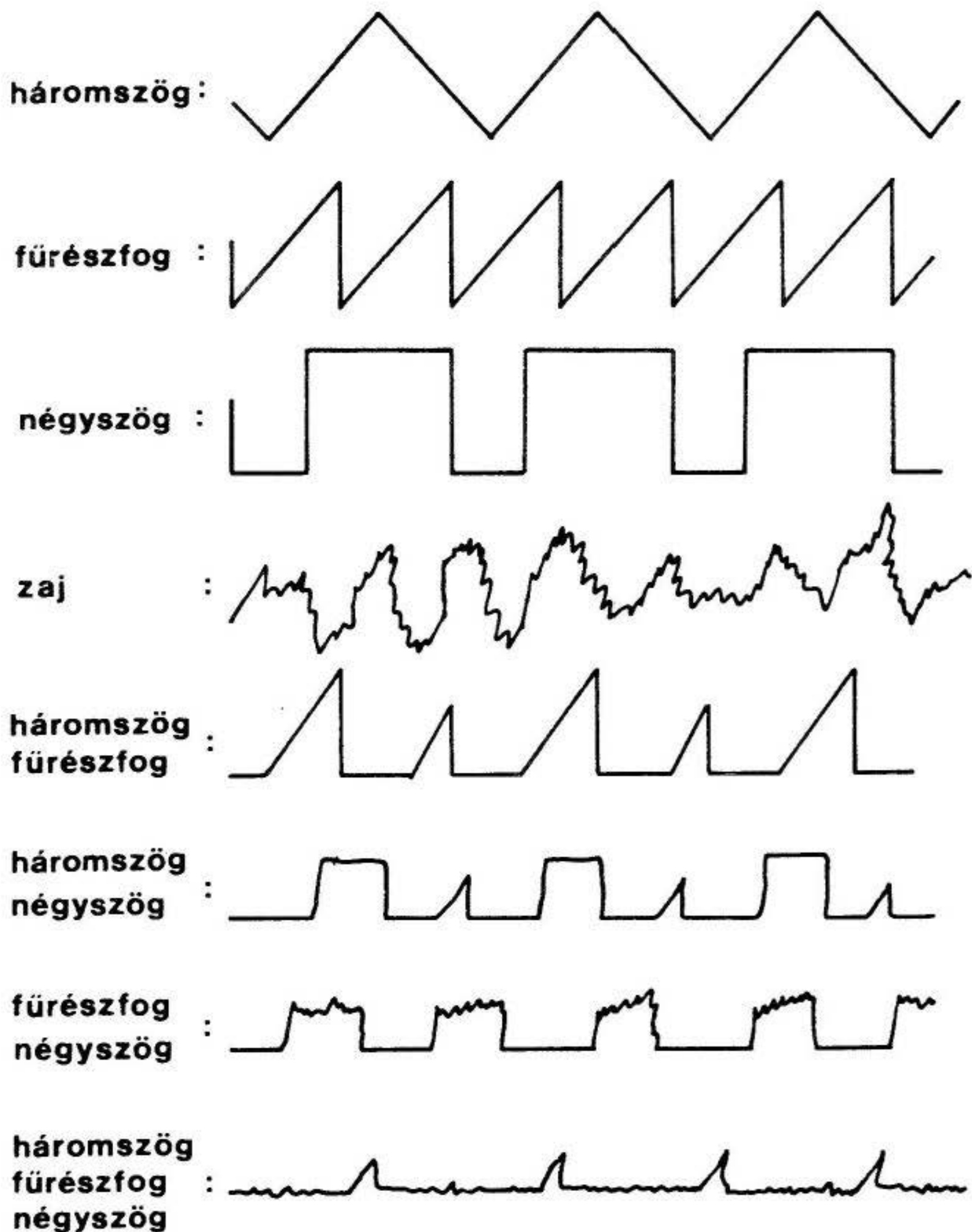
ahol F a keletkező hang frekvenciája, R a regiszterpár értéke. Az oszcillátorok lehetséges legnagyobb frekvenciája: $F_{\max} 3995 \text{ Hz}$.

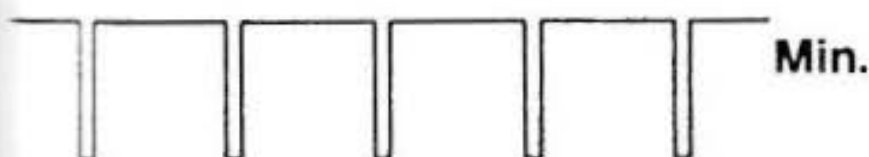
A hangskála programozásához szükséges adatokat a 4. táblázat foglalja össze. A logaritmus hangskála hangjainak paramétereit az alábbi képlet adja:

$$R = c \cdot 2^{(y-7+x/12)}$$

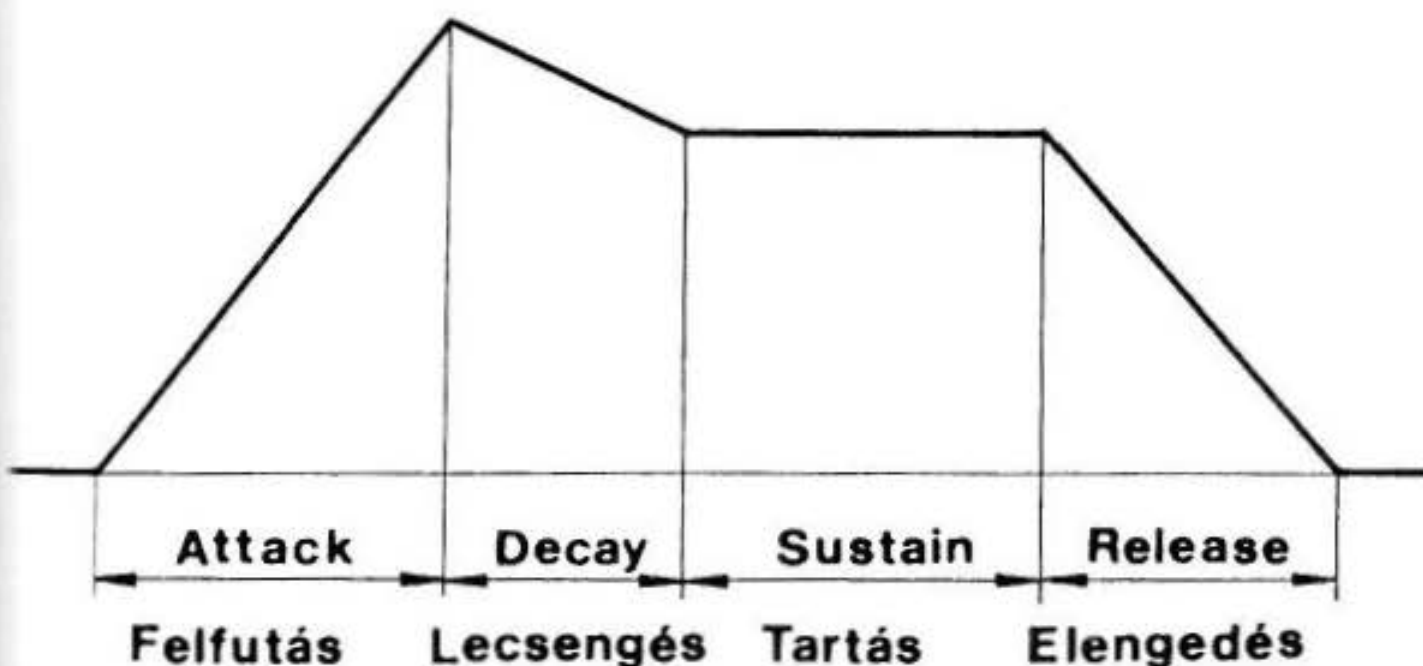
ahol $c = 34331$, y az oktáv száma (0–7),

x a hang száma az oktávon belül (0–11).





17. ábra. Torzított négyszögjel

18. ábra.
A hang burkológörbéje

Az oszcillátorok (rezgőkörök) által háromszög, fűrészfog, négyszög és zajhullámok állíthatók elő, valamint ezek kevert változatai.

A négyszög hullámforma programozása megkíván még egy paramétert, a kitöltési tényezőt. A kitöltési tényező két impulzus közötti alacsony jelszint hosszát adja meg. A maximális kitöltési tényező a négyszöghullámot a 17. ábra a) része szerint torzítja; minimális értéknél a helyzet fordított, ezt az ábra b) részén láthatjuk.

A kitöltési tényező egy 12 bites szám; maximális értéke 4095 lehet. Zéró, illetve maximális kitöltési tényező esetén túimpulzusok keletkeznek. A burkológörbe-generátort (ADSR) az 5. és 6. regiszteren keresztül programozhatjuk. A burkológörbe megválasztása jelentős mértékben befolyásolja a választott hullámforma által keltett hangérzetet.

A hang burkológörbét a 18. ábra szemlélteti.

- | | |
|-----------------------------|--|
| Felfutás: <i>Attack</i> . | Értéke meghatározza, hogy mennyi idő telhet el a maximális hangerő eléréséig. |
| Lecsengés: <i>Decay</i> . | Értéke azt az időtartamot szabja meg, amíg a hang a maximális hangerőről a <i>Tartás</i> szintjére esik vissza. |
| Tartás: <i>Sustain</i> . | Értéke a tartás hangerejét adja meg a maximális hangerőhöz viszonyítva. A <i>Tartás</i> értéke a <i>Lecsengés</i> fázisa végén lévő hang hangerejét adja meg. A <i>Tartás</i> időtartamát nem a regiszterben lévő érték, hanem a KEY bit bekapcsolt állapota szabja meg. Amíg a KEY bit magas, a hangerő a <i>Tartás</i> szintjén marad. |
| Elengedés: <i>Release</i> . | Értéke azt az időtartamot adja meg, amíg a hangerő a <i>Tartás</i> szintjéről nullára csökken. |

5. táblázat. A burkológörbe-generátor időállandói

Érték	Felfutás	Lecsengés	Elengedés
#0	2 ms	6 ms	6 ms
1	8 ms	24 ms	24 ms
2	16 ms	48 ms	48 ms
3	24 ms	72 ms	72 ms
4	38 ms	114 ms	114 ms
5	56 ms	168 ms	168 ms
6	68 ms	204 ms	204 ms
7	80 ms	240 ms	240 ms
8	100 ms	300 ms	300 ms
9	250 ms	750 ms	750 ms
10	500 ms	1,5 s	1,5 s
11	800 ms	2,4 s	2,4 s
12	1 s	3,0 s	3,0 s
13	3 s	9,0 s	9,0 s
14	5 s	15,0 s	15,0 s
15	8 s	24,0 s	24,0 s

A vezérlőregiszter bitjei határozzák meg a hullámformát, a hang be- és kikapcsolását, a gyűrűsmoduláció és a szinkronizálás mikéntjét. A hullámformát vezérlő bitek a már ismertek hullámokat képesek előállítani. A hang be- és kikapcsolását a KEY (0.) bit végzi.

A SYNC bit az oszcillátornak a többihez viszonyított fázishelyzetét határozza meg. A bit csak a többi szólam hasonló vezérlőbitjével együtt értelmezhető. A bitek különböző állapotai az oszcillátorok szinkronfutását szabják meg.

6. táblázat. A szinkronizáció

SYNC1	SYNC2	SYNC3	Értelmezés
0	0	0	Nincs szinkronizáció
1	0	0	Az 1. oszcillátort a 3. szinkronizálja
0	1	0	A 2. oszcillátort az 1. szinkronizálja
0	0	1	A 3. oszcillátort a 2. szinkronizálja
1	1	0	Az 1. és 2. oszcillátort a 3. szinkronizálja
0	1	1	A 2. és 3. oszcillátort az 1. szinkronizálja
1	0	1	Az 1. és 3. oszcillátort a 2. szinkronizálja
1	1	1	Mindhárom oszcillátor szinkronban

A RING bit a gyűrűsmódulációt kapcsolja be és ki. A gyűrűsmóduláció a bázisoszcillátor jelét a vezérlő oszcillátor frekvenciáját és hullámformáját figyelembe véve modulálja. A kimenőjel a két oszcillátor jelének logikai szorzata.

7. táblázat. A gyűrűsmóduláció

RING1	RING2	RING3	Értelmezés
0	0	0	Nincs gyűrűsmóduláció
1	0	0	Az 1. oszcillátort a 3. modulálja
0	1	0	A 2. oszcillátort az 1. modulálja
0	0	1	A 3. oszcillátort a 2. modulálja
1	1	0	Az 1. és 2. oszcillátort a 3. modulálja
0	1	1	A 2. és 3. oszcillátort az 1. modulálja
1	0	1	Az 1. és 3. oszcillátort a 2. modulálja
1	1	1	Mindhárom oszcillátor modulálja a többi

A TEST bit értékét a programozó állítja be. Alaphelyzetben értéke 0. Ha magas, a szólam nem hallatszik. Előfordulhat, hogy a SID önmaga is bebillenti ezt a bitet, ha a zajgenerátor mellé még másik hullámformát is bekapcsolunk. A szólam ekkor leblokkol, mert az oszcillátor kapcsolása elvileg ilyen variációt nem tesz lehetővé. A bit törlése ilyenkor a blokkolt oszcillátort felszabadítja.

Hang előállítása BASIC programmal

Egy hang megszólaltatása nem tartozik a bonyolult problémák közé. Nem kell törődni a hangok közötti szünetek beállításával, a lecsengés és a tartás közötti viszony meghatározásával. Elegendő a hang burkológörbét, frekvenciáját és hullámformáját megállapítani. Amiatt érdemes illusztrálni, mert a SID regisztereit meghatározott sorrendben kell beállítani. A hullámforma beállítása és a hang bekapcsolása után már késő a frekvencia megadása, és elkészt a burkológörbe-generátor programozása is.

A frekvencia, a burkológörbe és a kitöltési tényező (ha van) megadása történhet bármilyen sorrendben, de meg kell előzniük a hullámforma megadását és a hang bekapcsolását. Ahhoz, hogy a hang valóban hallható legyen, a hangerőt még a program elején be kell állítani.

```

100 SI=54272
110 POKE SI+24,15 :REM Hangerő
120 POKE SI ,0 :REM Frekvencia LO
130 POKE SI+1 ,170 :REM Frekvencia HI
160 POKE SI+5 ,0 :REM AD
170 POKE SI+6 ,9 :REM SR
180 POKE SI+4 ,17 :REM Hang be
190 POKE SI+4 ,16 :REM Hang ki

```

Ugyanaz a program négyszög hullámformával:

```

100 SI=54272
110 POKE SI+24,15 :REM Hangerő
120 POKE SI ,0 :REM Frekvencia LO
130 POKE SI+1 ,170 :REM Frekvencia HI
140 POKE SI+2 ,0 :REM Kitöltés LO
150 POKE SI+3 ,8 :REM Kitöltés HI
160 POKE SI+5 ,0 :REM AD
170 POKE SI+6 ,9 :REM SR
180 POKE SI+4 ,65 :REM Hang be
190 POKE SI+4 ,64 :REM Hang ki

```

A két példa bemutatja, hogyan lehet egy hangot előállítani. A beállított frekvenciát tulajdonképpen csak az *Elengedés* ideje alatt hallhatjuk, mert mind a *Felfutás*, mind a *Lecsengés* beállítható ideje rövid. A fenti két példa az elengedés alatt teszi a hangot hallhatóvá. Ennek a programozási módszernek az az előnye, hogy a megszólaltató program vagy szubrutin várakozóciklusok nélkül, gyorsan fut le.

Létezik olyan programozási módszer is, amikor a hangot a *Tartás* ideje alatt hallhatjuk. A hangot a SID egészen addig tartja, amíg a vezérlőregiszter (itt \$D404) KEY bitje magas. Ennek a programozási módszernek hátránya az, hogy viszonylag lassan fut le. A módszert az alábbi példaprogram szemlélteti:

100 POKE 53280,0:POKE 53281,0	
110 SI=54272	; A SID kezdőcíme
120 POKE SI+24,15	; A hangerő beállítása
130 W=16	; A hullámforma : háromszög
140 AD=112+10	; Felfutás: 68 ms, Lecsengés: 1.5 s
150 SR=112+8	; Tartás: 50 %, elengedés: 300 ms
160 POKE SI+5,AD	; A Felfutás-Lecsengés beállítása
170 POKE SI+6,SR	; A Tartás-Elengedés beállítása
180 READ A\$; A Következő hang adatai
190 IF A\$="-1" THEN END	; Ha nincs több, megállni
200 GOSUB 600	; A paraméterek kiszámítása
210 POKE SI ,LF	; A frekvencia alsó beállítása
220 POKE SI+1,HF	; A frekvencia felső beállítása
230 POKE SI+4,W+1	; A hang megszólaltatása
240 FOR I=1 TO T*50:NEXT	; Tartás - amíg a ciklus fut
250 POKE SI+4,W	; A hang kikapcsolása
260 FOR I=1 TO T*50:NEXT	; Megvárni az elengedés végét
270 GOTO 180	; Vissza új adatot feldolgozni
280 :	; Szubrutin a paraméterek kiszámításához
600 Y=VAL(LEFT\$(A\$,1))	; Az oktáv száma
610 X=VAL(MID\$(A\$,2,2))	; A hang oktávon belüli sorszáma
620 T=VAL(RIGHT\$(A\$,2))	; A Tartás hossza
630 R=INT(34331*2^(Y+X/12-7))	; A frekvenciaekvivalens kiszámítása
640 HF=INT(R/256)	; ill. felbontása felső
650 LF=R-256*HF	; és alsó byte-ra
660 RETURN	
670 :	; A hangok adatai
800 DATA 50202,50204,50202	; 50204= 5-02-04
810 DATA 50701,50502,50402	; 5 = az oktáv száma
820 DATA 50501,50202,40910	; 02 = a hang száma (ez esetben D hang)
830 DATA -1	; 04 = kb. 4*50 ms tartás

Ez a program több hang egymás utáni megszólaltatását oldja meg. A három szólam közül csak egyet használ. A hangjegyeket kódolt formában, DATA sorban keresi. A kódot a 600-as szubrutin számítja át. A program ugyan a háromszög hullámformát alkalmazza, de a W változóban más hullámformát is megadhatunk.

A program a hangsorozat megszólaltatását akkor fejezi be, amikor a READ utasítás –1 értéket olvasott be.

A SID *szűrői* bizonyos frekvenciákat nem engednek ki a kimenetre, emiatt azok akkor sem hallhatók, ha egyáltalán létrejöttek.

Az akusztika *négyféle* hangfrekvenciás szűrőt különböztet meg: aluláteresztő, feluláteresztő, sávszűrő és sávzáró szűrőt. Az aluláteresztő szűrő egy megadott frekvencia alatti hangokat engedi át, a feluláteresztő szűrő ennek épp ellenkezője. A sávszűrő és sávzáró szűrő szintén egymás ellentétei. A sávszűrő egy bizonyos frekvenciasávon kívül semmit sem enged át, a sávzáró pedig a sávon kívül mindent átereszt. A SID mind a négy szűrőt előállítja. A szűrők működését négy regiszter szabályozza.

A 20. és 21. regiszterben (\$D414–15) a szűrő *levágási frekvenciáját* tudjuk megadni. Érdekes, hogy a két regiszterben egy 11 bites szám helyezhető el, de nem a szokásos LO-HI byte módon. A 20. regiszterben a frekvencia 0–2 bitjei vannak, a többi (3–10) bitek pedig a 21. regiszterben (\$D415).

A 22. regiszterben (\$D416) az oszcillátorokhoz tartozó szűrők kapcsolóbitjei (0–2), a *rezonanciafrekvencia* vezérlőbitjei (4–7) és egy különleges bit, a FILTEX található. A FILTEX a chip EXT IN bemenetére érkező hangfrekvenciás jelet szűri és keveri a chip 3. oszcillátorának jeleihez. A rezonanciafrekvencia bitjei a szűrőt pseudo oszcillátorként vezérlik. A bitek tartalma (0–15) szabja meg, hogy a szűrő milyen mértékben vágja a frekvenciát, azaz a szűrés a frekvencia amplitúdójára vonatkozik. Ez a mód a hangzást jelentős mértékben befolyásolja.

A 23. regiszterben (\$D417) az összhangerő (0–15) beállítására szolgáló 0–3. bitek mellett a következő vezérlőbitek találhatók:

- 4. bit: aluláteresztő szűrő
- 5. bit: sávszűrő
- 6. bit: feluláteresztő szűrő
- 7. bit: a 3. szólam hangzásának megszüntetésére szolgáló bit

A megfelelő szűrőtípust a 23. regiszter (\$D417) 4–6. bitjei választják ki; a lehetséges kombinációkat a 8. táblázat mutatja be. A mindent lezáró szűrőtípus teljesen kiiktatja a megfelelő szólamot.

A 23. regiszter (\$D417) 7. bitjének akkor lehet szerepe, ha speciális szinkron hangvezérlést alkalmazunk. Bekapcsoljuk a 3. szólamot úgy, hogy azt az 1. szólam szinkronizálja, majd beállítja a 7. bitet, és az 1. szólam frekvenciájának másolatát olvashatjuk a 26. regiszterben (\$D41A). Erre akkor lehet szükség, ha a szólam frekvenciáját a hang megszólalása közben akarjuk folyamatosan szabályozni. A 26. regiszterben (\$D41A) a frekvencia felső byte-ja jelenik meg.

8. táblázat. A szűrők kiválasztása

6.	5.	4. bit	Értelmezés
0	0	0	Lezáró szűrő (mindent)
0	0	1	Aluláteresztő szűrő
0	1	0	Sávszűrő
1	0	0	Feluláteresztő szűrő
0	1	1	Aluláteresztő és sávszűrő
1	0	1	Alul- és feluláteresztő (sávzáró) szűrő
1	1	0	Feluláteresztő és sávszűrő
1	1	1	Mindent áteresztő

Hasonló a helyzet a 27. regiszterrel. A regiszterben (\$D41B) a 3. szólam burkológörbe-generátorának pillanatnyi értéke olvasható ki. Ebből lehet következtetni a relatív hangerőre, és ez további hangzás közbeni módosításra ad lehetőséget.

1.4.3. Az analóg/digitális átalakítók

Életünk analóg mennyiségek között zajlik. Mértékegységeink mindegyike analóg mennyiségek meghatározására, összehasonlítására szolgál. Analóg mennyiség a tömeg, a sebesség, a hőmérséklet, a fény intenzitása, a távolság, a hangerősség, a feszültség stb. A számítógép számára az analóg mennyiségek végtelenül finom fokozataik miatt feldolgozhatatlanok, hiszen pontos érzékelésükre a gép nem képes. Digitális feldolgozásuk együtt jár bizonyos pontatlansággal, mert a digitális elektronika csak véges számú fokozatot tud megkülönböztetni. Ez a pontatlanság a technika mai színvonalán már igen csekély mértékű, olykor észrevehetetlen.

Analóg jelek digitálissá alakítására többféle módszer létezik. A SID-ben található A/D átalakítók nem a legelterjedtebb módszerek egyikét alkalmazzák. Az átalakítók — egy kondenzátor segítségével — végső soron ellenállást mérnek. A kondenzátor 0,256 ms időközönként kisül. A feltöltődés a kimenetre kapcsolt ellenálláson keresztül megy végbe. A feltöltődés kezdetekor elindul egy számláló, amelyet a rendszerütem vezérel. Amint a kondenzátor feszültsége eléri a referenciafeszültséget, a számláló leáll. A kondenzátor feszültség szintjét egy komparátor ellenőrzi. Ha a külső ellenállás (paddle) értéke kisebb, mint 200Ω , a kondenzátor hamarabb feltöltődik, mintsem a számláló egyáltalán elindulhatna. Így a számláló értéke 0 marad. Ha az ellenállás nagyobb, mint $200k\Omega$, akkor a kondenzátor a számlálási idő alatt nem tud feltöltődni, ezért a számláló értéke 255 lesz. Az ellenállás értéke ne legyen kisebb 100Ω -nál, mert a mérési ciklus végén a kondenzátor ütésszerűen kisül, s ez túltöltődés esetén viszonylagosan nagy áramerősség kialakulásával jár együtt. Az áramerősség ekkora értéke kárt

tehet a SID-ben, amelynek belső áramköre csak néhány mA áramot képes elviselni. A mérési ciklus 0,256 ms, majd ugyanennyi szünet következik, a teljes ciklus tehát kb. 0,5 ms, azaz a mérés másodpercenként kb. 2000-szer ismétlődik meg.

Az analóg/digitális átalakítók külső csatlakozása a Control portokon található. A SID két A/D regisztere egyszerre tartozik mindkét port POTX, illetve POTY bemenetéhez. A kiválasztást a CIA1 0. regiszterének (\$DC00) 6. és 7. bitje végzi.

Az alábbiakban paddle-olvasásra mutatunk be egy példát. Az ilyen feladatok programozása nem különösebben bonyolult, de programunkban van egy apró trükk, az olvasás közbeni kitartóciklus, amely nélkül nem kapunk helyes eredményt. A program az eredmény megjelölésére hat BASIC változót használ:

AB: A paddle tűzgomb, ha lenyomva, értéke nem nulla

AX: A paddle X irány

AY: A paddle Y irány

BB: B paddle tűzgomb, nulla, ha nincs lenyomva

BX: B paddle X irány

BY: B paddle Y irány

A gépi program ezekbe a változókba helyezi az eredményt, tehát a BASIC programban nem kell bíbelődni a tárrekeszek (REGAB stb.) kiolvasásával. A portok így kényelmesen ellenőrizhetők. A program SYS 10000-rel hívható.

```

10 IFPEEK(10000)=120ANDPEEK(10015)=169THEN40
20 FORI=10000TO10150: READ A
30 POKEI,A:NEXT I
40 REM AB=PADDLE A TUZGOMB
50 REM AX=PADDLE A X IRANY
60 REM AY=PADDLE A Y IRANY
70 REM BB=PADDLE B TUZGOMB
80 REM BX=PADDLE B X IRANY
90 REM BY=PADDLE B Y IRANY
100 SYS10000
110 PRINT"PADDLE A TUZGOMB : "AB
120 PRINT"PADDLE AX : "AX
130 PRINT"PADDLE AY : "AY
140 PRINT"PADDLE B TUZGOMB : "BB
150 PRINT"PADDLE BX : "BX
160 PRINT"PADDLE BY : "BY
999 END
1000 DATA 120,169,128, 32,111, 39,141,161, 39,142,162, 39
1010 DATA 140,163, 39,169, 64, 32,111, 39,141,164, 39,142
1020 DATA 165, 39,140,166, 39,169,255,141, 2,220, 88,173
1030 DATA 161, 39,162, 65,160, 66, 32,136, 39,173,162, 39
1040 DATA 162, 65,160, 88, 32,136, 39,173,163, 39,162, 65
1050 DATA 160, 89, 32,136, 39,173,164, 39,162, 66,160, 66
1060 DATA 32,136, 39,173,165, 39,162, 66,160, 88, 32,136
1070 DATA 39,173,166, 39,162, 66,160, 89, 76,136, 39,141
1080 DATA 0,220, 9,192,141, 2,220,162,255,202,208,253
1090 DATA 174, 25,212,172, 26,212,173, 0,220, 41, 12, 96
1100 DATA 134, 69,132, 70,168,162, 0,134, 13,134, 14,138
1110 DATA 32,145,179, 32,231,176,133, 73,132, 74, 76,208
1120 DATA 187, 0, 0, 0, 0, 0, 0

```

2710			100		*=10000	
2710	78		110	MAIN	SEI	; a megszakítások letiltása
2711	A9	80	120		LDA #\$80	; %10000000, az A paddle kiválasztása
2713	20	6F	130		JSR OLVAS	; beolvasás
2716	8D	A1	140		STA REGAB	; A paddle tűzgomb
2719	8E	A2	150		STX REGAX	; A paddle X irány
271C	8C	A3	160		STY REGAY	; A paddle Y irány
271F	A9	40	170		LDA #\$40	; %01000000, a B paddle kiválasztása
2721	20	6F	180		JSR OLVAS	; beolvasás
2724	8D	A4	190		STA REGBB	; B paddle tűzgomb
2727	8E	A5	200		STX REGBX	; B paddle X irány
272A	8C	A6	210		STY REGBY	; B paddle Y irány
272D	A9	FF	220		LDA #\$FF	; minden vonal kimenetre
272F	8D	02	230		STA \$DC02	; a DDRA visszaállítása
2732	58		240		CLI	; a megszakítások engedélyezése
2733	AD	A1	250		LDA REGAB	
2736	A2	41	260		LDX #"A"	; az A paddle tűzgomb
2738	A0	42	270		LDY #"B"	
273A	20	88	280		JSR ERTEK	; értéke az AB-be
273D	AD	A2	290		LDA REGAX	
2740	A2	41	300		LDX #"A"	; az A paddle X irány
2742	A0	58	310		LDY #"X"	
2744	20	88	320		JSR ERTEK	; értéke az AX-be
2747	AD	A3	330		LDA REGAY	
274A	A2	41	340		LDX #"A"	; az A paddle Y irány
274C	A0	59	350		LDY #"Y"	
274E	20	88	360		JSR ERTEK	; értéke az AY-ba
2751	AD	A4	370		LDA REGBB	
2754	A2	42	380		LDX #"B"	; a B paddle tűzgomb
2756	A0	42	390		LDY #"B"	
2758	20	88	400		JSR ERTEK	; értéke a BB-be
275B	AD	A5	410		LDA REGBX	
275E	A2	42	420		LDX #"B"	; a B paddle X irány
2760	A0	58	430		LDY #"X"	
2762	20	88	440		JSR ERTEK	; értéke a BX-be
2765	AD	A6	450		LDA REGBY	
2768	A2	42	460		LDX #"B"	; a B paddle Y irány
276A	A0	59	470		LDY #"Y"	
276C	4C	88	480		JMP ERTEK	; értéke a BY-ba
276F	8D	00	490	OLVAS	STA \$DC00	; a paddle kiválasztása
2772	09	C0	500		ORA #\$C0	; %11000000, a paddle bitek bemenetre
2774	8D	02	510		STA \$DC02	; DDRA
2777	A2	FF	520		LDX #\$FF	
2779	CA		530	AA1	DEX	; várakozó ciklus
277A	D0	FD	540		BNE AA1	
277C	AE	19	550		LDX \$D419	; A/D1, paddle X
277F	AC	1A	560		LDY \$D41A	; A/D2, paddle Y
2782	AD	00	570		LDA \$DC00	; a paddle tűzgomb állásának beolvasása
2785	29	0C	580		AND #\$0C	; %00001100
2787	60		590		RTS	
2788	86	45	600	ERTEK	STX \$45	; a tárolandó
278A	84	46	610		STY \$46	; változó neve
278C	A8		620		TAY	
278D	A2	00	625		LDX #\$00	
278F	86	0D	630		STX \$0D	; a változó típus beállítása
2791	86	0E	640		STX \$0E	; aritmetikai-valós
2793	8A		645		TXA	
2794	20	91	650		JSR \$B391	; az A/Y átalakítása real alakra
2797	20	E7	660		JSR \$B0E7	; a változó keresése v. létrehozása
279A	85	49	670		STA \$49	; a változó címe
279C	84	4A	680		STY \$4A	; a változó területen
279E	4C	D0	690		JMP \$BBD0	; a lebegőpontos szám a változóba
27A1	00		700	REGAB	.BYTE \$00	
27A2	00		710	REGAX	.BYTE \$00	
27A3	00		720	REGAY	.BYTE \$00	
27A4	00		730	REGBB	.BYTE \$00	
27A5	00		740	REGBX	.BYTE \$00	


```
27A6 00          750 REGBY  .BYTE $00
27A7            760          .END

ZEILEN:69  SYMBOLE:10  FEHLER:0

AA1  =2779  ERTEK =2788  MAIN  =2710  OLVAS  =276F  REGAB  =27A1  REGAX  =27A2
REGAY  =27A3  REGBB  =27A4  REGBX  =27A5  REGBY  =27A6
```

1.5. A tárkezelő (AM)

A RAM és a ROM tartományokhoz a memória „szendvics” szerkezete miatt nem lehetséges egy időben hozzáférni. Ahhoz, hogy ez — időben megosztva — mégis lehetségessé váljon, szükség van egy vezérlő-ütemező egységre. Ez az egység az Address Manager, másképpen Tárkezelő egység, röviden AM.

Az AM — 825100 egy 28 lábú DIL-tokozású integrált áramkör.

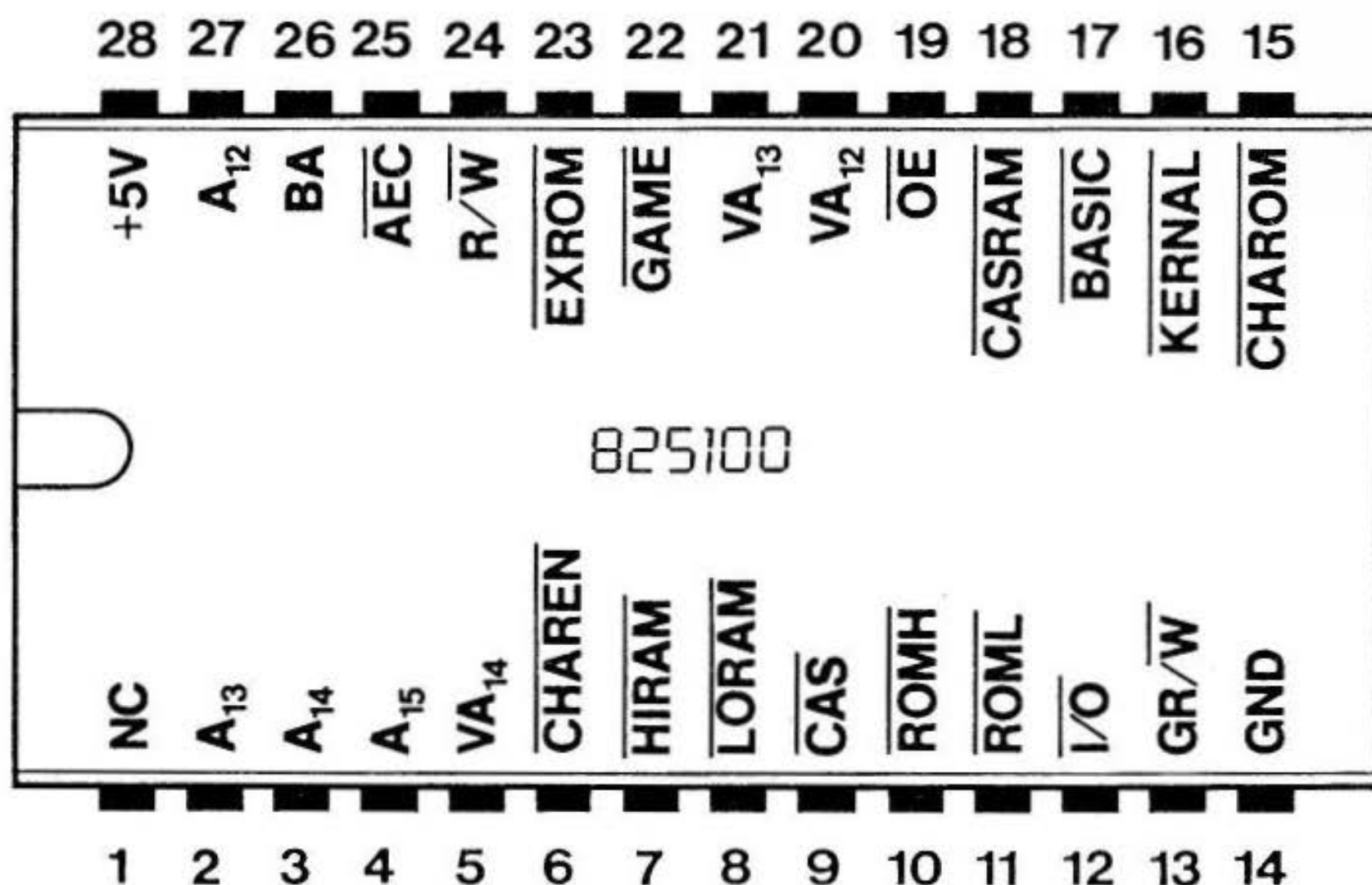
Az AM 16 bemeneti és 8 kimeneti jellel működik. A nyolc kimeneten 256 különböző kombináció lehetséges, de sok bemeneti kombináció azonos kimenetet eredményez. Emiatt az előforduló tárkonfigurációk száma csekély. A kimenetek állapotát szoftveresen is befolyásolhatjuk. A processzorport alsó 3 bitje ezt a célt szolgálja. A három bit 8 különböző állapotot hozhat létre.

Az 1. és a 4. kombináció azonos eredményt ad. Ennek oka az, hogy az AM-et még további vonalak is befolyásolják.

Kivezetéseinek jele és szerepe:

9. táblázat. A tárkezelő bitkonfigurációi

Bit		Write-Read		Write-Read		Write-Read		Write-Read
2. CHAREN	0	RAM - RAM	0	RAM - ROM	0	RAM - ROM	0	RAM - ROM
1. HIRAM	0	RAM - RAM	0	RAM - RAM	1	RAM - ROM	1	RAM - ROM
0. LORAM	0	RAM - RAM	1	RAM - RAM	0	RAM - RAM	1	RAM - ROM
2. CHAREN	1	RAM - RAM	1	I/O - I/O	1	I/O - I/O	1	I/O - I/O
1. HIRAM	0	RAM - RAM	0	RAM - RAM	1	RAM - ROM	1	RAM - ROM
0. LORAM	0	RAM - RAM	1	RAM - RAM	0	RAM - RAM	1	RAM - ROM



19. ábra. Az AM — 825100 tárkezelő tokozása

1. Nem használt
- 2—4. A₁₃-A₁₅: a címbusz 13 — 15. bitjei, bemenet
5. VA₁₄: a videocímbusz 14. bitje, bemenet
6. CHAREN: a processzorport 2. bitje (P₂), bemenet
7. HIRAM: a processzorport 1. bitje (P₁), bemenet
8. LORAM: a processzorport 0. bitje (P₀), bemenet
9. CAS: a dinamikus RAM oszlopcímző impulzusa, bemenet
10. ROMH: a bővítőport ROMH (\$A000 — FFFF) vonala, kimenet
11. ROML: a bővítőport ROML (\$8000 — 9FFF) vonala, kimenet
12. I/O: vezérlővonal, a CIA-k kiválasztását végzi, kimenet
13. GR/W: vezérlővonal, a COLOR RAM kiválasztását jelzi, kimenet
14. GND: rendszerföld
15. CHAROM: a CHAREN-t kiválasztó vezérlővonal, kimenet
16. KERNAL: a KERNAL-t kiválasztó vezérlővonal, kiemenet
17. BASIC: a BASIC ROM-ot kiválasztó vezérlővonal, kimenet
18. CASRAM: a dinamikus RAM oszlop kiválasztó vonala, kimenet
19. OE: Output Enable. Mindig földelve van
- 20—21. VA₁₂-VA₁₃: a videocímbusz 12 — 13. bitjei, bemenet
22. GAME: a bővítőport vezérlővonala. Ha alacsony, játékkazetta van a bővítőportban, bemenet

23. $\overline{\text{EXROM}}$: a bővíítőport vezérlővonala. Ha alacsony, kazetta van a bővíítőportban. A \$8000 — 9FFF területre érvényes, bemenet
24. $\text{R}/\overline{\text{W}}$: a processzor írásengedélyező vonala, bemenet
25. $\overline{\text{AEC}}$: Address Enable Control. A processzor címzésengedélyező vonala, bemenet
26. BA: Bus Available. A busz használatát engedélyező vonal, bemenet
27. A₁₂: a címbusz 12. bitje, bemenet
28. Vcc: +5 V tápfeszültség

1.6. A memória

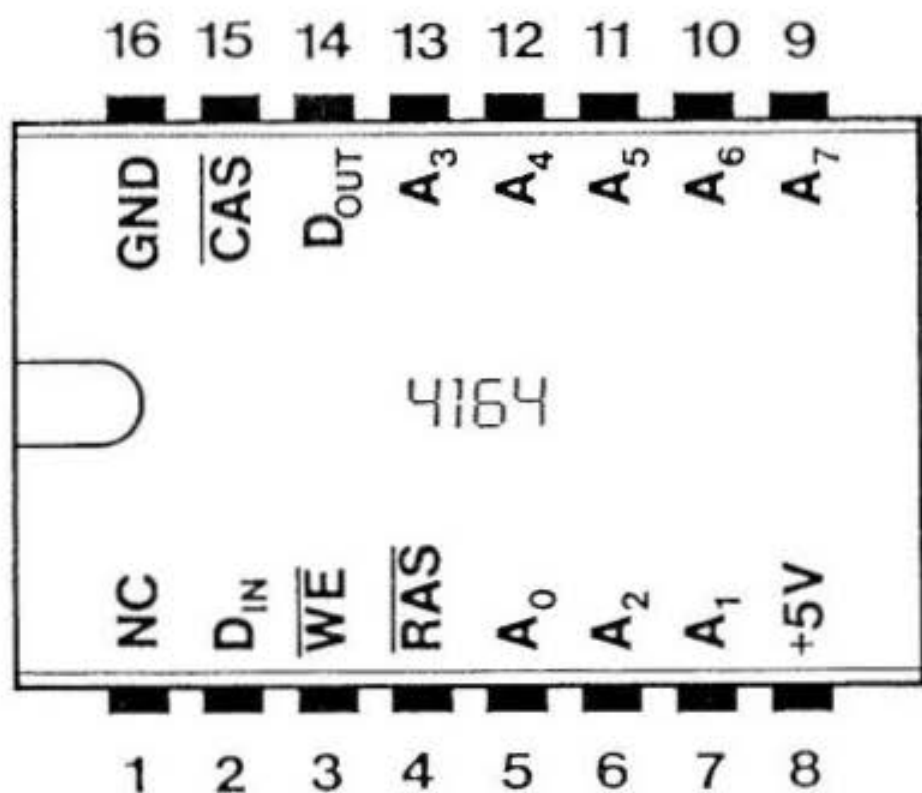
A számítógépekben alkalmazott táruk két típusra oszthatók; a *felejtő táruk* a tápfeszültség megszüntetése után elveszítik tartalmukat, míg a *nem felejtő táruk* általában csak külső behatásra (törlő impulzus, UV-fény stb.) változtatják meg, ha ez egyáltalán lehetséges. A nem felejtő táruk tartalmát rendszerint a gyártás során meghatározzák. Ezeket a tárukat az angol Read Only Memory (csak olvasható memória) kifejezés alapján ROM-nak nevezzük.

A felejtő táruknak két változata létezik: a dinamikus és a statikus. A két tárfajta közötti alapvető eltérés, hogy a statikus memória nem igényel működés közbeni, ciklikusan ismétlődő frissítést. A felejtő tárukat az angol Random Access Memory (véletlen elérésű memória) elnevezés rövidítése után RAM-nak nevezzük. A C64-esben dinamikus RAM chippek vannak; ezek az áramkörök tartalmuk megőrzéséhez folyamatos frissítőimpulzusokat várnak. A frissítésre tárolási elvük miatt van szükség. A dinamikus memória olyan tranzisztorok sokaságából áll, amelyek apró kapacitásokat töltenek fel, illetve sütnek ki. A magas logikai szint általában a kondenzátor feltöltött állapotának felel meg. Mivel ideális kondenzátor a gyakorlatban nem hozható létre, értelemszerű, hogy a kondenzátor töltése külső behatás híján is idővel csökkenni fog. E töltésszivárgást hivatott pótolni a frissítőimpulzus, amely maximum 2 ms-onként bekövetkezik, és helyreállítja az eredeti töltésszintet.

1.6.1. Az írható-olvasható táruk: RAM

A C64-esben 64 kbyte folyamatosan címezhető RAM van. Ezt 8 db 64 kbites chip párhuzamos kapcsolásával érték el, illetve az újabb gépekbe 2 db 256 kbites RAM chipet építenek. A 64 kbyte RAM a processzor számára byte-szervezésű, azaz egyszerre, egy-egy címzőimpulzussal 8 bithez férhet hozzá. Ez azonban nem jelenti azt, hogy a RAM fizikailag is byte-szervezésű. A nyolc RAM chip mindegyike 64 kbites. Ezek az áramkörök, mint diszkrét elemek, egymá-

gukban egyetlen byte-ot sem tárolnak, hanem mindegyik (65536) byte-ból mindig ugyanazt a bitet. A sorban az első RAM áramkör a processzor címtartományába eső összes byte-ból csak a 0. bitet tárolja. A második az 1. bitet és így tovább.



20. ábra. A 4164 RAM áramkör tokozása

A 4164 láb kiosztása

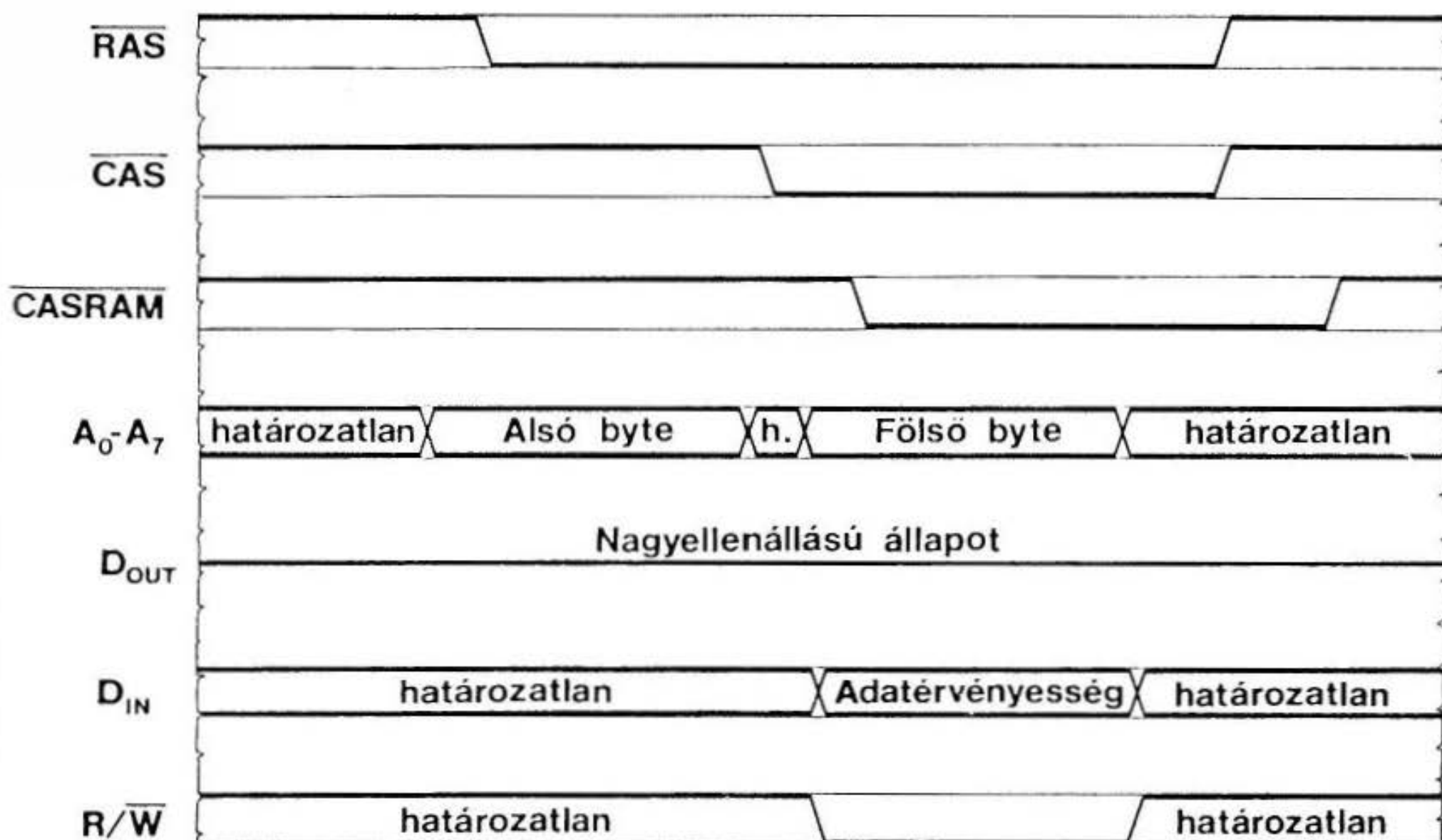
1. NC:	nem használt	9. A ₇ :	7. címbit
2. D _{IN}	adatbemenet	10. A ₆ :	6. címbit
3. $\overline{\text{WE}}$:	írásengedélyezés	11. A ₅ :	5. címbit
4. $\overline{\text{RAS}}$:	sorcímző impulzus	12. A ₄ :	4. címbit
5. A ₀ :	0. címbit	13. A ₃ :	3. címbit
6. A ₂ :	2. címbit	14. D _{OUT} :	adatkimenet
7. A ₁ :	1. címbit	15. $\overline{\text{CAS}}$:	oszlopcímző impulzus
8. 5 V	tápfeszültség	16. Föld	

Az áramkör $\overline{\text{WE}}$ kivezetéséhez a processzor R/ $\overline{\text{W}}$ vonala, a $\overline{\text{RAS}}$ kivezetéshez a VIC $\overline{\text{RAS}}$ vonala, a $\overline{\text{CAS}}$ kivezetéshez az AM CASRAM vonala van illesztve. Az áramkör bitjeinek címezése két lépésben történik, mert az áramkör nyolc címzővezetése a fizikai címnek csak a felét veheti át egyszerre. A címezés megosztását az SN74LS257 jelű multiplexerek végzik. Az A₀-A₇ bemenetekre előbb a cím alsó byte-ja, majd a felső byte-ja kerül.

A RAM adatforgalma

Írás (WRITE) esetén a VIC $\overline{\text{RAS}}$ vonala alacsonyra vált, ekkor a RAM átveszi a cím alsó byte-ját. A RAM címezését vezérlő két multiplexert a VIC $\overline{\text{CAS}}$ vonala vezérli. A jel alacsony állapota a cím felső byte-jának továbbítását engedélyezi, ellenkező esetben pedig a cím alsó byte-ja jelenhet meg a címbemeneteken. A RAS vonal alacsonyra állítása után nem sokkal a $\overline{\text{CAS}}$ vonal is alacsonyra vált; ekkor kell bekövetkeznie a cím felső byte-ja átvételének. A

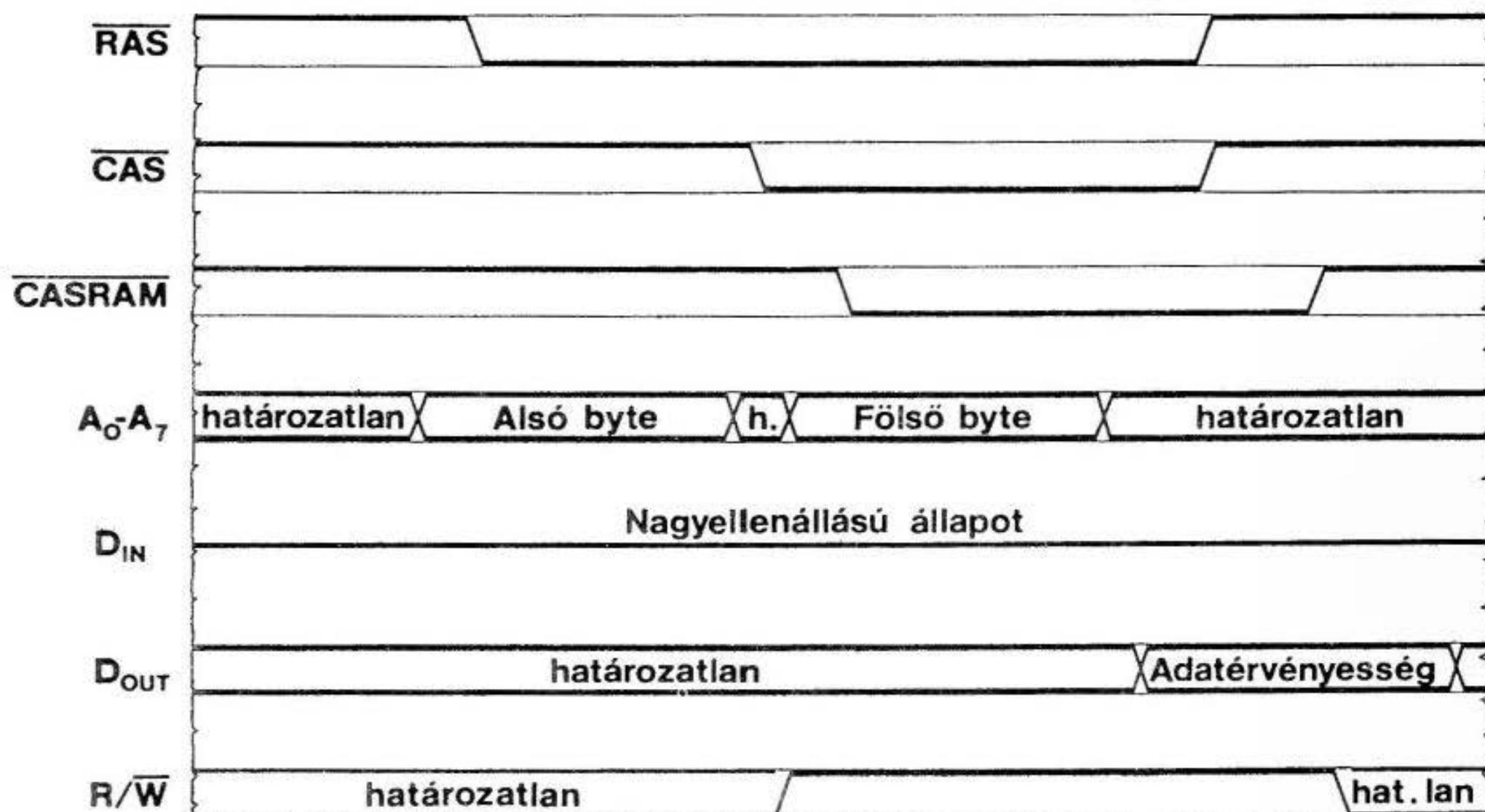
RAM azonban addig nem fogadja a felső byte-ot, amíg az AM $\overline{\text{CASRAM}}$ kimenete alacsony szintre nem kerül. A $\overline{\text{CASRAM}}$ alacsonyra váltását kevéssel megelőzve a processzor R/W vonala is alacsony szintet vesz fel. Ez a RAM számára azt jelenti, hogy az adatbuszon megjelenő adatokat átveheti, és az előzetesen, illetve közben megcímzett tárcellába továbbíthatja. Röviddel a bit átvétele után a VIC magasra állítja a $\overline{\text{RAS}}$ vonalat, amit azonnal követ $\overline{\text{CAS}}$ vonal is. Némi késleltetés után az AM is magasra állítja a $\overline{\text{CASRAM}}$ vonalat.



21. ábra. Az írás folyamata

Az olvasás (READ) előtti címzés módja azonos az írás előtti címzéssel. A folyamatban különbség az R/W vonal állapotában és az adatérvényesség helyében van. Az adatbit akkor kerülhet ki a buszra, amikor a RAM áramkörei már mindkét címbyte-ot átvették. Magától értetődik, hogy az adat csak ezután kerülhet ki a DOUT kimenetre. Az adatérvényesség a $\overline{\text{CASRAM}}$ magasra váltásával szűnik meg.

Mindkét hozzáférési módra érvényes, hogy az R/W vonal akkor vehet fel határozott állapotot, ha a $\overline{\text{CAS}}$ jel már alacsony, a $\overline{\text{CASRAM}}$ pedig még magas. A $\overline{\text{CASRAM}}$ ugyan a $\overline{\text{CAS}}$ -tól függ, ám az AM mind az alacsonyra, mind a magasra állítását késlelteti.



22. ábra. Az olvasás folyamata

A RAM felosztása

A 64 kbyte RAM ugyan folyamatos, bekapcsolás után mégis csak egy részét használhatjuk saját céljainkra. Alapállapotban ugyanis 20 kbyte-ot ROM fed, 2 kbyte-ot pedig az interpreter foglal le magának.

A RAM-ot nem a mikroszámítógépeknél szokásos módon osztották fel. A C64-esben a ROM-ok a címtartomány végén és nem az elején vannak. Az operációs rendszer által elfoglalt terület pedig közvetlenül a címtartomány legelején helyezkedik el. Ennek főként a processzor az oka: a 6510-es processzor ugyanis a nulláslapon levő címeket a felső byte (amely itt mindig \$00) megadása nélkül is elérheti, címezheti. A gépi programok így memóriatakarékosabbak és gyorsabbak lehetnek. A RAM emiatt nem is osztható másképpen, csak úgy, hogy a nulláslap szabadon elérhető RAM legyen. Így került a ROM a címtartomány végére.

A RAM-ot az általánosan elfogadott programozási gyakorlat 256 byte-os lapokra osztja. 64 kbyte címterület 256 x 256 byte-ból áll, ezért 256 lapot különböztethetünk meg. A lapok számozása a bennük foglalt tárcímek felső byte-ja szerint történik. Emiatt létezik 0-ás (\$00), 1-es (\$01) lap, és így tovább, melyek között a nulláslap kiemelt szerepet kapott. A processzor gépi nyelve lehetővé teszi, hogy a RAM-ot indirekt módon is megcímezhessek, s ezzel a címzési móddal olyan hajlékony programozási lehetőséget biztosít, amely más processzorokkal nem, illetve nehezen érhető el.

Kiemelt szerepe az 1., 2. és 3. lapnak van még. Az 1. lapot, a vermet a processzor használja.

A 2. és 3. lapot az interpreter és az operációs rendszer használja, de ezek egy része szabad. Alaphelyzetben a VIC a képernyőmemóriát a 4—7. lapon keresi. A 8—159. sorszámú lapok a BASIC programtár és a BASIC programok változói számára foglaltak. A RAM ezt követő lapjai többnyire csak kerülő úton érhetők el.

A nulláslap első két byte-ja kiemelt szerepet tölt be, mert itt érhető el a processzor adatirány- és adatregisztere. Itt a biteknek külön-külön is van jelentésük, az egyes bitek értelmét a 10. táblázatban adtuk meg.

10. táblázat. MOS 6510 I/O portok

Cím#	CímH	Bit	Magyarázat
0000	0		Adatirány regiszter
		7-0	(XX101111)
			Bit : 1 = Output, 0 = Input, X = nem használt
0001	1		Mikroprocesszor port
		7-6	Nem definiált
		5	Kazettamotor ellenőrző 1 = kikapcsolva
		4	Kazetta állapotszenzor 1 = zárva
		3	Kazetta adatkimenet
		2	CHAREN jelző 1 = RAM D000-DFFF
		1	HIRAM jelző 1 = ROM E000-FFFF
		0	LORAM jelző 1 = ROM A000-BFFF

A nulláslap többi része az interpreter és az operációs rendszer változóit tárolja. A 10. táblázatban szereplő byte-tartalmak csak jellemző értékek, nem feltétlenül egyeznek meg azzal, amit a gép működése során tapasztaluk. A nulláslap jól elkülöníthetően két részre osztható: a BASIC interpreter által használt változókra és az operációs rendszer változóira. A BASIC-részben helyezkedik el a CHRGET rutin, amely a BASIC programok feldolgozásában jut szerephez. A nulláslap több változója csak bizonyos esetekben foglalt. Ilyenek pl. a kazettás egység kezeléséhez fenntartott változók vagy az RS232 változói. Ezeket a byte-okat, ha az operációs rendszer nem használja, nyugodtan alkalmazhatjuk saját programjainkban.

Az 1. lapot a processzor visszatérési veremként használja. A verem azonban nemcsak erre szolgál: a lap elején levő 11 byte a REAL—ASCII átalakítás során átmeneti tárolóként működik. Szalagos üzemmódban 63 byte szolgál a hibajavító rutin céljaira. Ez a terület szintén az 1. lap elején van. A BASIC interpreter a veremben tárolja a FOR—NEXT ciklushoz tartozó változókat és a GOSUB utasítás visszatérési mutatóit. Az interpreter és az operációs rendszer ezenkívül számtalan esetben tárolja az akkumulátor értékét a veremben. A verem a magasabb címek felől töltődik az alacsonyabb címek felé, az üres verembe az első érték a \$01FF abszolút címre kerül, a második a \$01FE-re és így tovább. Mivel a verembe utoljára beírt érték olvasáskor elsőként kerül ki, az angol terminológia erre a veremszervezésre a LI-FO (Last In—First Out: utoljára be—először ki) elnevezést alkalmazza. Magyarul zsáktárolónak is nevezik.

A verem programból is elérhető, nem védett. Más kérdés az, hogy beleírni nem tanácsos, mert a processzor könnyen határozatlan állapotba kerülhet.

11. táblázat. A nulláslap

Címke : Cím\$: Cím# : Tart. :					A byte rövid leírása
D6510	: \$0000	:	0: \$2C	:	6510 adatirány-regiszter (DDR)
R6510	: \$0001	:	1: \$37	:	6510 processzor port
-----	: \$0002	:	2: \$00	:	Nem használt
BASIC változók					
ADRAY1	: \$0003	:	3: \$AA	:	Vektor: lebegőpontos-egész konverzió (alsó byte)
"	: \$0004	:	4: \$B1	:	Vektor: lebegőpontos-egész konverzió (felső byte)
ADRAY2	: \$0005	:	5: \$91	:	Vektor: egész-lebegőpontos konverzió (alsó byte)
"	: \$0006	:	6: \$B3	:	Vektor: egész-lebegőpontos konverzió (felső byte)
CHARAC	: \$0007	:	7: \$22	:	Keresendő karakter
ENDCHR	: \$0008	:	8: \$22	:	Jelző: idézőjel mód
TRMPOS	: \$0009	:	9: \$00	:	Képernyőoszlop az utolsó TAB kiadása előtt
VERCKB	: \$000A	:	10: \$00	:	Jelző: 0=LOAD ; 1=VERIFY
COUNTB	: \$000B	:	11: \$4C	:	Mutató az input pufferben, a dimenziók száma
DIMFLG	: \$000C	:	12: \$00	:	Jelző: DIM utasítás
VALTYP	: \$000D	:	13: \$00	:	Jelző: 255=szöveges ; 0=aritmetikai
INTFLG	: \$000E	:	14: \$00	:	Jelző: 128=egész ; 0=valós
DATSCN	: \$000F	:	15: \$00	:	Az idézőjel jelzője a LIST-nél. A DATA jelzője
GARBFL	:	:	:	:	A személggyűjtő rutin jelzője
SUBFLG	: \$0010	:	16: \$00	:	FN kapcsoló
INPFLG	: \$0011	:	17: \$00	:	Jelző: 0=INPUT ; 64=GET ; 152=READ
TANSGN	: \$0012	:	18: \$00	:	Az ATN előjele, az egyenlőség jelzője összehas.nál
INPRMT	: \$0013	:	19: \$00	:	Az aktív I/O egység
LINNUM	: \$0014	:	20: \$14	:	Egész érték: SYS,GOTO,GOSUB sorszámai (alsó byte)
"	: \$0015	:	21: \$00	:	Egész érték: SYS,GOTO,GOSUB sorszámai (felső byte)
TEMPPT	: \$0016	:	22: \$19	:	Mutató: az ideiglenes szöveg a szövegveremben
LASTPT	: \$0017	:	23: \$16	:	Az utolsó ideiglenes szöveg címe (alsó byte)
"	: \$0018	:	24: \$00	:	Az utolsó ideiglenes szöveg címe (felső byte)
TEMPST	: \$0019	:	25: -	:	Szövegverem
"	: \$001A	:	26: -	:	Szövegverem
"	: \$001B	:	27: -	:	Szövegverem
"	: \$001C	:	28: -	:	Szövegverem
"	: \$001D	:	29: -	:	Szövegverem
"	: \$001E	:	30: -	:	Szövegverem
"	: \$001F	:	31: -	:	Szövegverem
"	: \$0020	:	32: -	:	Szövegverem
"	: \$0021	:	33: -	:	Szövegverem
INDEX	: \$0022	:	34: -	:	Felhasznált területmutatók
"	: \$0023	:	35: -	:	Felhasznált területmutatók
"	: \$0024	:	36: -	:	Felhasznált területmutatók
"	: \$0025	:	37: -	:	Felhasznált területmutatók
RESHO	: \$0026	:	38: \$00	:	Függvénykiértékelő és aritmetikai regiszter
"	: \$0027	:	39: \$00	:	Függvénykiértékelő és aritmetikai regiszter
"	: \$0028	:	40: \$00	:	Függvénykiértékelő és aritmetikai regiszter
"	: \$0029	:	41: \$00	:	Függvénykiértékelő és aritmetikai regiszter
"	: \$002A	:	42: \$00	:	Függvénykiértékelő és aritmetikai regiszter
TXTTAB	: \$002B	:	43: \$01	:	Mutató: a BASIC prg. eleje (alsó byte)
"	: \$002C	:	44: \$08	:	Mutató: a BASIC prg. eleje (felső byte)
VARTAB	: \$002D	:	45: \$03	:	Mutató: a BASIC prg. vége (alsó byte)
"	: \$002E	:	46: \$08	:	Mutató: a BASIC prg. vége (felső byte)

Címke : Cím# : Cím#:Tart.:				A byte rövid leírása	
ARYTAB	: \$002F	: 47:	\$03	Mutató: a BASIC tömbök eleje	(alsó byte)
*	: \$0030	: 48:	\$08	Mutató: a BASIC tömbök eleje	(fölső byte)
STREND	: \$0031	: 49:	\$03	Mutató: a BASIC-tömbök vége+1	(alsó byte)
*	: \$0032	: 50:	\$08	Mutató: a BASIC-tömbök vége+1	(fölső byte)
FRETOP	: \$0033	: 51:	\$00	Mutató: a szövegterület eleje	(alsó byte)
*	: \$0034	: 52:	\$A0	Mutató: a szövegterület eleje	(fölső byte)
FREPSC	: \$0035	: 53:	\$00	Az első fel nem használt szövegváltozó	(alsó byte)
*	: \$0036	: 54:	\$00	mutatója	(fölső byte)
MEMTOP	: \$0037	: 55:	\$00	Mutató: a BASIC munkaterület vége	(alsó byte)
*	: \$0038	: 56:	\$A0	Mutató: a BASIC munkaterület vége	(fölső byte)
CURLIN	: \$0039	: 57:	\$00	Mutató: a jelenlegi BASIC sor száma	(alsó byte)
*	: \$003A	: 58:	\$FF	Mutató: a jelenlegi BASIC sor száma	(fölső byte)
OLDLIN	: \$003B	: 59:	\$00	Mutató: az előző BASIC sor száma	(alsó byte)
*	: \$003C	: 60:	\$00	Mutató: az előző BASIC sor száma	(fölső byte)
OLDTXT	: \$003D	: 61:	\$00	Mutató a CONT végrehajtásához	(alsó byte)
*	: \$003E	: 62:	\$00	Mutató a CONT végrehajtásához	(fölső byte)
DATLIN	: \$003F	: 63:	\$00	A feldolgozás alatt álló DATA-sor száma	(alsó byte)
*	: \$0040	: 64:	\$00	A feldolgozás alatt álló DATA-sor száma	(fölső byte)
DATPTR	: \$0041	: 65:	\$00	A feldolgozás alatt álló DATA byte mutatója	(alsó byte)
*	: \$0042	: 66:	\$08	A feldolgozás alatt álló DATA byte mutatója	(fölső byte)
INPPTR	: \$0043	: 67:	\$01	Vektor: a bevitel forrásának mutatója	(alsó byte)
*	: \$0044	: 68:	\$02	Vektor: a bevitel forrásának mutatója	(fölső byte)
VARNAM	: \$0045	: 69:	-	Jelenlegi BASIC változó neve	(első byte)
*	: \$0046	: 70:	-	Jelenlegi BASIC változó neve	(második byte)
VARPNT	: \$0047	: 71:	-	Mutató: a BASIC változó helye	(alsó byte)
*	: \$0048	: 72:	-	Mutató: a BASIC változó helye	(fölső byte)
FORPNT	: \$0049	: 73:	-	Mutató: a változók értékének mutatója	(alsó byte)
*	: \$004A	: 74:	-	Mutató: a változók értékének mutatója	(fölső byte)
EASTMP	: \$004B	: 75:	-	A programmutató átmeneti tára	(alsó byte)
*	: \$004C	: 76:	-	A programmutató átmeneti tára	(fölső byte)
*	: \$004D	: 77:	-	Az összehasonlító műveletek maszkja	
*	: \$004E	: 78:	-	Mutató az FN-re	(alsó byte)
*	: \$004F	: 79:	-	Mutató az FN-re	(fölső byte)
*	: \$0050	: 80:	-	Szövegleíró	
*	: \$0051	: 81:	-	Szövegleíró	
*	: \$0052	: 82:	-	Szövegleíró	
*	: \$0053	: 83:	-	Szövegleíró	
*	: \$0054	: 84:	\$4C	Konstans \$4C, ugrás (JMP) a függvényre	
*	: \$0055	: 85:	-	A függvény ugrási vektora	(alsó byte)
*	: \$0056	: 86:	-	A függvény ugrási vektora	(fölső byte)
*	: \$0057	: 87:	\$00	Aritmetikai regiszter, akku #3	
*	: \$0058	: 88:	\$00	Aritmetikai regiszter, akku #3	
*	: \$0059	: 89:	\$00	Aritmetikai regiszter, akku #3	
*	: \$005A	: 90:	\$00	Aritmetikai regiszter, akku #3	
*	: \$005B	: 91:	\$00	Aritmetikai regiszter, akku #3	
*	: \$005C	: 92:	\$00	Aritmetikai regiszter, akku #4	
*	: \$005D	: 93:	\$00	Aritmetikai regiszter, akku #4	
*	: \$005E	: 94:	\$00	Aritmetikai regiszter, akku #4	
*	: \$005F	: 95:	\$00	Aritmetikai regiszter, akku #4	
*	: \$0060	: 96:	\$00	Aritmetikai regiszter, akku #4	

Címke : Cím\$: Cím# : Tart.:				A byte rövid leírása	
FACEXP	: \$0061	: 97:	\$00	: Lebegőpontos akkumulátor #1	exponens
FACHO	: \$0062	: 98:	\$00	: Lebegőpontos akkumulátor #1	mantissza
"	: \$0063	: 99:	\$00	: Lebegőpontos akkumulátor #1	mantissza
"	: \$0064	: 100:	\$00	: Lebegőpontos akkumulátor #1	mantissza
"	: \$0065	: 101:	\$00	: Lebegőpontos akkumulátor #1	mantissza
FACSGN	: \$0066	: 102:	\$00	: Lebegőpontos akkumulátor #1	előjel
SGNFLG	: \$0067	: 103:	\$00	: Számláló a polinom kiértékeléséhez (pl. \$E059)	
BITS	: \$0068	: 104:	\$00	: Lebegőpontos akkumulátor #1	túlcordulás jelző
ARGEXP	: \$0069	: 105:	\$00	: Lebegőpontos akkumulátor #2	exponens
ARGHO	: \$006A	: 106:	\$00	: Lebegőpontos akkumulátor #2	mantissza
"	: \$006B	: 107:	\$00	: Lebegőpontos akkumulátor #2	mantissza
"	: \$006C	: 108:	\$00	: Lebegőpontos akkumulátor #2	mantissza
"	: \$006D	: 109:	\$00	: Lebegőpontos akkumulátor #2	mantissza
ARGSGN	: \$006E	: 110:	\$00	: Lebegőpontos akkumulátor #2	előjel
ARISGN	: \$006F	: 111:	\$00	: A két akku.előjelének összehasonlítása, 0=egyenlő	
FACOV	: \$0070	: 112:	\$00	: Lebegőpontos akkumulátor #1	low-order (Kerekítés)
FBUFPT	: \$0071	: 113:	-	: Mutató a polinom kiértékelésére	(alsó byte)
"	: \$0072	: 114:	-	: Mutató a polinom kiértékelésére	(felső byte)
CHRGET	: \$0073	: 115:	\$E6	: A CHRGET-rutin kezdőcíme	
"	: \$0074	: 116:	\$7A	: 0073 INC \$7A	; CHRGET
"	: \$0075	: 117:	\$D0	: 0075 BNE \$0079	
"	: \$0076	: 118:	\$02	: 0077 INC \$7B	
"	: \$0077	: 119:	\$E6	: 0079 LDA \$0000	; CHRGOT: a byte az akkuban
"	: \$0078	: 120:	\$7B	: 007C CMP #\$3A	; ":" ? ha Kettőspont, => RTS
CHRGOT	: \$0079	: 121:	\$AD	: 007E BCS \$008A	
TXTPTR	: \$007A	: 122:	-	: Programszámláló	(alsó byte)
"	: \$007B	: 123:	-	: Programszámláló	(felső byte)
CHRGET	: \$007C	: 124:	\$C9	: 0080 CMP #\$20	; " " ? a szóköz átlépése
"	: \$007D	: 125:	\$3A	: 0082 BEQ \$0073	
"	: \$007E	: 126:	\$B0	: 0084 SEC	
"	: \$007F	: 127:	\$0A	: 0085 SBC #\$30	
"	: \$0080	: 128:	\$C9	: 0087 SEC	
"	: \$0081	: 129:	\$20	: 0088 SBC #\$D0	
"	: \$0082	: 130:	\$F0	: 008A RTS	
"	: \$0083	: 131:	\$EF	: A CHRGET rutin a ROM-ban található \$E3A2-től,	
"	: \$0084	: 132:	\$38	: de a RESET alatt a RAM-ba másolódik.	
"	: \$0085	: 133:	\$E9	: Csak a RAM-ban működik.	
"	: \$0086	: 134:	\$30	: Az X és az Y regisztert nem használja.	
"	: \$0087	: 135:	\$38	: Ha a CARRY=0, => az akku tartalma \$30-\$39	
"	: \$0088	: 136:	\$E9	: közé esik.	
"	: \$0089	: 137:	\$D0	: Ha a ZERO=0, => az akku tartalma vagy \$00,	
"	: \$008A	: 138:	\$60	: vagy \$3A (sorvég vagy Kettőspont).	
RNDX	: \$008B	: 139:	-	: Az utolsó RND érték	
"	: \$008C	: 140:	-	: Az utolsó RND érték	
"	: \$008D	: 141:	-	: Az utolsó RND érték	
"	: \$008E	: 142:	-	: Az utolsó RND érték	
"	: \$008F	: 143:	-	: Az utolsó RND érték	

Címke : Cím# : Cím# : Tart. :				A byte rövid leírása

KERNAL változók				
STATUS	: \$0090	: 144:	\$00	: I/O állapotszó (BASIC ST)
STKEY	: \$0091	: 145:	\$FF	: Jelző: STOP billentyű/RVS billentyű
SVXT	: \$0092	: 146:	\$00	: Időzítési Konstans a Kazettás egységhez
VERCK	: \$0093	: 147:	\$00	: Jelző: 0=LOAD ; 1=VERIFY, a LOAD rutin állítja be
C3PO	: \$0094	: 148:	\$00	: Jelző: IEC busz output puffere, 0=üres, 128=megtelt
BSOUR	: \$0095	: 149:	\$00	: Az IEC busz output puffere
SYND	: \$0096	: 150:	\$00	: Jelző: EOT vétele szalagról (End of Tape)
XSAV	: \$0097	: 151:	\$00	: A regiszterek közbelső tára
LOTND	: \$0098	: 152:	\$00	: Nyitott file-ok száma
DFTN	: \$0099	: 153:	\$00	: Elsődleges input eszköz (0=Klaviatúra)
DFTO	: \$009A	: 154:	\$03	: Elsődleges output eszköz CMD (3=Képernyő)
PRTY	: \$009B	: 155:	\$00	: A szalag paritásbyte-ja
DPSW	: \$009C	: 156:	\$00	: Jelző: a byte vétele megtörtént.
MSGFLG	: \$009D	: 157:	\$80	: Jelző: \$80=direkt mód ; \$00=program mód
PTR1	: \$009E	: 158:	\$00	: Szalag: 1 menet, ellenőrzőösszeg
T1	:	:	:	: 4tmeneti tároló 1
PTR2	: \$009F	: 159:	\$00	: Szalag: 2 menet, hibajavítás
T2	:	:	:	: 4tmeneti tároló 2
TIME	: \$00A0	: 160:	-	: Valós idejű óra
*	: \$00A1	: 161:	-	: Valós idejű óra
*	: \$00A2	: 162:	-	: Valós idejű óra
R2D2	: \$00A3	: 163:	\$00	: A soros busz használja
PCNTR	:	:	:	: A szalagos üzemmód számlálója
BSOUR1	: \$00A4	: 164:	\$00	: Ciklusszámláló (ha \$A3=8, => \$A4=0)
CNTDN	: \$00A5	: 165:	\$00	: Szinkronjel visszaszámláló íráshoz
COUNT	:	:	:	: A soros busz rutinok használják
BUFPT	: \$00A6	: 166:	\$00	: Szalag I/O puffer pointer
INBIT	: \$00A7	: 167:	\$00	: Az RS232 input-bit ideiglenes tárolása
SHCNL	:	:	:	: Szalag: számláló
BITC1	: \$00A8	: 168:	\$00	: A vevő bit számlálása
RER	:	:	:	: Szalag: olvasási hiba
RINONE	: \$00A9	: 169:	\$00	: INPUT-jelző START-bit ellenőrzése
REZ	:	:	:	: Szalag: nullák beolvasva
RIDATA	: \$00AA	: 170:	\$00	: INPUT-puffer byte puffereelési/szerkesztési célokra
ROFLG	:	:	:	: Szalag: olvasás mód
RIPRTY	: \$00AB	: 171:	\$00	: INPUT-paritásbit tárolása
SHCNH	:	:	:	: Szalag: számláló
SAL	: \$00AC	: 172:	\$00	: A szalagpuffer és a scroll mutatója (alsó byte)
SAH	: \$00AD	: 173:	\$00	: A szalagpuffer és a scroll mutatója (felső byte)
EAL	: \$00AE	: 174:	\$00	: A program vége, mutató a LOAD/SAVE utasításnál
EAH	: \$00AF	: 175:	\$00	: A program vége, mutató a LOAD/SAVE utasításnál
CMPO	: \$00B0	: 176:	\$00	: Kazettás egység időzítési állandó (alsó byte)
TEMP	: \$00B1	: 177:	\$00	: Kazettás egység időzítési állandó (felső byte)
TAPE1	: \$00B2	: 178:	-	: Szalagpuffer Kezdet (alsó byte)
TAPE1	: \$00B3	: 179:	-	: Szalagpuffer Kezdet (felső byte)
BITTS	: \$00B4	: 180:	\$00	: Output-bit számlálása
NXTBIT	: \$00B5	: 181:	\$00	: Output következő küldendő bitje
RODATA	: \$00B6	: 182:	\$00	: Output byte puffer/szétbontási hely
FNLEN	: \$00B7	: 183:	\$00	: Az aktuális file-név hossza

Címke :	Cím\$:	Cím# :	Tart. :	A byte rövid leírása	
LA	: \$00B8	: 184:	\$00	:Az aktuális logikai file-szám	
SA	: \$00B9	: 185:	\$00	:Az aktuális file másodlagos címe	
FA	: \$00BA	: 186:	\$00	:Az aktuális file egységszáma	
FNADR	: \$00BB	: 187:	\$00	:Mutató: az aktuális file-név	(alsó byte)
"	: \$00BC	: 188:	\$00	:Mutató: az aktuális file-név	(felső byte)
ROPRTY	: \$00BD	: 189:	\$00	:Output paritásbit tárolása	
FSBLK	: \$00BE	: 190:	\$00	:Kazetta READ/WRITE blokkszámláló	
MYCH	: \$00BF	: 191:	\$00	:A soros busz output puffere	
CASI	: \$00C0	: 192:	\$00	:Jelző: motor bekapcsolva	
STAL	: \$00C1	: 193:	\$00	:A képernyős input/output kezdőcíme	(alsó byte)
STAH	: \$00C2	: 194:	\$A0	:A képernyős input/output kezdőcíme	(felső byte)
MEMUSS	: \$00C3	: 195:	\$30	:A képernyős input/output végcíme	(alsó byte)
"	: \$00C4	: 196:	\$FD	:A képernyős input/output végcíme	(felső byte)

A képernyőszerkesztő változói

LSTX	: \$00C5	: 197:	\$00	:A lenyomott billentyű: 64=nincs lenyomva	
NDX	: \$00C6	: 198:	\$00	:A billentyűzet pufferban tárolt karakterek száma	
RVS	: \$00C7	: 199:	\$00	:Jelző: REVERSE-írásmód: 1=igen;0=nem	
INDX	: \$00C8	: 200:	\$00	:A sor vége,bevitelkor	
LSXP	: \$00C9	: 201:	\$00	:A kurzor X pozíciója az INPUT számára	
LSTP	: \$00CA	: 202:	\$00	:A kurzor Y pozíciója az INPUT számára	
SFDX	: \$00CB	: 203:	\$01	:Lenyomott billentyű:64=nincs lenyomva	
BLNSW	: \$00CC	: 204:	\$00	:A kurzor engedélyezése: 0=van kurzor ; 1=nincs	
BLNCT	: \$00CD	: 205:	\$02	:A kurzorvillogás számlálója	
GDBLN	: \$00CE	: 206:	\$20	:A kurzor alatti karakter; CHR\$(n)	
BLNON	: \$00CF	: 207:	\$00	:Jelző: 1=Kurzor látható fázisban	
CRSW	: \$00D0	: 208:	\$00	:Jelző: INPUT vagy GET történik?	
PNT	: \$00D1	: 209:	\$28	:Az aktuális képernyősor címe	(alsó byte)
"	: \$00D2	: 210:	\$04	:Az aktuális képernyősor címe	(felső byte)
PNTR	: \$00D3	: 211:	\$00	:A kurzor aktuális oszloppozíciója	
QTSW	: \$00D4	: 212:	\$00	:A szerkesztő "*" módban? 0=nem	
LNMX	: \$00D5	: 213:	\$27	:A fizikai képernyősor hossza	
TBLX	: \$00D6	: 214:	\$01	:A kurzor aktuális sorpozíciója	
DATA	: \$00D7	: 215:	-	:Egyéb célokra	
INSRT	: \$00D8	: 216:	\$00	:Jelző: az INSERT-módban beszúrando karakterek száma	
LDTB1	: \$00D9	: 217:	\$84	:A képernyősor kezdetének MSB-je	
"	: \$00DA	: 218:	\$84	:A képernyősor kezdetének MSB-je	
"	: \$00DB	: 219:	\$84	:A képernyősor kezdetének MSB-je	
"	: \$00DC	: 220:	\$84	:A képernyősor kezdetének MSB-je	
"	: \$00DD	: 221:	\$84	:A képernyősor kezdetének MSB-je	
"	: \$00DE	: 222:	\$84	:A képernyősor kezdetének MSB-je	
"	: \$00DF	: 223:	\$84	:A képernyősor kezdetének MSB-je	
"	: \$00E0	: 224:	\$85	:A képernyősor kezdetének MSB-je	
"	: \$00E1	: 225:	\$85	:A képernyősor kezdetének MSB-je	
"	: \$00E2	: 226:	\$85	:A képernyősor kezdetének MSB-je	
"	: \$00E3	: 227:	\$85	:A képernyősor kezdetének MSB-je	
"	: \$00E4	: 228:	\$85	:A képernyősor kezdetének MSB-je	
"	: \$00E5	: 229:	\$85	:A képernyősor kezdetének MSB-je	
"	: \$00E6	: 230:	\$86	:A képernyősor kezdetének MSB-je	
"	: \$00E7	: 231:	\$86	:A képernyősor kezdetének MSB-je	
"	: \$00E8	: 232:	\$86	:A képernyősor kezdetének MSB-je	

Címke :	Cím# :	Cím#:	Tart.:	A byte rövid leírása	
LDTB1	: \$00E9 :	233:	\$86 :	A képernyősor kezdetének MSB-je	
"	: \$00EA :	234:	\$86 :	A képernyősor kezdetének MSB-je	
"	: \$00EB :	235:	\$86 :	A képernyősor kezdetének MSB-je	
"	: \$00EC :	236:	\$87 :	A képernyősor kezdetének MSB-je	
"	: \$00ED :	237:	\$87 :	A képernyősor kezdetének MSB-je	
"	: \$00EE :	238:	\$87 :	A képernyősor kezdetének MSB-je	
"	: \$00EF :	239:	\$87 :	A képernyősor kezdetének MSB-je	
"	: \$00F0 :	240:	\$87 :	A képernyősor kezdetének MSB-je	
"	: \$00F1 :	241:	\$87 :	A képernyősor kezdetének MSB-je	
LINTMP	: \$00F2 :	242:	\$87 :	A sorindex átmeneti tárolója	
USER	: \$00F3 :	243:	\$01 :	Mutató :a COLOR-RAM helye	(alsó byte)
"	: \$00F4 :	244:	\$08 :	Mutató :a COLOR-RAM helye	(felső byte)
KEYTAB	: \$00F5 :	245:	\$81 :	Billentyűzet-mátrix dekódoló tár címe	(alsó byte)
"	: \$00F6 :	246:	\$EB :	Billentyűzet-mátrix dekódoló tár címe	(felső byte)

Az RS232 változói

RIBUF	: \$00F7 :	247:	\$00 :	RS232 input puffer mutató	(alsó byte)
"	: \$00F8 :	248:	\$00 :	RS232 input puffer mutató	(felső byte)
ROBUF	: \$00F9 :	249:	\$00 :	RS232 output puffer mutató	(alsó byte)
"	: \$00FA :	250:	\$00 :	RS232 output puffer mutató	(felső byte)
FREKZP	: \$00FB :	251:	\$00 :	Nem használt	
"	: \$00FC :	252:	\$00 :	Nem használt	
"	: \$00FD :	253:	\$00 :	Nem használt	
"	: \$00FE :	254:	\$00 :	Nem használt	
BASZPT	: \$00FF :	255:	- :	REAL/ASCII átalakítás esetén használt	

A 2. lap az interpreter INPUT-pufferét, a nyitott file-ok adatait, néhány rendszerváltozót, a képszerkesztő és az RS232 néhány változóját tárolja. Említést érdemel a képernyőszerkesztő billentyűzetpuffere, amely a másik veremszervezési módszert példázza. Az ide elsőként bekerülő byte elsőként is távozik. Az ilyen szervezési elvet a FIFO (First In — First Out: először be — először ki) elnevezés mutatja. A magyar megfelelő a silótároló. A puffer folyamatosan balra tömörített; a tömörítésről az IRQ megszakítást kezelő rutin gondoskodik.

A lap végén 87 szabad byte található. Itt a 7. sprite adatai helyezhetők el, vagy felhasználhatók más programozási célra.

A 3. lapon az operációs rendszer és az interpreter egyes rutinjainak indirekt vektorai található. Ezek a vektorok nyújtanak lehetőséget a BASIC interpreter és a KERNAL operációs rendszer bővítésére vagy módosítására. A lap végén a 192 byte-os kazettapuffer található, de ez általában szabad.

A C64-esben még egy RAM áramkör van, a COLOR RAM vagy *színtár*, amelyben a VIC a karakterekhez tartozó színt tárolja. Ezt a RAM tárolót címzés szempontjából a VIC külön kezeli. Helye a címtartományban rögzített, mindig a \$D800 (56296) címen kezdődik.

A C64-es hardvere által előállított 16-féle szín azonosításához elég egy 4 bites szám is, emiatt a COLOR RAM is négybites szervezésű. A tárcímek felső félbyte-ja (nibble) határozatlan. Az áramkört az AM szabályos külső egységként vezérli.

12. táblázat. A memória további részei

Címke : A terület címe (Tart.): Megjegyzés			
-----	:\$00FF-010A:	255-	266: Lebegőpontos-ASCII átalakítás munkaterülete
BAD	:\$0100-013E:	256-	318: A szalagolvasás hibajavító puffere
-----	:\$0100-01FF:	256-	511: A processzor visszatérési verme
BUF	:\$0200-0258:	512-	600: Az interpreter INPUT-puffere
A nyitott file-ok táblázatai			
LAT	:\$0259-0262:	601-	610: KERNAL táblázat: aktív logikai file-számok
FAT	:\$0263-026C:	611-	620: KERNAL táblázat: az egységszámok
SAT	:\$026D-0276:	621-	630: KERNAL táblázat: a file-ok másodlagos címei
Rendszerváltozók			
KEYD	:\$0277-0280:	631-	640: Billentyűzet puffer (FIFO)
MEMSTR	:\$0281	:	641: \$00: Memória-kezdet
"	:\$0282	:	642: \$08: az operációs rendszer számára
MEMSI2	:\$0283	:	643: \$00: Memória-tető
"	:\$0284	:	644: \$A0: az operációs rendszer számára
TIMOUT	:\$0285	:	645: \$00: A soros timeout jelzője
A Képszerkesztő változói			
COLOR	:\$0286	:	646: \$0E: Az aktuális karakter színkódja
GDCOL	:\$0287	:	647: \$0E: A kurzor alatti háttér színkódja
HIBASE	:\$0288	:	648: \$04: A képernyőmemória kezdőcíme (lap)
XMAX	:\$0289	:	649: \$0A: A billentyűzet puffer hossza (max. 10)
RPTFLG	:\$028A	:	650: \$00: \$80=mindegyik billentyű ismétel, \$40:egyik sem
KOUNT	:\$028B	:	651: - : Ismétlési sebesség számláló
DELAY	:\$028C	:	652: - : Ismétlési kitértés számláló
SHFLAG	:\$028D	:	653: \$00: Jelző: \$01=SHIFT; \$02=Commodore; \$04=Control
LSTSHF	:\$028E	:	654: \$00: A SHIFT jelzője
KEYLOG	:\$028F	:	655: \$48: Vektor: A billentyűzet
"	:\$0290	:	656: \$EB: táblázat kiválasztása
MODE	:\$0291	:	657: \$00: \$00=SHIFT engedélyezve ; \$80=SHIFT letiltva
AUTODN	:\$0292	:	658: \$00: Jelző: Auto scroll ; 0=engedélyezve
RS232 változók			
M26CTR	:\$0293	:	659: - : RS232: vezérlőregiszter
M26CDR	:\$0294	:	660: - : RS232: parancsregiszter
M26AJB	:\$0295	:	661: - : Nem standard
"	:\$0296	:	662: - : átviteli sebesség
RSSTAT	:\$0297	:	663: - : RS232: státuszregiszter
BITNUM	:\$0298	:	664: - : Az adatbitek száma RS232-nél
BAUDOF	:\$0299	:	665: - : RS232: átviteli
"	:\$029A	:	666: - : sebesség
RIDBE	:\$029B	:	667: - : Mutató: Az RS232-esen fogadott byte
RIDBS	:\$029C	:	668: - : RS232-es input
RODBS	:\$029D	:	669: - : Mutató: Az RS232-esen továbbítandó byte
RODBE	:\$029E	:	670: - : RS232-es output

Címke : A terület címe (Tart.): Megjegyzés

```

IRQTMP :$029F      : 671: - : Szalagüzem mód alatti
*       :$02A0      : 672: - :                               IRQ tárolása
----- :$02A1      : 673: - : NMI megszakítás jelző
----- :$02A2      : 674: - : Jelző: CIA1 A időzítő vezérlés
----- :$02A3      : 675: - : Jelző: CIA1, megszakítás
----- :$02A4      : 676: - : Jelző: Az A időzítő foglalt
----- :$02A5      : 677: - : Képernyő sor-marker
----- :$02A6      : 678: 1 : Jelző: 1=PAL, 0=NTSC

```

----- :\$02A7-02FF: 679- 767: Nem használt terület

A BASIC interpreter vektorai

```

IERROR :$0300      : 768- 769: Hibauzenet küldése           : $E3B8
IMAIN   :$0302      : 770- 771: BASIC melegstart          : $A483
ICRNCH  :$0304      : 772- 773: Átalakítás interpreter kóddá : $A57C
IQPLOP  :$0306      : 774- 775: LIST rutin                 : $A71A
IGONE   :$0308      : 776- 777: BASIC utasításcímbeolvasás  : $A7E4
IEVAL   :$030A      : 778- 779: A kifejezés kiértékelése     : $AE86

```

Regiszterek a SYS utasítás számára

```

SAREG   :$030C      : 780: $00: Akkumulátor
SXREG   :$030D      : 781: $00: X regiszter
SYREG   :$030E      : 782: $00: Y regiszter
SPREG   :$030F      : 783: $00: Státuszregiszter

```

Az USR függvény címe

```

USRPOK  :$0310      : 784: : $4C-JMP az USR függvényre
USRADD  :$0311-0312: 785- 786: Az USR függvény címe       : $B248
----- :$0313      : 787: - : Nem használt

```

Az operációs rendszer vektorai

```

CINV    :$0314-0315: 788- 789: Hardware megszakító rutin   : $EA31
CBINV   :$0316-0317: 790- 791: BRK megszakító rutin        : $FE66
NMINV   :$0318-0319: 792- 793: Nem maszkolható megszakítás : $FE47
IOPEN   :$031A-031B: 794- 795: KERNAL OPEN rutin           : $F34A
ICLOSE  :$031C-031D: 796- 797: KERNAL CLOSE rutin          : $F291
ICLKIN  :$031E-031F: 798- 799: KERNAL CHKIN rutin          : $F20E
ICKOUT  :$0320-0321: 800- 801: KERNAL CHKOUT rutin         : $F250
ICLRCH  :$0322-0323: 802- 803: KERNAL CLRCHN rutin         : $F333
IBASIN  :$0324-0325: 804- 805: KERNAL CHRIN rutin          : $F157
IBSOUT  :$0326-0327: 806- 807: KERNAL CHROUT rutin         : $F1CA
ISTOP   :$0328-0329: 808- 809: KERNAL STOP rutin           : $F6ED
IGETIN  :$032A-032B: 810- 811: KERNAL GETIN rutin          : $F13E
ICLALL  :$032C-032D: 812- 813: KERNAL CLALL rutin          : $F32F
USRCMD  :$032E-032F: 814- 815: A STOP/RESTORE vektora      : $FE66
ILOAD   :$0330-0331: 816- 817: KERNAL LOAD rutin           : $F4A5
ISAVE   :$0332-0333: 818- 819: KERNAL SAVE rutin           : $F5ED

```

 Címke : A terület címe (Tart.): Megjegyzés

Kazettapuffer

----- :\$0334-033B: 820- 827: Nem használt terület
 TBUFFR :\$033C-03FB: 828- 1019: Kazettapuffer
 ----- :\$03FC-03FF: 1020- 1023: Nem használt terület

Képernyőmemória

VICSCN :\$0400-07FF: 1024- 2047: 1 kbyte Képernyőmemória
 ----- :\$0400-07E7: 1024- 2023: 1000 byte video mátrix
 ----- :\$07E8-07F7: 2024- 2039: Nem használt terület
 ----- :\$07F8-07FF: 2040- 2047: A 0.-7. sprite-ok területmutatói

A BASIC programtár

----- :\$0800-9FFF: 2048-40959: BASIC programtár, változóterületek
 ----- :\$8000-9FFF:32768-40959: A bővítőportba helyezett ROM

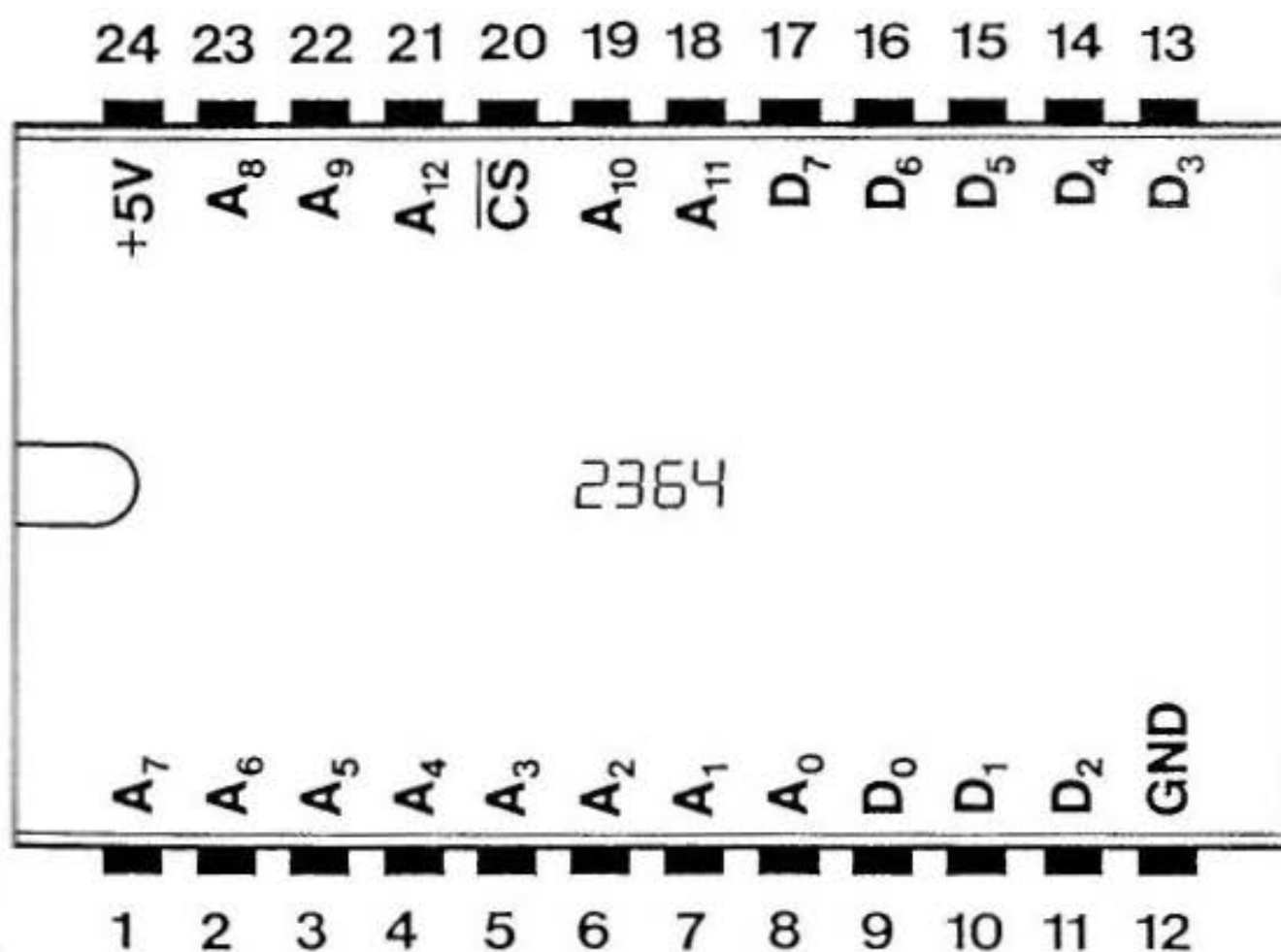
A BASIC és a KERNAL ROM

BASIC :\$A000-BFFF:40960-49151: A BASIC ROM
 ----- :\$C000-CFFF:49152-53247: Nem használt RAM
 CHAREN :\$D000-DFFF:53248-57343: A Karaktergenerátor ROM
 ----- :\$D000-DFFF:53248-57343: Az I/O chipek regiszterei és a színtár
 VIC :\$D000-D02E:53248-53294: A VIC regiszterei
 SID :\$D400-D41C:54272-54300: A SID regiszterei
 COLOR :\$D800-DBE7:55296-56295: A színtár - 1000 byte
 CIA1 :\$DC00-DC0F:56320-56335: A CIA1 regiszterei
 CIA2 :\$DD00-DD0F:56576-56591: A CIA2 regiszterei
 I/O1 :\$DE00-DEFF:56832-57087: I/O1, vagy nem használt terület
 I/O2 :\$DF00-DFFF:57088-57343: I/O2, vagy nem használt terület
 KERNAL :\$E000-FFFF:57344-65535: A KERNAL ROM

1.6.2. Csak olvasható memória: ROM

A C64-esben található három ROM chip mindegyike egyszer programozható fix tár, amelyeket még a gyártó programoz. A \$A000—BFFF címterületet lefedő ROM neve BASIC. Kapacitása 64 kbit, szervezése 8k x 8 bit. A \$E000—FFFF címtartományt fedő ROM neve KERNAL. Kapacitása és szervezése az előzővel azonos. A \$D000—DFFF címterületen helyezkedik el a CHAREN, amely a karakterek bitmintáit tárolja. Kapacitása 32 kbit, szervezése

4k x 8 bit. Az áramkörök 24 lábú DIL-tokozásban kerülnek f orgalomba. A BASIC és KERNAL tokozása, lábkiosztása megegyezik.



23. ábra. A ROM chip

- 1—8. A₇-A₀: címbitek
- 9—11. D₀-D₂: adatbitek
- 12. GND: föld
- 13—17. D₃ D₇: adatbitek
- 18—19. A₁₁ A₁₀: címbitek
- 20. CS, Chip Select: kiválasztó vezérlővonal
- 21—23. A₁₂-A₈: címbitek
- 24. Tápfeszültség

A CHAREN tokozása szintén 24 lábú DIL, a lábkiosztás is megegyezik, egy kivétellel: a tok 21. lába nem a 12. címbitet fogadja, hanem állandó jelleggel elő van feszítve +5 V-tal. A chipek CS jelű 20. lábára az AM megfelelő vezérlővonalra van kötve, KERNAL, BASIC, CHAROM. E vonalak alacsony állapota esetén az olvasási kísérlet a ROM-ok felé irányul.

A BASIC az interpreter, a KERNAL az operációs rendszer gépi nyelvű programját tartalmazza. Mindkettő lényegileg szubrutinok gyűjteménye, különösen a KERNAL. A két ROM chip programjának fontosabb belépési pontjait táblázatba rendezve ismertetjük, megadva a szubrutinok rövid megnevezését is.

A BASIC interpreter rutinjai

Cím\$: Cím# : Megjegyzés

A000 : 40960 : BASIC Hidegstartvektor (\$E394)
 A002 : 40962 : BASIC NMI vektor (\$E37B)
 A004 : 40964 : A ROM modul azonosítója: "CBMBASIC"
 A00C : 40972 : A BASIC utasítások címei-1
 A052 : 41042 : A BASIC függvények címei-1
 A080 : 41988 : A BASIC műveletek címei, és az elsőbbségi kódok
 A09E : 41118 : Az utasítások listája
 A19E : 41374 : BASIC hibüzenetek
 A328 : 41768 : A hibüzenetek címei
 A364 : 41828 : Az interpreter üzenetei (OK,ERROR,IN,READY,BREAK)
 A38A : 41866 : A FOR-NEXT utasítások változóinak keresése a veremben
 A3B8 : 41912 : Blokk-léptető rutin
 A3FB : 41979 : Szabad hely keresése a veremben
 A408 : 41992 : Helyfoglalás a memóriában
 A435 : 42037 : OUT OF MEMORY hibüzenet
 A437 : 42039 : JMP \$E38B - indirekt ugrás a hibüzenet-rutinra
 A43A : 42042 : Hibüzenetkezelő rutin
 A469 : 42083 : Break-belépési pont
 A474 : 42100 : Ready-belépési pont
 A480 : 42112 : BASIC melegstart. Várakozás az utasításra
 A49C : 42140 : Programsor törlése és/vagy beiktatása
 A4A9 : 42153 : Programsor törlése
 A4ED : 42221 : Programsor beiktatása
 A533 : 42291 : A BASIC programsorok újraláncolása
 A560 : 42336 : Egy sor beolvasása az INPUT-pufferbe
 A571 : 42353 : STRING TOO LONG hibüzenet
 A579 : 42361 : Egy sor átalakítása interpreter kóddá
 A613 : 42515 : Egy programsor kezdőcímének kiszámítása
 A642 : 42562 : A NEW utasítás
 A65E : 42590 : A CLR utasítás
 A68E : 42638 : CHRGET programszámláló a BASIC-kezdetre
 A69C : 42652 : A LIST utasítás
 A717 : 42775 : BASIC-kód átalakítása szöveggé
 A742 : 42818 : A FOR utasítás
 A7AE : 42926 : Az interpreter-ciklus (utasításvégrehajtás)
 A7ED : 42989 : A BASIC utasítás végrehajtása
 A807 : 43015 : Kettőspont ellenőrzése
 A80E : 43022 : GO & TO kód ellenőrzése
 A81D : 43037 : A RESTORE utasítás
 A82C : 43052 : A STOP billentyű ellenőrzése
 A82F : 43055 : A STOP utasítás
 A831 : 43057 : Az END utasítás
 A857 : 43095 : A CONT utasítás
 A859 : 43097 : CAN'T CONTINUE hibüzenet
 A871 : 43121 : A RUN utasítás
 A883 : 43139 : A GOSUB utasítás
 A8A0 : 43168 : A GOTO utasítás
 A8D2 : 43218 : A RETURN utasítás
 A8E0 : 43232 : RETURN WITHOUT GOSUB hibüzenet
 A8E3 : 43235 : UNDEFN'D STATEMENT hibüzenet
 A8F8 : 43256 : A DATA utasítás
 A906 : 43270 : A következő utasítás relatív címének megkeresése

Cím# : Cím# : Megjegyzés

```

A328 : 43304 : Az IF utasítás
A338 : 43323 : A REM utasítás
A348 : 43339 : Az ON utasítás
A368 : 43371 : A sorszám betöltése a $14-15 címekre
A3A5 : 43429 : A LET utasítás
A3C4 : 43460 : Egész érték kijelölés
A3D6 : 43478 : Valós érték kijelölés
A3D9 : 43481 : Szöveges érték kijelölés
AA1D : 43549 : Egy karakter numerikus ellenőrzése
AA2C : 43564 : Szöveges értékadás
AA80 : 43648 : A PRINT# utasítás
AA86 : 43654 : A CMD utasítás
AA80 : 43680 : A PRINT utasítás
AAF8 : 43768 : TAB (C=1) és SPC (C=0)
AB1E : 43806 : Szöveg kiírása, mutató az A/Y-ban
AB3B : 43835 : Egy kurzor jobbra vagy szóköz kiírása
AB45 : 43845 : Egy kérdőjel kiírása
AB4D : 43853 : Hibakezelés input műveleteknél
AB57 : 43863 : Hiba READ közben
AB5B : 43867 : Hiba GET közben
AB62 : 43874 : Hiba INPUT közben
AB66 : 43878 : FILE DATA hibaüzenet
AB7B : 43899 : A GET utasítás
ABA5 : 43941 : Az INPUT# utasítás
ABBF : 43967 : Az INPUT utasítás
ABF9 : 44025 : A beolvasás végrehajtása
AC06 : 44038 : A READ utasítás
ACBF : 44223 : OUT OF DATA hibaüzenet
ACFC : 44284 : Az INPUT üzenetei
AD1E : 44318 : A NEXT utasítás
AD30 : 44336 : NEXT WITHOUT FOR hibaüzenet
AD8A : 44426 : FRMNUM, numerikus kifejezés beolvasása
AD8D : 44429 : Numerikus ellenőrzés
AD8F : 44431 : Karakteres ellenőrzés
AD99 : 44441 : TYPE MISMATCH hibaüzenet
AD9E : 44446 : FRMEVL-tetszőleges kifejezés kiértékelése
AE83 : 44675 : A kifejezés következő elemének betöltése
AEA8 : 44712 : A Pi konstans (3.14159265)
AED4 : 44756 : A NOT utasítás
AEE3 : 44771 : Összehasonlítás az FN tokennel
AEEA : 44778 : Összehasonlítás az SGN tokennel
AEF1 : 44785 : Zárójelek közötti kifejezés beolvasása
AEF7 : 44791 : Karakterek vizsgálata BASIC szövegben
AEFA : 44794 : ")" vizsgálata
AEFD : 44797 : "," vizsgálata
AEFF : 44799 : Egy akkuban levő karakter ellenőrzése
AF08 : 44808 : SYNTAX ERROR hibaüzenet
AF14 : 44820 : BASIC-en belüli változó kiértékelése
AF28 : 44840 : A BASIC változó betöltése
AF61 : 44897 : Az egész típusú változó beolvasása
AF6E : 44910 : A valós típusú változó beolvasása
AF84 : 44932 : Az idő beolvasása

```

 Cím# : Cím# : Megjegyzés

AF92 : 44946 : BASIC ST beolvasása
 AFA0 : 44960 : A valós típusú változó beolvasása a FAC-ba
 AFA7 : 44967 : Függvényvizsgálat
 AFB1 : 44977 : Szövegfüggvény kezelése
 AFD1 : 45009 : Numerikus függvény kiértékelése és végrehajtása
 AFE6 : 45030 : Az OR utasítás
 AFE9 : 45033 : Az AND utasítás
 B016 : 45078 : Összehasonlítás
 B02E : 45102 : Szöveges változók összehasonlítása
 B081 : 45185 : A DIM utasítás
 B08B : 45195 : A változó beolvasása, dimenzionálás
 B113 : 45331 : Alfabetikus jel ellenőrzése
 B11D : 45341 : A változó első tárolása
 B128 : 45352 : Új változó előállítása
 B194 : 45460 : Az első tömbelem mutatójának kiszámítása
 B1A5 : 45477 : -32768 lebegőpontos állandó
 B1AA : 45482 : Lebegőpontos/egész átváltás
 B1B2 : 45490 : Egy kifejezés beolvasása az egész típusú változóba
 B1D1 : 45521 : Egy tömb megkeresése vagy létrehozása
 B245 : 45637 : BAD SUBSCRIPT hibüzenet
 B248 : 45640 : ILLEGAL QUANTITY hibüzenet
 B24D : 45645 : REDIM'D ARRAY hibüzenet
 B261 : 45665 : A tömbváltozó tárolása
 B2E9 : 45801 : A tömbelem keresése
 B30E : 45838 : Egy tömbelem címének kiszámítása
 B34C : 45900 : Rutin a tömb nagyságának kiszámításához
 B37D : 45949 : A FRE függvény
 B391 : 45969 : Egész-lebegőpontos átalakítás
 B39E : 45982 : A POS függvény
 B3A6 : 45990 : A parancsmód tesztelése
 B3AB : 45995 : ILLEGAL DIRECT hibüzenet
 B3AE : 45998 : UNDEF'D FUNCTION hibüzenet
 B3B3 : 46003 : A DEF FN utasítás
 B3E1 : 46049 : Az FN-szintaxis ellenőrzése
 B3F4 : 46068 : Az FN függvény
 B465 : 46181 : A STR\$ függvény
 B475 : 46197 : A szövegmutató kiszámítása
 B487 : 46215 : A szöveg beolvasása, mutató az A/Y-ban
 B4CA : 46282 : A szövegmutató bevezetése a szövegleíró verembe (\$19-21)
 B4D0 : 46288 : FORMULA TOO COMPLEX hibüzenet
 B4F4 : 46324 : Helyfoglalás a szöveg számára (hossza A-ban)
 B526 : 46374 : Garbage Collection (szemétgyűjtés)
 B5B0 : 46525 : A megszüntetési lehetőség ellenőrzése
 B606 : 46598 : Az érvénytelen szövegek törlése
 B63D : 46653 : A szövegek összeadása
 B67A : 46714 : A szöveg áthelyezése a lefoglalt területre
 B6A3 : 46755 : FRESTR, az érvénytelen szövegek törlése a veremből
 B6DB : 46811 : A szövegmutató eltávolítása a szövegleíró veremből
 B6EC : 46828 : A CHR\$ függvény
 B700 : 46848 : A LEFT\$ függvény
 B72C : 46892 : A RIGHT\$ függvény
 B737 : 46903 : A MID\$ függvény

Cím\$	Cím#	Megjegyzés
B761	46945	A szövegpáráméterek beolvasása a veremből
B77C	46972	A LEN függvény
B782	46978	A szöveg paramétereinek beolvasása
B78B	46987	Az ASC függvény
B79B	47003	GETBYT, egybyte-os paraméter beolvasása az X-be
B7AD	47021	A VAL függvény
B7EB	47035	GETADR és GETBYT, 16 és 8 bites értékek betöltése
B7F7	47095	GETADR, 16 bites érték átváltása címformátumra
B80D	47117	A PEEK függvény
B824	47140	A POKE utasítás
B82D	47149	A WAIT utasítás
B849	47177	FAC=FAC+.5
B850	47184	Kivonás, FAC=Konstans(A/Y)-FAC
B853	47187	Kivonás, FAC=ARG-FAC
B862	47202	Az összeadás/Kivonás rutin Kiegészítése
B867	47207	Összeadás, FAC=Konstans(A/Y)+FAC
B86A	47210	Összeadás, FAC=ARG+FAC
B847	47431	A FAC invertálása
B97E	47486	OVERFLOW ERROR hibaüzenet
B983	47491	A megadott lebegőpontos regiszter jobbralejtetése
B9BC	47548	LOG állandók
B9EA	47594	A LOG függvény
BA28	47656	Szorzás, FAC=Konstans(A/Y)*FAC
BA2B	47659	Szorzás, FAC=ARG*FAC
BA59	47705	Szorzás bitenként
BA8C	47756	ARG=Konstans(A/Y)
BAB7	47799	A hatványkitevő kiszámítása
BAD4	47828	A FAC előjelének vizsgálata
BAE2	47842	FAC=FAC*10
BAF9	47865	Lebegőpontos Konstans:10
BAFE	47870	FAC=FAC/10
BB0F	47887	Osztás, FAC=Konstans(A/Y)/FAC
BB12	47890	Osztás, FAC=ARG/FAC
BB8A	48010	DIVISION BY ZERO hibaüzenet
BB8F	48015	Az aritmetikai regiszter (\$26-29) átvitele a FAC-ba
BBA2	48034	Az A/Y állandó beolvasása a FAC-ba
BBC7	48071	A FAC átvitele az akku#4-be
BBCA	48074	A FAC átvitele az akku#3-ba
BBD0	48080	A FAC átvitele a változóba
BBFC	48124	Az ARG átvitele a FAC-ba
BC0C	48140	A FAC átvitele az ARG-ba
BC1B	48155	A FAC kerekítése
BC2B	48171	A FAC előjelének beolvasása
BC39	48185	Az SGN függvény
BC58	48216	Az ABS függvény
BC5B	48219	A FAC és az (A/Y) Konstans összehasonlítása
BC9B	48283	Lebegőpontos szám átalakítása egész típusúra
BCCC	48332	Az INT függvény
BCE9	48361	A mantissa kitöltése nullákkal
BCF3	48371	ASCII szám átalakítása lebegőpontoszá
BDB3	48563	A lebegőpontos/ASCII átalakítás állandói
BDC2	48578	A sorszám kiírása hibaüzenetnél

 Cím# : Cím# : Megjegyzés

BDCD : 48589 : Az A/X-ben tárolt pozitív egész szám kiírása
 BDDD : 48605 : A FAC átváltása ASCII formátumra
 BF11 : 48913 : 0.5 állandó az SQR függvényhez
 BF16 : 48918 : 32 bites állandók ASCII-ra alakításhoz
 BF3A : 48954 : A TI-TI\$ átalakítás állandói
 BF71 : 49009 : Az SQR függvény
 BF78 : 49016 : Hatványozás, FAC=ARG felemelve az (A/Y) kitevőre
 BF7B : 49019 : Hatványozás, FAC=ARG↑FAC
 BFB4 : 49076 : Előjelváltás
 BF8F : 49087 : Az EXP függvény állandói
 BFED : 49133 : Az EXP függvény
 E043 : 57411 : Polinomszámítás, $Y=(A1)*X+(A2)*X^3+(A3)*X^5+ \dots$
 E059 : 57433 : Polinomszámítás, $Y=(A0)+(A1)*X+(A2)*X^2+(A3)*X^3+ \dots$
 E08D : 57485 : Állandók az RND-hez
 E097 : 57495 : Az RND függvény
 E0F9 : 57593 : Hibakiértékelés I/O rutinok után
 E10C : 57612 : BSOUT, egy karakter kiírása, kód az akkuban
 E112 : 57618 : BASIN, egy karakter beolvasása, kód az akkuban
 E118 : 57624 : CHKOUT, az output egység kijelölése
 E11E : 57630 : CHKIN, az input egység kijelölése
 E124 : 57636 : GETIN, egy karakter az akkuba
 E12A : 57642 : A SYS utasítás
 E156 : 57686 : A SAVE utasítás
 E165 : 57701 : A VERIFY utasítás
 E168 : 57704 : A LOAD utasítás
 E1BE : 57790 : Az OPEN utasítás
 E1C7 : 57799 : A CLOSE utasítás
 E1D4 : 57812 : A LOAD és a SAVE paramétereinek beolvasása
 E200 : 57856 : Egybyte-os paraméter beolvasása
 E206 : 57862 : További karakterek ellenőrzése
 E20E : 57870 : Vessző és a folytatás ellenőrzése
 E219 : 57881 : Az OPEN paramétereinek beolvasása
 E264 : 57956 : A COS függvény
 E26B : 57963 : A SIN függvény
 E2B4 : 58036 : A TAN függvény
 E2E0 : 58080 : SIN és COS állandók
 E30E : 58126 : Az ATN függvény
 E33E : 58174 : Az ATN iterációs állandói
 E37B : 58235 : BASIC NMI-belépési pont
 E394 : 58260 : BASIC hidegindítás
 E3A2 : 58274 : A CHRGET rutin másolata
 E3BA : 58298 : Az RND függvény kezdőértéke (.811635157)
 E3BF : 58303 : A BASIC RAM inicializálása
 E422 : 58402 : A bekapcsolási üzenet kiírása
 E447 : 58439 : A BASIC-vektorok táblázata
 E453 : 58451 : A BASIC-vektorok betöltése
 E45F : 58463 : Rendszerüzenetek
 E4AD : 58541 : A BASIC CHKOUT
 E4D3 : 58579 : A startbitkapcsoló visszaállítása
 E4DA : 58586 : A háttérszín beállítása (karakter)
 E4E0 : 58592 : Vár a COMMODORE billentyűre (8.5 sec)
 E4EC : 58604 : Az RS232 átviteli állandói, PAL változat

Az operációs rendszer rutinjai

Cím\$	CímH	Megjegyzés	
E580	58624	A CIA báziscímének beolvasása	: \$FFF3
E585	58629	A sorok és oszlopok számának beolvasása	: \$FFED
E58A	58634	A kurzor beállítása, C=0:beállítás, C=1:lekérdezés	: \$FFF0
E518	58648	Képernyő reset	
E544	58692	A Képernyő törlése	
E566	58726	Kurzor home	
E56C	58732	A kurzorpozíció kiszámítása, a mutatók beállítása	
E591	58769	Rutin a billentyűzési várakozóciklushoz	
E59A	58778	Video-reset, nem használt belépési pont	
E5A0	58784	A VIC inicializálása	
E5B4	58804	Egy karakter beolvasása a billentyűzet-pufferből	
E5CA	58826	Billentyűzési várakozóciklus	
E632	58930	Egy karakter beolvasása a Képernyőről	
E684	59012	Az idézőjel vizsgálata	
E691	59025	A beolvasott karakter kiírása a Képernyőre	
E686	59062	A sorkezdet újraszámítása	
E701	59137	Visszalépés az előző sorra	
E716	59158	Kiírás a Képernyőre	
E7D4	59348	127-nél nagyobb kódú karakter kezelése	
E832	59442	Kurzor fel	
E87C	59516	A következő sorra ugrás	
E891	59537	Return végrehajtása	
E8A1	59553	Ellenőrzés: a sorszám csökkentése	
E883	59571	Ellenőrzés: a sorszám növelése	
E8CB	59595	A színkód vizsgálata	
E8DA	59610	A színkódok táblázata	
E8EA	59626	A Képernyő görgetése	
E965	59749	Egy sor beszúrása	
E9C8	59848	A sor görgetése felfelé	
E9E0	59872	A karakter színének beállítása	
E9F0	59888	Az X. sor videoram mutatója	
E9FF	59903	Az X. sor törlése	
EA13	59923	A kurzorvillogás-számláló beállítása	
EA1C	59932	A karakter és a szín beállítása a Képernyőn	
EA24	59940	A színtár mutatójának kiszámítása	
EA31	59953	Maszkolható megszakítás rutin (IRQ)	
EA87	60039	A billentyűzet lekérdezése	
EB48	60232	A billentyűzet-dekódoló mátrix kiválasztása	
EB79	60281	A dekódoló táblázatok címei	
EB81	60289	1. dekódoló táblázat - SHIFT nélkül	
EBC2	60354	2. dekódoló táblázat - SHIFT-tel	
EC03	60419	3. dekódoló táblázat - COMMODORE-ra	
EC44	60484	A vezérlőkarakterek vizsgálata	
EC78	60536	4. dekódoló táblázat - CTRL billentyűvel	
ECB9	60601	A videovezérlő állandói	
ECE7	60647	"LOAD (CR) RUN (CR)" szöveg	
ECF0	60656	A Képernyősor-kezdet LSB táblázata	
ED09	60681	TALK küldése	: \$FFB4
ED0C	60684	LISTEN küldése	: \$FFB1
ED40	60736	Egy byte továbbítása az IEC buszon	
EDAD	60845	DEVICE NOT PRESENT	
EDB0	60848	TIME OUT	

 Cím# : Cím# : Megjegyzés

EDB9	: 60857	: SECOND, másodlagos cím LISTEN után	: \$FF93
EDC7	: 60871	: TKSA, masodlagos cím TALK után	: \$FF96
EDDD	: 60893	: IECOUT, egy byte továbbítása az IEC buszra	: \$FFA8
EDEF	: 60911	: UNTALK küldése	: \$FFAB
EDFE	: 60926	: UNLISTEN küldése	: \$FFAE
EE13	: 60947	: IECIN, egy byte beolvasása az IEC-buszról	: \$FFA5
EE85	: 61061	: CLOCK HI rutin	
EE8E	: 61070	: CLOCK LO rutin	
EE97	: 61079	: DATA HI rutin	
EEA0	: 61088	: DATA LO rutin	
EEA9	: 61097	: Egy bit beolvasása a Carry-be	
EEB3	: 61107	: Késleltető rutin, Kb. 750 mikrosec.	
EEBB	: 61115	: RS232 output rutin	
EF06	: 61190	: Új RS232 byte küldése	
EF2E	: 61230	: Hiba: nincs DSR	
EF3B	: 61243	: Hiba: nincs CTS	
EF4A	: 61258	: RS232 - az adatbitek számának meghatározása	
EF59	: 61273	: RS232 - az adatbit beolvasása	
EF7E	: 61310	: Az adatfogadás előkészítése	
EFC5	: 61381	: Paritáshiba érzékelése	
EFCA	: 61386	: Túlcordulás érzékelése	
EFC0	: 61389	: Break érzékelése	
EF00	: 61392	: Kerethiba	
EFE1	: 61409	: RS232 CHKOUT, kiírás az RS232-esre	
F00D	: 61453	: Hiba: nincs DSR	
F014	: 61549	: Kiírás az RS232 pufferbe	
F04D	: 61517	: RS232 CHKIN, az RS232 input beállítása	
F086	: 61574	: GET az RS232-ről	
F0A4	: 61604	: RS232 - várakozás az átvitel végére	
F0BD	: 61629	: Rendszerüzenetek (SEARCHING, FOUND, stb.)	
F12B	: 61739	: A rendszerüzenetek kiírása	
F13E	: 61758	: GETIN, egy karakter beolvasása	: \$FFE4
F14E	: 61774	: GETIN az RS232-ről	
F157	: 61783	: BASIN - egy karakter beolvasása	: \$FFCF
F16A	: 61802	: BASIN a képernyőről	
F179	: 61817	: BASIN szalagról	
F199	: 61849	: Egy karakter beolvasása szalagról	
F1AD	: 61877	: IEC input	
F1B8	: 61880	: RS232 input	
F1CA	: 61898	: BSOUT, egy karakter kiírása	
F1DD	: 61917	: Output szalagra	
F208	: 61960	: RS232 output	
F20E	: 61966	: CHKIN - az input egység beállítása	: \$FFC6
F22A	: 61994	: CHKIN a szalagos egységre	
F237	: 62007	: IEC CHKIN	
F250	: 62032	: CHKOUT - a kimeneti egység beállítása	: \$FFC9
F279	: 62073	: Output IEC-buszra	
F291	: 62097	: CLOSE - logikai file szám az akkuban	: \$FFC3
F2AB	: 62123	: RS232 file lezárása	
F2C8	: 62152	: A szalag-file lezárása	
F2EE	: 62190	: Az IEC-file lezárása	
F30F	: 62223	: A logikai file-szám ellenőrzése	

 Cím\$: Cím# : Megjegyzés

F31F : 62239 : A fileparaméterek beállítása
 F32F : 62255 : CLALL - minden megnyitott file lezárása : \$FFC7
 F333 : 62259 : CLRCHN - lezárja az aktív I/O csatornát : \$FFCC
 F34A : 62282 : OPEN : \$FFC0
 F305 : 62421 : Egy file megnyitása az IEC buszon
 F409 : 62473 : RS232 OPEN
 F483 : 62595 : A CIA-K beállítása RS232 üzemre
 F49E : 62622 : A LOAD rutin : \$FFD5
 F4B8 : 62648 : LOAD az IEC-buszról
 F533 : 62771 : LOAD szalagról
 F5AF : 62885 : SEARCHING FOR (filenév) Kiírása
 F5C1 : 62913 : A filenév Kiírása
 F5D2 : 62930 : LOADING/VERIFYING Kiírása
 F5DD : 62941 : SAVE rutin : \$FFD8
 F5FA : 62970 : Tárolás az IEC buszon
 F659 : 63065 : SAVE a szalagos egységre
 F68F : 63119 : SAVING Kiírása
 F69B : 63131 : UDTIM - az idő növelése : \$FFE4
 F6DD : 63197 : Az idő beolvasása : \$FFDE
 F6E4 : 63204 : Az idő beállítása : \$FFDB
 F6ED : 63213 : A STOP billentyű lekérdezése : \$FFE1
 F6FB : 63227 : Az operációs rendszer üzeneteinek Kiírása
 F6FB : 63227 : TOO MANY FILES hibüzenet
 F6FE : 63230 : FILE OPEN hibüzenet
 F701 : 63233 : FILE NOT OPEN hibüzenet
 F704 : 63236 : FILE NOT FOUND hibüzenet
 F707 : 63239 : DEVICE NOT PRESENT hibüzenet
 F70A : 63242 : NOT INPUT FILE hibüzenet
 F70D : 63245 : NOT OUTPUT FILE hibüzenet
 F710 : 63248 : MISSING FILENAME hibüzenet
 F713 : 63251 : ILLEGAL DEVICE NUMBER hibüzenet
 F72C : 63276 : A programfejléc beolvasása a szalagról
 F76A : 63354 : A fejléc generálása és felírása a szalagra
 F7D0 : 63440 : A szalagpuffer Kezdőcímének beolvasása
 F7D7 : 63447 : A szalagpuffer Kezdő- és végcímének Kiszámítása
 F7EA : 63466 : A szalag fejlécének Keresése
 F80D : 63501 : A szalagpuffer mutatójának növelése
 F817 : 63511 : Várakozás a kazettás egység billentyűjére
 F82E : 63534 : Billentyű lekérdezés
 F838 : 63544 : Várakozás írásra
 F841 : 63553 : A blokk olvasása szalagról
 F84A : 63562 : A program betöltése szalagról
 F864 : 63588 : A puffer felírása a szalagra
 F86B : 63595 : A blokk vagy a program felírása a szalagra
 F8BE : 63678 : Kivárni az I/O lezárást
 F8D0 : 63696 : A STOP billentyű vizsgálata
 F8E2 : 63714 : Előkészítés olvasásra
 F92C : 63788 : Megszakításrutin szalagolvasáshoz
 FB8E : 64398 : A kezdőcím áthelyezése
 FB97 : 64407 : A bitszámláló beállítása
 FBA6 : 64422 : Egy bit felírása a szalagra
 FBCE : 64461 : Megszakításrutin szalagra íráshoz

 Cím\$: Cím# : Megjegyzés

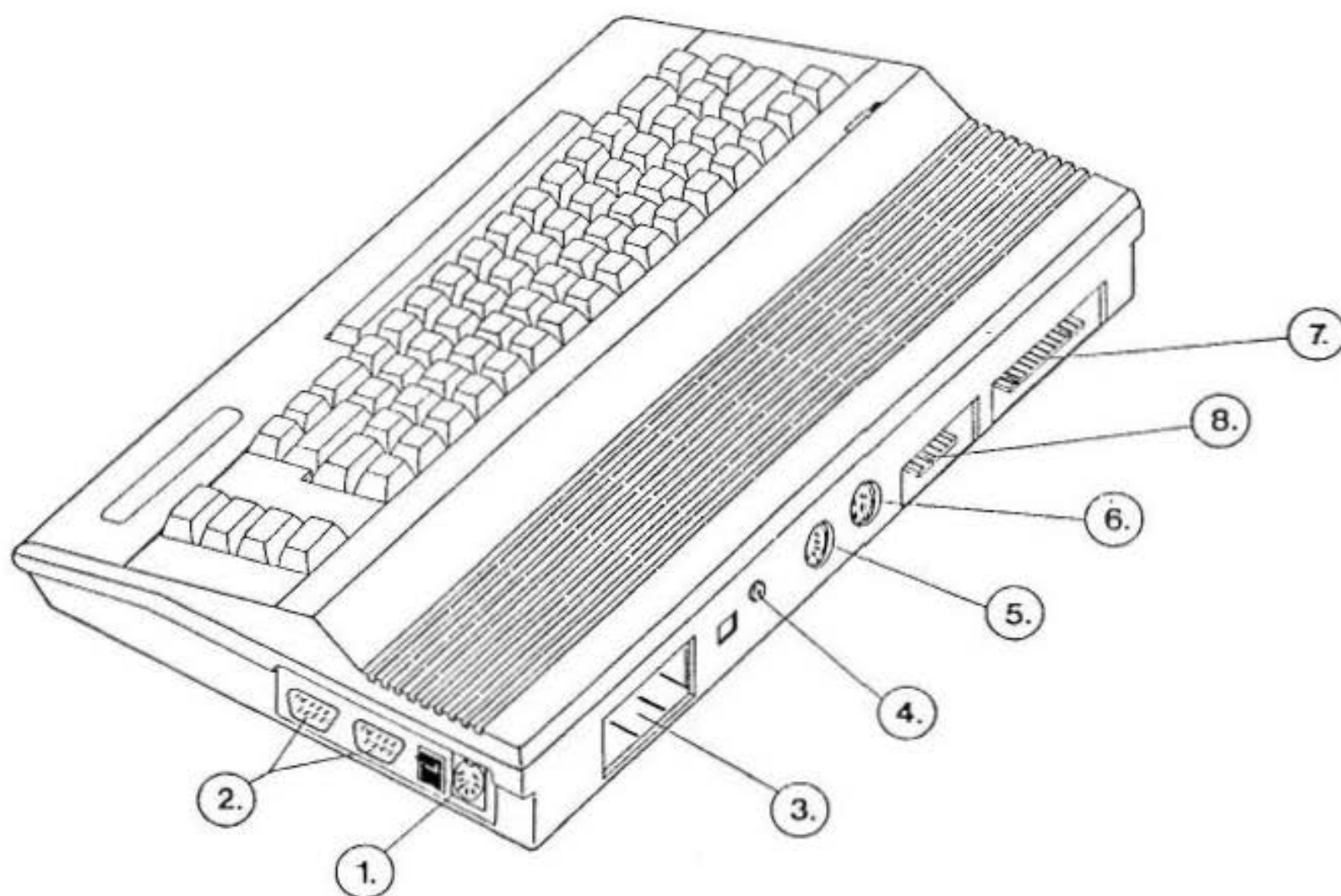
FC6A : 64618 : Megszakításrutin íráshoz
 FC93 : 64659 : A normál IRQ visszaállítása
 FCB8 : 64696 : Az IRQ-vektor beállítása
 FCCA : 64714 : A motor Kikapcsolása
 FCD1 : 64721 : A végcím elérésének ellenőrzése
 FCDB : 64731 : A címmutató növelése
 FCE2 : 64738 : Hardver RESET
 FD02 : 64770 : A ROM-EEPROM modul ellenőrzése a \$8000-tól
 FD10 : 64784 : A ROM-modul azonosítója
 FD15 : 64789 : A hardver és az I/O vektorok beállítása : \$FF8A
 FD30 : 64816 : A hardver táblázat és az I/O vektorok
 FD50 : 64848 : A BASIC-KERNAL munkaterület inicializálása
 FD9B : 64923 : IRQ vektorok
 FDA3 : 64936 : A megszakítás inicializálása : \$FF84
 FDF9 : 65017 : SETNAM, a filenév paramétereinek beállítása : \$FFBD
 FE00 : 65024 : SETFLS, az aktív file paramétereinek beállítása : \$FFBA
 FE07 : 65031 : A státusz beolvasása : \$FFB7
 FE18 : 65048 : A rendszerüzenetek jelzőjének beállítása
 FE1C : 65052 : A státusz beállítása
 FE21 : 65057 : Az IEC busz timeout jelzőjének beállítása : \$FFA2
 FE25 : 65061 : MEMTOP, a BASIC RAM felső határa : \$FF99
 FE34 : 65076 : MEMBOT, a BASIC RAM alsó határa : \$FF9C
 FE43 : 65091 : NMI rutin
 FE66 : 65126 : NMI melegstart
 FE72 : 65138 : Az RS232 NMI rutinja
 FEBC : 65212 : Kilépés a megszakításból
 FEC2 : 65218 : Az RS232 NTSC baud-Rate állandói
 FED6 : 65238 : NMI-rutin RS232 inputhoz
 FF07 : 65287 : NMI-rutin RS232 outputhoz
 FF2E : 65326 : Az időzítő-érték továbbítása
 FF43 : 65347 : IRQ-hívás szalagos üzemmódban
 FF48 : 65352 : IRQ rutin, belépési pont
 FF5B : 65371 : Video reset
 FF6E : 65390 : Az interrupt időzítő beállítása
 FF81 : 65409 : Ugrási táblázat az operációs rendszer rutinjaihoz
 FFFA : 65530 : NMI vektor \$FE43
 FFFC : 65532 : RESET vektor \$FCE2
 FFFE : 65534 : IRQ vektor \$FF48

13. táblázat. Az operációs rendszer ugrótáblája

Cím#	Cím#	Név	Tartalom	Magyarázat
FF81	65409	CINT	JMP \$FF5B	: Video-controller reset
FF84	65412	IOINIT	JMP \$FDA3	: I/O reset (CIA 1 & CIA 2)
FF87	65415	RAMTES	JMP \$FD50	: A munkaterület inicializálása
FF8A	65418	RESTOR	JMP \$FD15	: I/O vektorok beállítása
FF8D	65421	VECTOR	JMP \$FD1A	: - " -
FF90	65424	SETMSG	JMP \$FE18	: A státus beállítása
FF93	65427	SECOND	JMP \$EDB9	: LISTEN, másodlagos cím
FF96	65430	TKSA	JMP \$EDC7	: TALK, másodlagos cím
FF99	65433	MEMTOP	JMP \$FE25	: A RAM tető beállítása/beolvasása
FF9C	65436	MEMBOT	JMP \$FE34	: A RAM kezdet beállítása/beolv.
FF9F	65439	SNCKEY	JMP \$EA87	: A billentyűzet lekérdezése
FFA2	65442	SETTMD	JMP \$FE21	: IEC TIME OUT beállítása
FFA5	65445	ACPTR	JMP \$EE13	: Input az IEC-buszról
FFA8	65448	CIOUT	JMP \$EDDD	: Output az IEC-buszra
FFAB	65451	UNTLK	JMP \$EDEF	: UNTALK
FFAE	65454	UNLSN	JMP \$EDFE	: UNLISTEN
FFB1	65457	LISTEN	JMP \$ED0C	: LISTEN
FFB4	65460	TALK	JMP \$ED09	: TALK
FFB7	65463	READST	JMP \$FE07	: A státus beolvasása
FFBA	65466	SETFLS	JMP \$FE00	: A file paramétereinek beállítása
FFBD	65469	SETNAM	JMP \$FDF9	: A filenév paramétereinek beáll.
FFC0	65472	OPEN	JMP(\$031A) \$F34A	: OPEN
FFC3	65475	CLOSE	JMP(\$031C) \$F291	: CLOSE
FFC6	65478	CHKIN	JMP(\$031E) \$F20E	: CHKIN - az input egység beáll.
FFC9	65481	CHKOUT	JMP(\$0320) \$F250	: CHKOUT - az output egység beáll.
FFCC	65484	CLRCHN	JMP(\$0322) \$F333	: CLRCHN - I/O visszaállítás
FFCF	65487	CHRIN	JMP(\$0324) \$F157	: BASIN - input
FFD2	65490	CHROUT	JMP(\$0326) \$F1CA	: BSOUT - output
FFD5	65493	LOAD	JMP \$F49E	: LOAD
FFD8	65496	SAVE	JMP \$F5DD	: SAVE
FFDB	65499	SETTIM	JMP \$F6E4	: Az idő beállítása
FFDE	65502	RDTIM	JMP \$F6DD	: Az idő beolvasása
FFE1	65505	STOP	JMP(\$0328) \$F6ED	: A STOP bill. lekérdezése
FFE4	65508	GETIN	JMP(\$032A) \$F13E	: GETIN
FFE7	65511	CLALL	JMP(\$032C) \$F32F	: CLALL
FFEA	65514	UDTIM	JMP \$F69B	: Az idő növelése
FFED	65517	SCREEN	JMP \$E505	: A képernyőméret beolvasása
FFF0	65520	PLOT	JMP \$E50A	: A kurzor olvasása/beállítása
FFF3	65523	IOBASE	JMP \$E500	: Az I/O modul kezdőcímének beolv.

1.7. A csatlakozók

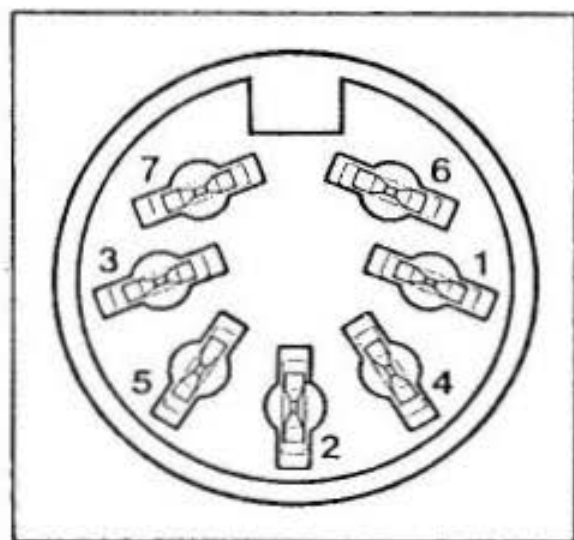
A C64-es meglehetősen sok csatlakozóval rendelkezik. A csatlakozók többségének valamilyen adatforgalom lebonyolítása a funkciója. Az ábrákon a C64-es modernebb változata látható, de mind a csatlakozók száma, mind elhelyezkedése azonos a régi változattal.



24. ábra. A C64-es csatlakozói

- | | |
|-----------------------------|---------------------------------|
| 1. Tápfeszültség-csatlakozó | 5. Audio/video vonalkimenet |
| 2. Control Port 1 & 2 | 6. Serial port, soros busz |
| 3. Expansion (bővítő-) port | 7. User port, felhasználói port |
| 4. RF kimenet televízióhoz | 8. A kazettás egység portja |

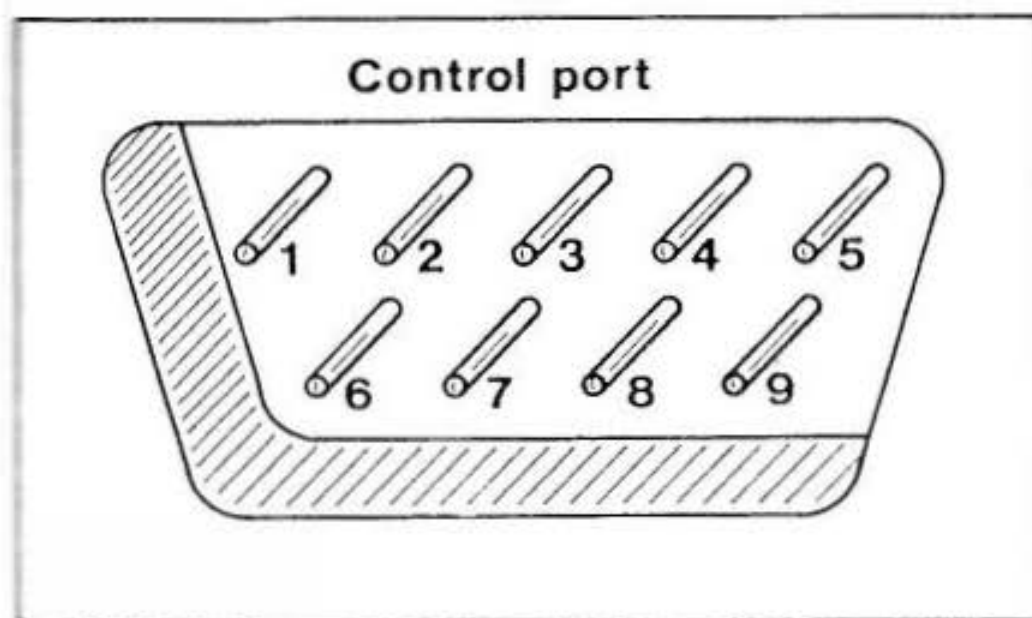
A tápfeszültség csatlakozója DIN szabványú 7 tűs tuchel csatlakozó aljzat. A 7 tűből csak 4 foglalt.



A csatlakozó lábkiosztása:

- | | |
|---|------------------------------------|
| 1 | GND: rendszerföld |
| 2 | NC: nem használt |
| 3 | NC: nem használt |
| 4 | NC: nem használt |
| 5 | +5: V tápfeszültség |
| 6 | 9 V AC ~: 9 V váltakozó feszültség |
| 7 | 9 V AC ~: 9 V váltakozó feszültség |

A Control Port 1 & 2 a joystick, a paddle, illetve a fényceruza csatlakoztatására szolgál. Ez a csatlakozó egy 9 tűs D Submin aljzat Lábkiosztása:



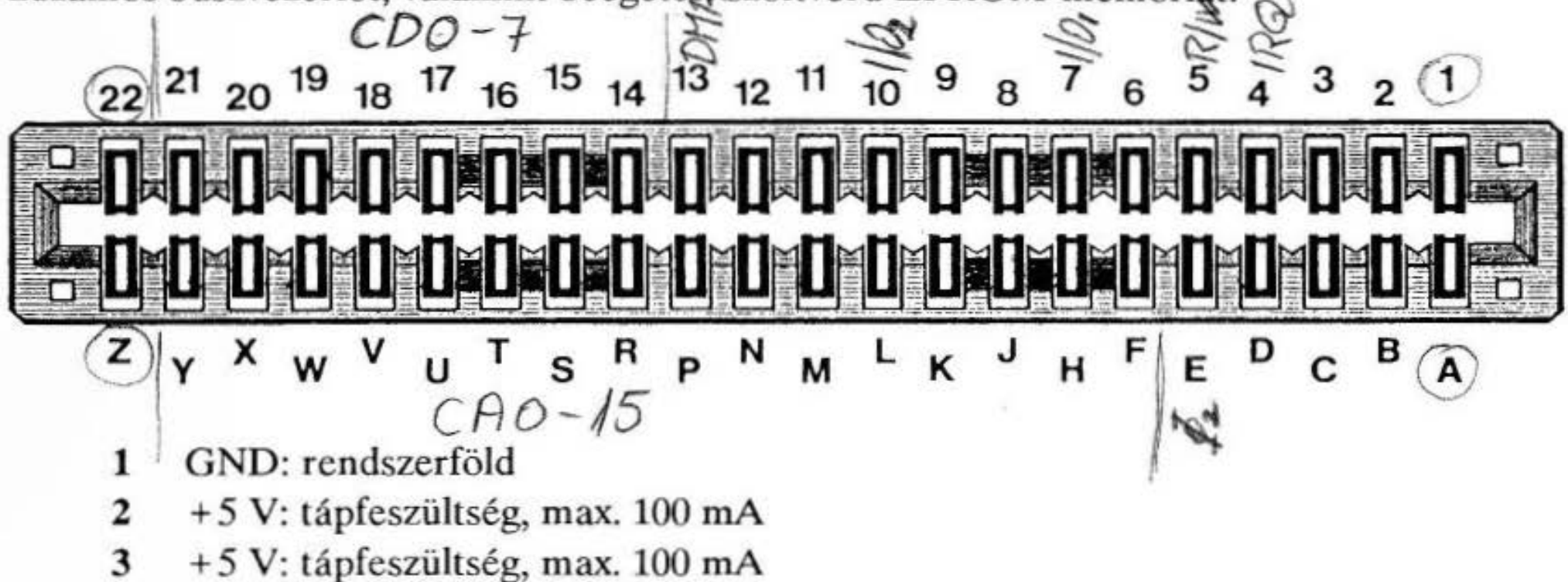
Control Port 1

- 1 JOYA0: joystick bitek
- 2 JOYA1: joystick bitek
- 3 JOYA2: joystick bitek
- 4 JOYA3: joystick bitek
- 5 POT AY: paddle A, Y potenciométer
- 6 BUTTON A/LP: joystick tűzgomb, fényceruza ütembemenet
- 7 +5 V: tápfeszültség, max. 100 mA
- 8 GND: rendszerföld
- 9 POT AX: paddle A, X potenciométer

Control Port 2

- 1 JOYB0: joystick bitek
- 2 JOYB1: joystick bitek
- 3 JOYB2: joystick bitek
- 4 JOYB3: joystick bitek
- 5 POT BY: paddle B, Y potenciométer
- 6 BUTTON B: joystick tűzgomb
- 7 +5 V: tápfeszültség, max. 100 mA
- 8 GND: rendszerföld
- 9 POT BX: paddle B, X potenciométer

A processzorkártya az expansion (bővítő-) porton keresztül bővíthető. Funkciója és alkalmazása szerteágazó, mivel a processzor jeleinek többsége megtalálható itt. Csatlakoztathatunk erre a portra RAM- vagy ROM-bővítőt, társprocesszort, további CIA chipe(ke)t, párhuzamos buszvezérlőt, valamint beégetett szoftverű EPROM-memóriát.



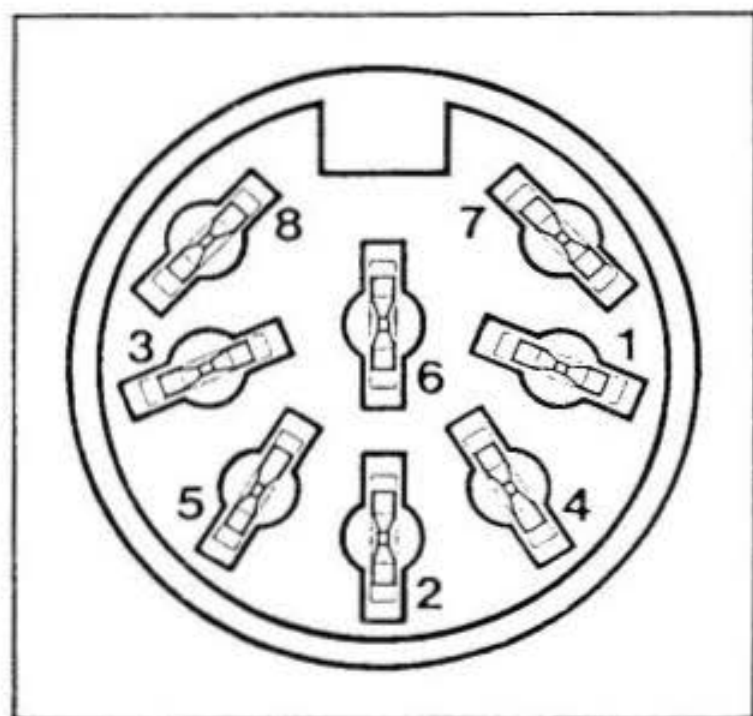
- 1 GND: rendszerföld
- 2 +5 V: tápfeszültség, max. 100 mA
- 3 +5 V: tápfeszültség, max. 100 mA

- 4 $\overline{\text{IRQ}}$: a processzor IRQ vonala
- 5 $\text{CR}/\overline{\text{W}}$: a processzor R/W vonala
- 6 Dot Clock: a VIC raszterfrekvenciája, 7,88198 MHz (NTSC=8,18181 MHz)
- 7 $\overline{\text{I/O}}_1$: ha alacsony, a külső I/O egység a \$DE00 — DEFF területen érhető el
- 8 $\overline{\text{GAME}}$: bemenet, az AM vezérlővonala a \$A000 — BFFF vagy \$E000 — FFFF terület kiválasztására
- 9 $\overline{\text{EXROM}}$: bemenet, az AM vezérlővonala a \$8000 — 9FFF terület kiválasztására
- 10 $\overline{\text{I/O}}_2$: ha alacsony, a külső I/O egység a \$DF00 — DFFF területen érhető el
- 11 $\overline{\text{ROML}}$: kimenet, az AM által vezérelt vonal külső ROM elérésére
- 12 BA: Bus Available, a VIC jele; ha magas, a busz használható
- 13 $\overline{\text{DMA}}$: bemenet, Direct Memory Access, olvasáskérelem jelzése
- 14 — 21 CD7 — CD0: adatbusz
- 22 GND: rendszerföld
- A GND: rendszerföld
- B $\overline{\text{ROMH}}$: kimenet, az AM által vezérelt vonal külső ROM elérésére
- C $\overline{\text{RESET}}$: a processzor $\overline{\text{RESET}}$ vonala
- D $\overline{\text{NMI}}$: a processzor $\overline{\text{NMI}}$ vonala
- E Φ_2 : rendszerütem, 985,24 kHz (NTSC=1,0227 MHz)
- F — Y CA15 — CA0: címbusz
- Z GND: rendszerföld

A televízióhoz csatlakoztatható RF kimenethez a számítógép belsejében tartozik egy kis fémdoboz. A dobozban található elektronika feladata, hogy a VIC szín- és világosságjelét, valamint a SID hangjelét egyesítse, és a televíziók számára fogható nagyfrekvenciás jellé alakítsa át. Az így létrehozott jel a televíziók antennabemenetére csatlakoztatható. A C64-es modulátora a IV. televíziósávban a 36. csatornára állítja be a VIC jelét (590 — 598 MHz). A csatlakozó szabványos RCA aljzat.

Az audio-video vonalkimenet DIN szabványú, 8 tűs tuchel csatlakozó aljzat; a 8 tűből csak 6 foglalt.

A csatlakozó lábkiosztása:



- 1 Luminance: világosságjel
- 2 GND: rendszerföld
- 3 Audio Out: hangjelkimenet
- 4 Composite Video Out: összetett videojel
- 5 Audio In: hangjelbemenet
- 6 Chroma: színjel
- 7 NC: nem használt
- 8 NC: nem használt

Az 1-4-6 tűk mindegyike videojel-kimenet; a rajtuk keresztül továbbított jelek azonban némileg különböznek egymástól. A Composite Video Out jele olyan összetett jel, amely egyaránt tartalmazza a televízió (monitor) számára szükséges világosságjelet, valamint a színek elkülönítéséhez szükséges színjelet. A színjel olyan felharmonikusa az alapjelnek (világosságjel), amelynek „ráültetése” a képfrekvenciára nem okoz interferenciát — így a színjel és a világosságjel egyszerűen elkülöníthető.

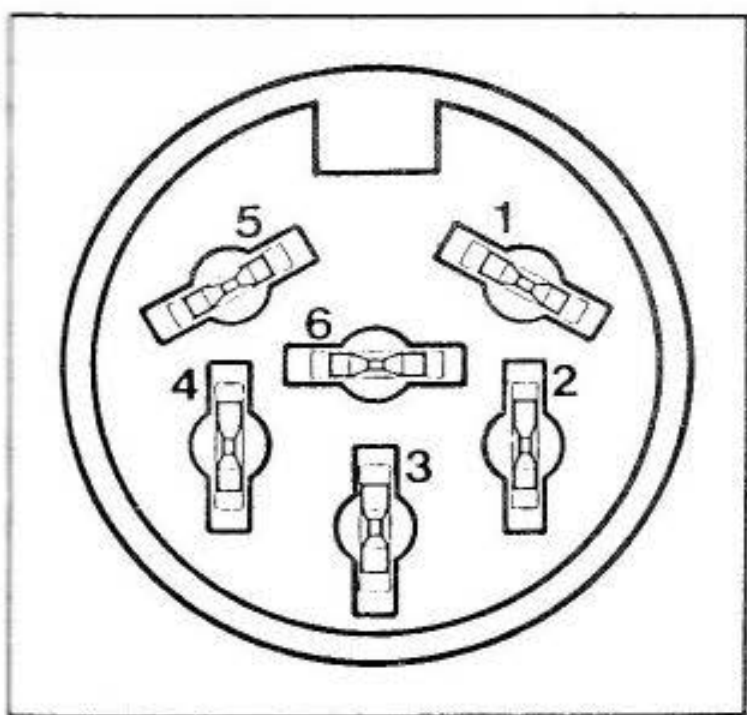
A kimenetekre színes vagy monokróm monitorok, illetve videojel-bemenettel rendelkező televíziók csatlakoztathatók. A színes monitorok általában jobb képet adnak, ha a világosságjel és színjel külön-külön vezetéken érkezik hozzájuk. A monokróm monitorokban nincs olyan szűrőfokozat, mint amilyen a fekete-fehér televíziókban. Azaz az összetett videojel többnyire rosszabb képet eredményez, mint az egyszerű világosságjel. (Egyébként a fekete-fehér televíziók is csak akkor képesek kiszűrni a színjelet, ha a számítógép RF kimenetére vannak csatlakoztatva.) Tehát a színes monitorokat mindegy hogyan, a monokróm monitorokat pedig lehetőleg a világosságjelre kössük, mert így biztosan éles képet kapunk.

A SID által létrehozott hang a 3. tűn jelenik meg. A jel erősítés nélkül hangszórón keresztül nem hallható, csak fejhallgatón.

Az 5. tű a SID EXT IN bemenetére van kötve. A chip ezen a vonalon keresztül kapja meg a külső hangforrás jelét. A jelfeszültség nem haladhatja meg a 3 V-ot, ellenkező esetben a SID belső áramköre tönkremegy.

A soros busz (serial port) DIN szabványú, 6 tűs tuchel csatlakozó aljzat; mind a 6 tűje foglalt.

A csatlakozó lábkiosztása:

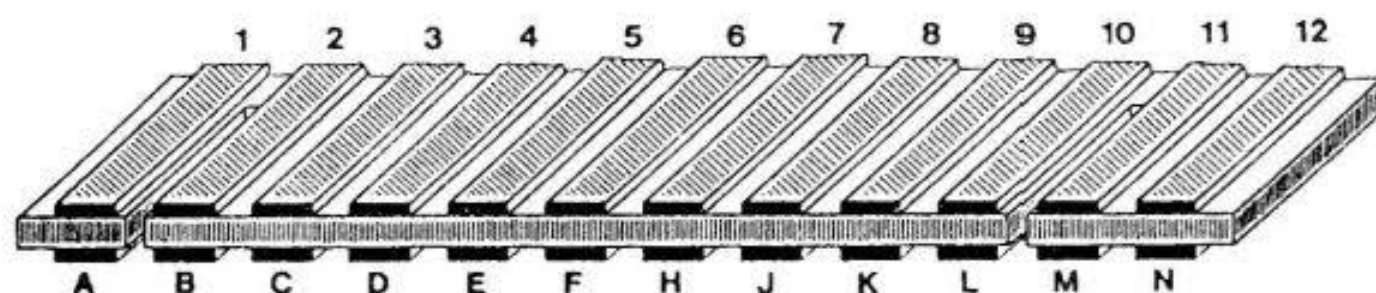


- 1 SRQ IN: Service Request In
- 2 GND: rendszerföld
- 3 ATN IN/OUT: Attention
- 4 CLK IN/OUT: szinkron órajel
- 5 DATA IN/OUT: adatvonal
- 6 RESET: a processzor $\overline{\text{RESET}}$ vonala

A vonalak részletes leírása a CIA2 chip ismertetésénél található.

A felhasználói csatlakozó (user port) lapos, 24 tűs csatlakozó aljzat; mind a 24 tű foglalt.

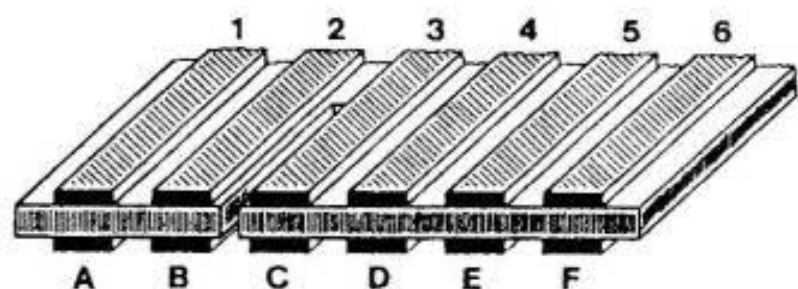
A csatlakozó lábkiosztása:



1	GND: rendszerföld	9	ATN IN: Attention
2	+5 V: max. 100 mA	10	9 V AC: max. 100 mA
3	$\overline{\text{RESET}}$: processzor $\overline{\text{RESET}}$ vonala	11	9 V AC: max. 100 mA
4	CNT1: CIA1 CNT	12	GND: rendszerföld
5	SP1: CIA1 SP	A	GND: rendszerföld
6	CNT2: CIA2 CNT	B	$\overline{\text{FLAG2}}$: CIA2 $\overline{\text{FLAG}}$
7	SP2: CIA2 SP	C—M	PB ₀ —PB ₇ : CIA2 B portjának bitjei
8	$\overline{\text{PC2}}$: CIA2 $\overline{\text{PC}}$	N	GND: rendszerföld

A csatlakozó funkcióit a CIA2 chip leírása tartalmazza.

A kazettás egység portja



A — 1	GND: rendszerföld
B — 2	+5 V: tápfeszültség, max. 100 mA
C — 3	MOTOR: a motor tápfeszültsége, stabilizált +6 V
D — 4	READ: olvasóvezeték, a CIA1 $\overline{\text{FLAG}}$ vonalára kötve
E — 5	WRITE: írás, a processzor P ₃ portja
F — 6	SENSE: állapotszenzor, a processzor P ₄ portja

A MOTOR vonalat a processzor P₅ portbitje vezérli. A vezérlővonal egészen pontosan azt határozza meg, hogy a kimeneten megjelenjen-e a +6 V-os egyenfeszültség, vagy sem. A READ vonal közvetlen kapcsolatban van a CIA1 $\overline{\text{FLAG}}$ vonalával, amely össze van kötve a soros busz Service Request In vonalával is.

A SENSE vonal, amelyet szintén a processzor érzékel, a kazettás egység állapotszenzora. A WRITE vonalon keresztül továbbítja a processzor az adatbiteket a szalagos egység felé.

2. A 6510-es processzor nyelve és programozása

Az előző fejezetben a számítógép hardverét ismertettük. Egy számítógépet azonban nem elégséges hardver oldalról ismerni. A program legalább olyan fontos, mint az elektronika. Az elektronika a gép teste, a program a „lelke”. Ezért a programozási tudnivalókkal legalább olyan részletesen kell foglalkozni, mint a gép hardverével. A C64-est alapállapotban két szinten lehet programozni, BASIC-ben és gépi nyelven. E fejezetben a gépi nyelvvel kívánunk foglalkozni.

2.1. A gépi nyelv

Gépi nyelv — gépi kód — assembly. Mindhárom meghatározás ugyanarra a programozási megoldásra utal, különbség mégis van közöttük, ezért érdemes mindenekelőtt a fogalmakat tisztázni. Gépi nyelv az a nyelv, amelyet a processzor megért és végre tud hajtani. Mivel az információ átvételében az emberek és a számítógépek között jelentős eltérés van, szükség van arra, hogy az imént definiált gépi nyelv mindkét fél számára érthető legyen. E két vetület elnevezése a gépi kód, illetve az assembly. A gépi kódot a processzor érti, az assemblyt pedig az ember. A gépi kód digitális információ, az assembly pedig alfanumerikus jelölésrendszer.

2.1.1. Az assembly

Az assembly, mint a neve is mutatja (to assemble — szerelni), alkatrészekből áll, amelyeket egymás mellé lehet „szerelni”. Az assembly programozás erősen emlékeztet a közismert LEGO játékra. Az alkatrészek a fantáziától függően véges, de beláthatatlanul sok variációban szerelhetők össze.

Az assembly jelölésrendszer legjelentősebb eleme a *mnemonik*, amely a gépi nyelvű utasítást szimbolizálja. A mnemonik három betűből áll. E három betű többnyire egyértelműen utal az utasítás által végzett műveletre. A jelölésrendszer további fontos eleme a *címzés mód*, pontosabban annak jelölése. Minél több címzési mód lehetséges, a processzor nyelve annál hajlékonyabb.

Az assembly programozás nem nélkülözheti a *címke*zést. A programon belül lehetnek kiemelt részek, illetve belső, kiemelt címek. Azért, hogy ezek egyértelműen megkülönböztethetők legyenek, és helyük rögzített legyen, szükség van azonosításukra. A címke erre szolgál.

A processzort gépi nyelven programozni assembler fordítókkal érdemes. Ezek lefordítják gépi kódra az általunk megírt programot. Az assemblerek szintaxisa általában csak kevésbé különbözik egymástól. A C64-esre számos ilyen fordítóprogram készült már. Minden assembler más-más szintű szolgáltatást nyújt. E szolgáltatások mennyiségben és minőségben is erősen eltérhetnek egymástól, de van néhány elem, amiben minden assembler megegyezik. Ezek között a legelső a programsor szerkezete:

sorszám	címke	mnemonik	címzés mód
---------	-------	----------	------------

A címke alkalmazása nem kötelező.

Általában minden assembler tartalmaz valamilyen *direktívát*. Direktívának azokat az utasításokat nevezzük, amelyek az assemblert utasítják és nem a processzort. Ilyen pl. az egy byte vagy egy szó elhelyezésére szolgáló direktíva. A direktívák a mnemonikok helyére írhatók.

Az assemblerek a fordítást többnyire két menetben végzik (több lehet, kevesebb nem). Az első menetben az assembler felméri a program hosszát, és előállítja a címkéhez tartozó abszolút címeket, amelyeket táblázatban tárol. A második menet során állítja elő a tényleges gépi kódot. Az előállított gépi kód azután vagy közvetlenül a tárba, vagy egy előzőleg megnyitott állományba kerül.

2.1.2. A processzor címzés módjai

A 6510-es processzor sajátossága, hogy a RAM-ot a cím felső byte-ja nélkül is el tudja érni. Igaz, ez csak a nulláslapra vonatkozik. Az egyébként nem túl gyors processzort ezzel a megoldással nagyobb sebességre ösztökélték, mert a címbuszra nem kell kihelyezni a cím felső byte-ját, s a memóriában sem kell helyet biztosítani a cím felső byte-jának. A 6510-es processzor lényegében csak 8-féle címzési módot tud megkülönböztetni, de a kivételekkel ez 12-féle címzésre növekszik. A mindenkori utasításkód 2 — 4 bitjei a címzés meghatározására szolgálnak.

A címzés módok:

000:	indexelt indirekt
001:	nulláslap
010:	közvetlen — beleértett — címzés nélküli
011:	abszolút
100:	indirekt, indexelt — relatív
101:	nulláslap, X — nulláslap, Y
110:	abszolút, Y
111:	abszolút, X

A nulláslapra 5 különböző címzésfajta is mutat. A nulláslap ezért különösen fontos a 6510-es processzor programozásában.

A *közvetlen címzés* az egyszerűbb címzés módok egyike. Az utasításbyte után álló byte tartalma a megfelelő regiszterbe kerül, illetve az aritmetikai művelet operandusa lesz. Az operandus a program része, emiatt változtatása nehézkes. A programszámláló az e címzéssel ellátott utasítás végrehajtása után kettővel növekszik.

Assembler jelölése: LDA # $\$AA$, CMP # $\$FF$, SBC # $\$40$, ORA # $\$20$ stb.

A *nulláslap címzés* a 65XX processzorok sajátossága. Lényegében a tárcím abszolút címzése, de a cím felső byte-ja mindig $\$00$, ha a nulláslapra irányul a címzés. Emiatt a processzor megengedi a felső byte elhagyását. Az operandus annak a tárcímnek a tartalma, amelyre a hivatkozás történik. A tárcím tartalma egyszerűen változtatható. A programszámláló értéke az utasítás végrehajtása után kettővel növekszik.

Assembler jelölése: LDA $\$22$, SBC $\$45$, CMP $\$71$ stb.

Az *abszolút címzés* a tár adott rekeszére vonatkozik. Az utasításkód után 2 byte áll, alsó és felső byte sorrendben. Ez adja meg azt a tárcímet, amelynek tartalma képezi az utasítás operandusát. A programszámláló az utasítás végrehajtása után hárommal növekszik.

Assembler jelölése: LDA $\$4061$, LSR $\$FD20$, ASL $\$1842$ stb.

A *beleértett* vagy *akkumulátor* címzési módban az utasításkód mögött nem áll operandus. Az utasítás az akkumulátorra vonatkozik. Az utasításszámláló értéke eggyel növekszik.

Assembler jelölése: ROL A, LSR A stb.

A *relatív címzés* kizárólag a feltételes ugró utasítások címzési megoldása. A processzor az

elágazás helyének meghatározásához a számítást az utasítás és operandusa első byte-jához képest végzi el. Az utasításslámláló tartalma az ugrásnak megfelelően módosul.

A processzornak fel kell ismernie negatív számokat is, hogy ugrást hátrafelé is végre tudjon hajtani. A szám negatív előjelét a 7. bit adja. Ha magas, a processzor a számot egy negatív szám kettes komplementeként fogja értelmezni. A processzor a kettes komplement számot a következőképpen számítja át negatív számmá:

- a 7. bitet (a jelzőt) alacsonyra állítja;
- a 0–6. biteket invertálja (0 1-re, 1 0-ra változik);
- az eredményhez hozzáad 1-et.

A programozó számára a kettes komplement képzése jelentkezik feladatként (lásd még a *Függelékben*).

Könnyen belátható, hogy 8 biten -128 és $+127$ között ábrázolható negatív és pozitív szám. Az ugróutasítás operandusa is az említett szisztéma szerint értelmezhető. Ha a hetedik bit magas, akkor az ugrás hátrafelé, ha pedig alacsony, akkor előre történik.

A címezsmód assembler szintaxisa: BEQ \$6AF5, BVC \$2758 stb.

Az *X regiszterrel indexelt nulláslap* címezés a nulláslapra irányul, de az utasítás operandusa nem a megadott byte lesz. A byte címéhez hozzáadódik az X regiszter tartalma, s az operandus e két számjegy összegeként előálló tárcím tartalma lesz. Legyen egy utasítás címezése \$73,X, az X regiszter tartalma \$08. Az utasítás operandusa a $(\$73 + \$08 =)$ \$8B tárcím tartalma.

Így egyszerű módon férhetünk hozzá több tárcímhez egyetlen utasítással, anélkül, hogy a programban módosítást kellene végrehajtani. Az utasítások végrehajtása a programszámlálót kettővel növeli.

A címezsmód assembler szintaxisa: LDA \$22,X, EOR \$33,X stb.

Az *Y regiszterrel indexelt nulláslap* címezés mechanizmusát és szintaxisát tekintve azonos az előző pontban ismertetett címezéssel. Az X regisztert az Y regiszter helyettesíti.

Az *X regiszterrel indexelt abszolút* címezés lehetővé teszi, hogy a memória teljes címterületét elérhessük. Működési mechanizmusa egyébként megegyezik a hasonló nulláslap címezés mechanizmusával. Az utasítások végrehajtása után a programszámláló értéke hárommal növekszik.

A címezsmód assembler szintaxisa: ROR \$1170,X, ADC \$806F,X stb.

Az *Y regiszterrel indexelt abszolút* címezés teljes mértékben megegyezik az előző pontban ismertetett címezési móddal. Az X regisztert azonban az Y regiszter helyettesíti.

Az *indexelt indirekt* címezés működése meglehetősen összetett, ezért hosszas magyarázat helyett inkább egy konkrét példa: legyen az utasítás STA (\$22,X), az X regiszter tartalma pedig \$06. A processzor a \$22-höz hozzáadja a \$06-ot. Ez \$28. Az így kapott érték a nulláslapon a \$28–29 regiszterpárra mutat. Innen betölti a címet (ami mondjuk, legyen \$2225), s az akkumulátor tartalmát végül a \$2225 címre helyezi.

Az utasítás igen hajlékony tárkezelést tesz lehetővé, bár csekély az olyan programozási feladatok száma, ahol szükség lehet rá. Az utasítások a programszámlálót kettővel növelik.

A címzés mód assembler szintaxisa: STA (\$22,X), CMP (\$40,X) stb.

Az *indirekt, indexelt* címzés mód is példán keresztül érthető meg a legkönnyebben. Az utasítás LDA (\$FE),Y, az Y regiszter tartalma \$20, a tárcímek tartalma: \$00FE: \$00 — \$00FF: \$80. A processzor betölti a \$FE — FF címeken található címbyte-okat (\$8000), majd ehhez hozzáadja az Y regiszter tartalmát. Az így kapott \$8020 címről tölti fel az akkumulátort. Az utasítás végrehajtása után a programszámláló értéke kettővel növekszik. A címzés segítségével — csak az Y regisztert változtatva — elérhetünk egy egész lapot. A címzés mód kellő hajlékonyságot biztosít programjaink számára, könnyen érthető és követhető, ezért gyakran használjuk.

A címzés assembler szintaxisa: LDA (\$FE),Y, ADC(02),Y stb.

A *címzés nélküli utasítások* jelentős csoportot képeznek; ezek többnyire a processzor belső regisztereivel kapcsolatosak. Végrehajtásuk után a programszámláló értéke eggyel növekszik.

Nem hagyható említés nélkül, hogy a JMP utasításnak van indirekt címzés módja is. A címzés lényegében módosított abszolút címzés. Az utasításkód mögött álló cím azonban nem töltődik be a programszámlálóba, hanem mutatóként szolgál. A mutató által megadott tárcím és a mögötte álló tárcím tartalma töltődik be a programszámlálóba. A processzor a közvetett módon megadott címen folytatja a programvégrehajtást. Ennek a címzés módnak köszönhető, hogy az operációs rendszer és az interpreter nyitott a változtatásra, mert a legfontosabb rendszerrutinok egy indirekt ugrótáblán keresztül érhetők el. Így a ROM-rutinok felcserélhetők saját rutinjainkkal.

2.1.3. A processzor utasításkészlete

Az utasításkészlet két nagy, ezeken belül pedig több kisebb csoportra bontható. Megkülönböztetünk standard és tiltott utasításokat. Az első csoportba azok az utasítások tartoznak, amelyeket a MOS Standard Assembler felismer — ezeket használja a BASIC interpreter és az operációs rendszer gépi nyelvű programja is. A tiltott kódok azok a kódok, amelyek nem tartoznak az első csoportba. Mindkét kódkészlet tovább csoportosítható.

A standard utasításkészlet:

- töltő utasítások (LDA, LDX, LDY)
- tároló utasítások (STA, STX, STY)
- transzfer utasítások (TAX, TXA, TAY, TYA, TXS, TSX)
- logikai utasítások (AND, ORA, EOR)
- aritmetikai utasítások (ADC, SBC, ASL, LSR, ROL, ROR)
- flag (jelző-) utasítások (SEC, SED, SEI, CLC, CLD, CLI, CLV)
- ugró utasítások (JMP, JSR, BNE, BEQ, BCC, BPL, BMI, BVC, BVS)
- számláló utasítások (INC, DEC, INX, INY, DEX, DEY)
- összehasonlító utasítások (CMP, CPX, CPY)

- veremkezelő utasítások (PLA, PHA, PLP, PHP)
- egyéb utasítások (RTS, RTI, BRK, NOP, BIT)

A tiltott kódok:

- amelyek megállítják a processzort;
- amelyek nem végeznek műveletet;
- amelyek két standard műveletet kapcsolnak össze;
- amelyeknek nincs standard megfelelőjük.

A tiltott kódok mnemonikjai (mivel a kódok használaton kívüliek) erősen önkényesek. Az eredeti Commodore-irodalom ezeket nem említi, és ezért irányadó elnevezésük sem létezik. A megadott mnemonikok emiatt nem feltétlenül egyeznek meg más források jelölésével. A mnemonikok megadásánál a döntő szempont az utasításkód által végzett művelet volt, de a szűkös kombinációs lehetőségek miatt eltérések vannak.

A tiltott kódok programozási szempontból nem jelentősek, szerepük minden tekintetben csekély. Olyan esetekben juthatnak szerephez, ahol a program algoritmusának elrejtése, titkosítása a cél. E könyv megjelenése előtt ez a titkosítási mód már értelmét veszítette.

2.2. A processzor utasításai

A MOS 6510-es processzor maximum 256 utasítást képes megkülönböztetni. A 256 utasítás közül egyesek ugyanolyan vagy hasonló műveletet végeznek, ezért azokat összevontan tárgyaljuk. Felsorolásunk tartalmazza mind a standard, mind a tiltott kódokat, s mivel betűrendes sorrendet követünk, a tiltott kódok nincsenek egy csoportban. A tiltott kódokra az utasítás fejlécében hívjuk fel a figyelmet.

A processzor állapotregiszterének bitjeit, illetve azok változását táblázat foglalja össze minden utasításnál. A táblázatban csillag (*) jelzi a változás lehetőségét, ha az nem egyértelmű. Ha a változás egyértelműen 0 vagy 1, azt a megfelelő számjegy mutatja. Ha a bit értéke nem változik, azt – jel jelzi.

Az utasításkód bitjeit szintén táblázat tartalmazza. A táblázat egyszerűsített, ahol nincs minden bit feltüntetve, ott a felső sorban levő bitek az irányadók.

Az utasításkódot decimális (#) formában nem adjuk meg. Erre a gépi nyelvű programozás során gyakorlatilag nincs szükség.

ABS

Tiltott kód

STOP — hatására a processzor felfüggeszti működését. E helyzetéből csak a RESET billentheti ki.

Regiszterek:

PS: A jelzőknek nincs jelentőségük
AC: A regisztereknek nincs jelentőségük

N	V	B	D	I	Z	C
—	—	—	—	—	—	—

Formátum:

7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	0
0	0	0	1	0	0	1	0
0	0	1	0	0	0	1	0
0	0	1	1	0	0	1	0
0	1	0	0	0	0	1	0
0	1	0	1	0	0	1	0
0	1	1	0	0	0	1	0
0	1	1	1	0	0	1	0
1	0	0	1	0	0	1	0
1	0	1	1	0	0	1	0
1	1	0	1	0	0	1	0
1	1	1	1	0	0	1	0

KÓD	SZINTAXIS	CIKLUSIDŐ
\$02	ABS	—
\$12	ABS	—
\$22	ABS	—
\$32	ABS	—
\$42	ABS	—
\$52	ABS	—
\$62	ABS	—
\$72	ABS	—
\$92	ABS	—
\$B2	ABS	—
\$D2	ABS	—
\$F2	ABS	—

AAX cím

Tiltott kód

ACCU AND X — az akkumulátor és az X regiszter közötti AND eredménye az abszolút címre kerül. A jelzők és a regiszterek nem változnak.

Regiszterek:

PS: N: nem változik
V: nem változik
B: nem változik
D: nem változik
I: nem változik
Z: nem változik
C: nem változik

N	V	B	D	I	Z	C
—	—	—	—	—	—	—

AC: nem változik
XR: nem változik
YR: nem változik

SP: nem változik

PC: $PC = PC + 3$

Formátum:

7	6	5	4	3	2	1	0
<hr/>							
1	0	0	0	1	1	1	1

KÓD SZINTAXIS CIKLUSIDŐ

\$8F AAX\$xxxx 4 μ s

ADC operandus

ADDITION WITH CARRY — bitenkénti (bináris) összeadás az akkumulátor és a megcímzett byte között. A Carry flag túlsordulás esetén magas lesz. Összeadás előtt a Carry-t törölni kell, mert értékét hozzáadja az akkuhoz. Az eredmény mindig az akkuban képződik.

Regiszterek:

PS: N: magas, ha az eredmény
7. bitje magas

V: magas, ha a művelet során
volt bináris túlsordulás

B: nem változik

D: nem változik

I: nem változik

Z: magas, ha az eredmény zéró

C: az összeadás eredményétől függően magas, ill. alacsony

N	V	B	D	I	Z	C
*	*	—	—	—	*	*

AC: az eredmény mindig ide kerül

XR: nem változik

YR: nem változik

SP: nem változik

PC: $PC = PC + 2$ (3)

Formátum:

	7	6	5	4	3	2	1	0
	<hr/>							
	0	1	1	címzés		0	1	
indexelt indirekt				0	0	0		
nulláslap				0	0	1		
közvetlen				0	1	0		
abszolút				0	1	1		
indirekt, indexelt				1	0	0		
nulláslap, X				1	0	1		
abszolút, Y				1	1	0		
abszolút, X				1	1	1		

KÓD SZINTAXIS CIKLUSIDŐ

\$61	ADC (\$xx,X)	6 μ s
\$65	ADC \$xx	3 μ s
\$69	ADC #\$xx	2 μ s
\$6D	ADC \$xxxx	4 μ s
\$71	ADC (\$xx),Y	5 μ s (6)
\$75	ADC \$xx,X	4 μ s
\$79	ADC \$xxxx,X	4 μ s (5)
\$7D	ADC \$xxxx,X	4 μ s (5)

AND operandus

AND WITH ACCU — logikai ÉS művelet az akku és a megcímzett byte tartalma között. A művelet eredménye az akkuba kerül. Az N és Z jelzők a művelet eredményének megfelelően állítódnak be.

Regiszterek:

- PS: N: magas, ha az eredmény 7. bitje magas
- V: nem változik
- B: nem változik
- D: nem változik
- I: nem változik
- Z: az eredménytől függően beállítódik
- C: nem változik
- AC: az eredmény mindig ide kerül
- XR: nem változik
- YR: nem változik
- SP: nem változik
- PC: PC=PC+2 (3)

N	V	B	D	I	Z	C
*	—	—	—	—	*	—

Formátum:

	7	6	5	4	3	2	1	0	KÓD	SZINTAXIS	CIKLUSIDŐ
	0	0	1	címzés		0	1				
indexelt indirekt				0	0	0			\$21	AND (\$xx,X)	6µs
nulláslap				0	0	1			\$25	AND \$xx	3µs
közvetlen				0	1	0			\$29	AND #\$xx	2µs
abszolút				0	1	1			\$2D	AND \$xxxx	4µs
indirekt, indexelt				1	0	0			\$31	AND (\$xx),Y	5µs (6)
nulláslap, X				1	0	1			\$35	AND \$xx,X	4µs
abszolút, Y				1	1	0			\$39	AND \$xxxx,Y	4µs (5)
abszolút, X				1	1	1			\$3D	AND \$xxxx,X	4µs (5)

ANL byte

Tiltott kód

AND WITH ACCU, LOGICAL SHIFT RIGHT — logikai ÉS művelet után biteltolás jobbra. Az akku és a címzésben szereplő byte közötti AND művelet eredményén, amely az akkuban képződik, végrehajtódik az LSR művelet is. Az N jelző az LSR miatt mindig 0 lesz, a Z jelző az eredménytől függően állítódik be. A Carry az AND művelet eredményének legalsó bitjét veszi át.

Regiszterek:

PS: N: mindig nulla
 V: nem változik
 B: nem változik
 D: nem változik
 I: nem változik
 Z: magas, ha az eredmény zéró
 C: a biteltolás alulcsordulásától függően magas vagy alacsony

N	V	B	D	I	Z	C
0	—	—	—	—	*	—

AC: ide kerül az eredmény
 XR: nem változik
 YR: nem változik
 SP: nem változik
 PC: $PC = PC + 2$

Formátum:

7 6 5 4 3 2 1 0
 —————
 0 1 0 0 1 0 1 1

KÓD SZINTAXIS CIKLUSIDŐ
 \$4B ANL # \$xx 2 μ s

ANN byte

Tiltott kód

AND WITH ACCU, NEGATIVE TO CARRY — logikai ÉS művelet, a negatív eredményt a Carry is jelzi. Az akku és a megcímzett byte közötti AND művelet eredménye az akkuban keletkezik. A Z jelző az eredménytől függően állítódik be. Ha az eredmény 7. bitje magas, akkor nemcsak az N jelző, hanem a Carry is magasra vált. Ellenkező esetben alacsony marad.

Regiszterek

PS: N: az eredménytől függően
 magas vagy alacsony
 V: nem változik
 B: nem változik
 D: nem változik
 I: nem változik
 Z: az eredménytől függően állítódik be
 C: a Negative jelzővel azonos állapotot vesz fel

N	V	B	D	I	Z	C
*	—	—	—	—	*	*

AC: itt képződik az eredmény
 XR: nem változik
 YR: nem változik
 SP: nem változik
 PC: $PC = PC + 2$

Formátum:

7	6	5	4	3	2	1	0	KÓD	SZINTAXIS	CIKLUSIDŐ
0	0	0	0	1	0	1	1	\$0B	ANN # \$xx	2µs
0	0	1	0	1	0	1	1	\$2B	ANN # \$xx	2µs

ANX byte

Tiltott kód

AND WITH ACCU, TRANSFER TO X — logikai ÉS művelet, az akku tartalma az X-be kerül. Az operandus és %11101110 (\$EE) között végrehajtott AND, valamint az akku és %00010001 (\$11) között végrehajtott AND művelet eredményei közötti logikai VAGY (OR) művelet után az eredmény előbb az akkuba, majd onnan az X regiszterbe kerül. Az eredménytől függően változnak az N és Z jelzők.

Regiszterek:

- PS: N: az eredménytől függően változik
- V: nem változik
- B: nem változik
- D: nem változik
- I: nem változik
- Z: az eredménytől függően változik
- C: nem változik

N	V	B	D	I	Z	C
*	—	—	—	—	*	—

- AC: az eredmény itt keletkezik
- XR: átveszi az akku tartalmát
- YR: nem változik
- SP: nem változik
- PC: PC=PC+2

Formátum:

7	6	5	4	3	2	1	0	KÓD	SZINTAXIS	CIKLUSIDŐ
1	0	1	0	1	0	1	1	\$AB	ANX # \$xx	2µs

ASL operandus

ARITHMETIC SHIFT LEFT — biteltolás balra. A megcímzett byte tartalmát 2-vel szorozza, azaz minden bitet egy helyiértékkel feljebb mozgat. A legalsó bit automatikusan 0 lesz, a legfelső bitet a Carry veszi át. Az eredmény az N és Z jelzőket beállítja.

Regiszterek:

PS: N: az eredménytől függően beállítódik
 V: nem változik
 B: nem változik
 D: nem változik
 I: nem változik
 Z: az eredménytől függően beállítódik
 C: az operandus legfelső bitjét veszi át

N	V	B	D	I	Z	C
*	—	—	—	—	*	*

AC: csak a beleértett címzés módnál változhat
 XR: nem változik
 YR: nem változik
 SP: nem változik
 PC: $PC = PC + 2$ (3)

Formátum:

	7	6	5	4	3	2	1	0
	0	0	0	címzés			1	0
nulláslap				0	0	1		
beleértett				0	1	0		
abszolút				0	1	1		
nulláslap,X				1	0	1		
abszolút,X				1	1	1		

KÓD SZINTAXIS CIKLUSIDŐ

\$06	ASL \$xx	5 μ s
\$0A	ASL A	2 μ s
\$0E	ASL \$xxxx	6 μ s
\$16	ASL \$xx,X	6 μ s
\$1E	ASL \$xxxx,X	7 μ s

*ASO operandus**Tiltott kód*

ARITHMETIC SHIFT LEFT, OR ACCU — biteltolás balra, logikai VAGY művelet az akkuval. Biteltolás balra a megcímzett byte-on, a byte 7. bitjét átveszi a Carry jelző. A megcímzett byte módosított tartalma és az akku tartalma között logikai VAGY művelet hajtodik végre. Az eredmény az akkuban képződik. Az N és Z jelzők az eredmény alapján állítódnak be.

Regiszterek:

PS: N: az akku tartalma szerint állítódik be
 V: nem változik
 B: nem változik
 D: nem változik
 I: nem változik
 Z: az akku tartalma szerint állítódik be
 C: a megcímzett byte 7. bitjét veszi át

N	V	B	D	I	Z	C
*	—	—	—	—	*	*

AC: az eredmény itt keletkezik
XR: nem változik
YR: nem változik
SP: nem változik
PC: PC=PC+2 (3)

Formátum:

	7	6	5	4	3	2	1	0	KÓD	SZINTAXIS	CIKLUSIDŐ
	0	0	0	címzés			1	1			
indexelt indirekt				0	0	0			\$03	ASO (\$xx,X)	8µs
nulláslap				0	0	1			\$07	ASO \$xx	5µs
abszolút				0	1	1			\$0F	ASO \$xxxx	6µs
indirekt, indexelt				1	0	0			\$13	ASO (\$xx),Y	8µs
nulláslap,X				1	0	1			\$17	ASO \$xx,X	6µs
abszolút,Y				1	1	0			\$1B	ASO \$xxxx,Y	7µs
abszolút,X				1	1	1			\$IF	ASO \$xxxx,X	7µs

AXI operandus

Tiltott kód

Logikai ÉS művelet az X regiszterrel. Az akku és az X regiszter közötti logikai ÉS művelet eredménye az akkuba kerül. Az akku és a megcímzett byte után álló byte tartalma közötti logikai ÉS eredménye a megcímzett byte-ba kerül. A jelzők nem változnak.

Regiszterek:

PS: N: nem változik
V: nem változik
B: nem változik
D: nem változik
I: nem változik
Z: nem változik
C: nem változik

N	V	B	D	I	Z	C
—	—	—	—	—	—	—

AC: részleteredmény keletkezik benne
XR: nem változik
YR: nem változik
SP: nem változik
PC: PC=PC+2

Formátum:

	7	6	5	4	3	2	1	0	KÓD	SZINTAXIS	CIKLUSIDŐ
	1	0	0	címzés		1	1				
indirekt, indexelt				1	0	0			\$93	AXI (\$xx),Y	6 μ s

AXR operandus*Tiltott kód*

AND WITH X REGISTER — logikai ÉS az akku és az X regiszter között. Az eredmény a megcímzett byte-ba kerül. A jelzők és a regiszterek nem változnak.

Regiszterek:

PS: N: nem változik
 V: nem változik
 B: nem változik
 D: nem változik
 I: nem változik
 Z: nem változik
 C: nem változik

N	V	B	D	I	Z	C
—	—	—	—	—	—	—

AC: nem változik
 XR: nem változik
 XR: nem változik
 SP: nem változik
 PC: PC=PC+2

Formátum:

	7	6	5	4	3	2	1	0	KÓD	SZINTAXIS	CIKLUSIDŐ
	1	0	0	címzés		1	1				
nulláslap				0	0	1			\$87	AXR \$xx	3 μ s

AXS operandus*Tiltott kód*

Az akku és az X regiszter közötti logikai ÉS eredménye a veremmutatóba kerül. A veremmutató és a címoperandus felső byte-ja+1 közötti logikai ÉS művelet eredménye kerül a megcímzett byte-ba. A jelzők nem változnak. (Pl. ha a címzés \$ACF2, akkor A AND X \rightarrow SP, majd SP AND #\$AD \rightarrow \$ACF2,Y.)

Regiszterek:

PS: N: nem változik
 V: nem változik
 B: nem változik

N	V	B	D	I	Z	C
—	—	—	—	—	—	—

- D: nem változik
- I: nem változik
- Z: nem változik
- C: nem változik
- AC: nem változik
- XR: nem változik
- YR: nem változik
- SP: az akku és az X közötti AND értékét veszi át
- PC: PC=PC+3

Formátum:

7	6	5	4	3	2	1	0	KÓD	SZINTAXIS	CIKLUSIDŐ
1	0	0	címzés	1	1			\$9B	AXS \$xxxx,Y	5µs
			1	1	0					

AXX operandus Tiltott kód

Az akku és az XR közötti logikai ÉS; az eredmény a megcímzett byte-ba kerül. A jelzők nem változnak.

Regiszterek:

- PS: N: nem változik
- V: nem változik
- B: nem változik
- D: nem változik
- I: nem változik
- Z: nem változik
- C: nem változik

N	V	B	D	I	Z	C
—	—	—	—	—	—	—

- AC: nem változik
- XR: nem változik
- YR: nem változik
- SP: nem változik
- PC: PC=PC+2

Formátum:

7	6	5	4	3	2	1	0	KÓD	SZINTAXIS	CIKLUSIDŐ
1	0	0	címzés	1	1			\$83	AXX (\$xx,X)	6µs
			0	0	0					

indexelt indirekt

AXY operandus*Tiltott kód*

Logikai ÉS művelet az akku és az X regiszter között. Az eredmény a megcímzett byte-ba kerül. A jelzők nem változnak.

Regiszterek:

PS: N: nem változik
 V: nem változik
 B: nem változik
 D: nem változik
 I: nem változik
 Z: nem változik
 C: nem változik

AC: nem változik
 XR: nem változik
 YR: nem változik
 SP: nem változik
 PC: $PC = PC + 2$

N	V	B	D	I	Z	C
—	—	—	—	—	—	—

Formátum:

	7	6	5	4	3	2	1	0
	1	0	0	címzés		1	1	
nulláslap, Y				1	0	1		

KÓD	SZINTAXIS	CIKLUSIDŐ
-----	-----------	-----------

\$97	AXY \$xx, Y	4 μ s
------	-------------	-----------

AYY operandus*Tiltott kód*

Logikai ÉS művelet az akku és az X regiszter között. Kettős logikai ÉS: az akku és az X regiszter közötti ÉS művelet eredménye és a cím felső byte-ja +1 közötti újabb ÉS művelet eredménye kerül a címzésben meghatározott tárcímre. A jelzők nem módosulnak. (Pl. ha a cím \$3F00, akkor $AC \text{ AND } XR \rightarrow \text{közbenső tároló}$; majd $\text{tároló AND } (\$3F + 1) \rightarrow \$3F00, Y$)

Regiszterek:

PS: N: nem változik
 V: nem változik
 B: nem változik
 D: nem változik
 I: nem változik
 Z: nem változik
 C: nem változik

N	V	B	D	I	Z	C
—	—	—	—	—	—	—

AC: nem változik
 XR: nem változik
 YR: nem változik
 SP: nem változik
 PC: $PC = PC + 3$

Formátum:

	7	6	5	4	3	2	1	0	
	1	0	0	címzés			1	1	
abszolút, Y				1	1	1			
									<i>KÓD SZINTAXIS CIKLUSIDŐ</i>
									\$9F AYY \$xxxx,Y 5μs

Bxx byte

BRANCH ON xx — ugrás, ha a feltétel teljesült: ha a processzor állapotregiszterében a megfelelő bit a feltételnek eleget tesz, ugrás következik be a byte értékének megfelelően (+127...–128).

BCC: Ugrás, ha a Carry alacsony
 BCS: Ugrás, ha a Carry magas
 BEQ: Ugrás, ha a Zero magas
 BNE: Ugrás, ha a Zero alacsony
 BMI: Ugrás, ha a Negative magas
 BPL: Ugrás, ha a Negative alacsony
 BVC: Ugrás, ha az Overflow alacsony
 BVS: Ugrás, ha az Overflow magas

Regiszterek:

PS: N: nem változik
 V: nem változik
 B: nem változik
 D: nem változik
 I: nem változik
 Z: nem változik
 C: nem változik

N	V	B	D	I	Z	C
—	—	—	—	—	—	—

AC: nem változik
 XR: nem változik
 YR: nem változik
 SP: nem változik
 PC: $PC = PC + 2 \pm \text{operandus}$

Formátum:

7 6 5 4 3 2 1 0

feltétel 1 0 0 0 0

0 0 0

0 0 1

0 1 0

0 1 1

1 0 0

1 0 1

1 1 0

1 1 1

KÓD SZINTAXIS* CIKLUSIDŐ**

\$10 BPL \$xxxx 2μs

\$30 BMI \$xxxx 2μs

\$50 BVC \$xxxx 2μs

\$70 BVS \$xxxx 2μs

\$90 BCC \$xxxx 2μs

\$B0 BCS \$xxxx 2μs

\$D0 BNE \$xxxx 2μs

\$F0 BEQ \$xxxx 2μs

BIT cím

BIT TEST — bitvizsgálat. Az akkumulátor és a megcímzett byte között logikai ÉS kapcsolat létesül. Ennek eredménye alapján állítódik be a Z jelző. A megcímzett byte, valamint az akku tartalma nem változik. Az állapotregiszter 6. és 7. bitje átveszi a megcímzett byte 6. és 7. bitjét.

Regiszterek:

PS: N: átveszi a tárcím 7. bitjét

V: átveszi a tárcím 6. bitjét

B: nem változik

D: nem változik

I: nem változik

Z: módosulhat

C: nem változik

N	V	B	D	I	Z	C
*	*	—	—	—	*	—

AC: nem változik

XR: nem változik

YR: nem változik

SP: nem változik

PC: PC=PC+2 (3)

Formátum:

7 6 5 4 3 2 1 0

0 0 1 címzés 0 0

nulláslap 0 0 1

abszolút 0 1 1

KÓD SZINTAXIS CIKLUSIDŐ

\$24 BIT \$xx 3μs

\$2C BIT \$xxxx 4μs

* A szintaxisok assembler szintaxisok.

** A ciklusidő 1 s-mal növekszik, ha a feltétel teljesült, és további 1 s-mal, ha az ugrás laphatárt lép át.

BRK

BREAK — megszakításkérelem programon belül. Az utasítás feldolgozásakor a processzor először alacsonyra állítja az állapotregiszter 4. bitjét, és magasra 2. bitjét. Ezután a programszámlálót, majd az állapotregisztert a verembe menti, és a veremmutatót hárommal csökkenti. Betölti a programszámlálóba a \$FFFE — FFFF tárcímeken található címet, és az utasításvégrehajtást innen folytatja. Ezt követően végrehajtódik a megszakítási rutin. A megszakítás nem tiltható le az állapotregiszter 2. bitjének magasra állításával, az utasítást a processzor mindenképpen végrehajtja.

Regiszterek:

- PS: N: nem változik
- V: nem változik
- B: alacsonyra vált
- D: nem változik
- I: magasra vált
- Z: nem változik
- C: nem változik

N	V	B	D	I	Z	C
—	—	0	—	1	—	—

- AC: nem változik
- XR: nem változik
- YR: nem változik
- SP: SP = SP - 3
- PC: felveszi a \$FFFE — FFFF címeken található byte-ok értékét

Formátum:

7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0

KÓD	SZINTAXIS	CIKLUSIDŐ
\$00	BRK	7µs

CLx

CLEAR x — a megadott jelző törlése. (Az állapotregiszter többi bitjére és a regiszterekre nem hat.)

Regiszterek:

- PS: N: nem változik
- V: változhat
- B: nem változik
- D: változhat
- I: változhat
- Z: nem változik
- C: változhat

	N	V	B	D	I	Z	C
CLC	—	—	—	—	—	—	0
CLD	—	—	—	0	—	—	—
CLI	—	—	—	—	0	—	—
CLV	—	0	—	—	—	—	—

AC: nem változik
 XR: nem változik
 YR: nem változik
 SP: nem változik
 PC: $PC = PC + 1$

Formátum:

7 6 5 4 3 2 1 0

0 0 0 1 1 0 0 0

1 1 0

0 1 0

1 0 1

KÓD SZINTAXIS CIKLUSIDŐ

\$18 CLC 2 μ s

\$D8 CLD 2 μ s

\$58 CLI 2 μ s

\$B8 CLV 2 μ s

CLC: Törli az átvitelbitet.

CLD: Törli a Decimális üzemmódot.

CLI: Törli az Interrupt bitet; engedélyezi a megszakításokat.

CLV: Törli a túlsordulásbitet.

CMP operandus

COMPARE — összehasonlítja az akkumulátor és a megcímzett byte tartalmát.

Regiszterek:

PS: N: az összehasonlítás eredménye szerint változik

V: nem változik

B: nem változik

D: nem változik

I: nem változik

Z: az összehasonlítás eredménye szerint változik

C: az összehasonlítás eredménye szerint változik

N	V	B	D	I	Z	C
*	—	—	—	—	*	*

AC: nem változik

XR: nem változik

YR: nem változik

SP: nem változik

PC: $PC = PC + 2$ (3)

Formátum:

	7	6	5	4	3	2	1	0	KÓD	SZINTAXIS	CIKLUSIDŐ
	1	1	0	címrész			0	1			
indexelt indirekt				0	0	0			\$C1	CMP (\$xx,X)	6 μ s
nulláslap				0	0	1			\$C5	CMP \$xx	3 μ s
közvetlen				0	1	0			\$C9	CMP #\$xx	2 μ s
abszolút				0	1	1			\$CD	CMP \$xxxx	4 μ s
indirekt, indexelt				1	0	0			\$D1	CMP (\$xx),Y	5 μ s (6)
nulláslap, X				1	0	1			\$D5	CMP \$xx,X	4 μ s
abszolút, Y				1	1	0			\$D9	CMP \$xxxx,Y	4 μ s (5)
abszolút, X				1	1	1			\$DD	CMP \$xxxx,X	4 μ s (5)

Az akku tartalmából kivonja a tárcím tartalmát. A következő esetek lehetségesek:

- a) az akku és a byte tartalma egyenlő: N=0, Z=1, C=1,
- b) az akku tartalma nagyobb: N=0, Z=0, C=1,
- c) az akku tartalma kisebb: N=1, Z=0, C=0.

CPX operandus

COMPARE XR — összehasonlítja az X regiszter és a megcímezett tárhely tartalmát.

Regiszterek:

- PS: N: az összehasonlítás
eredményétől függően változik
- V: nem változik
- B: nem változik
- D: nem változik
- I: nem változik
- Z: az összehasonlítás eredményétől függően változik
- C: az összehasonlítás eredményétől függően változik

N	V	B	D	I	Z	C
*	—	—	—	—	*	*

- AC: nem változik
- XR: nem változik
- YR: nem változik
- SP: nem változik
- PC: PC=PC+2 (3)

Formátum:

	7	6	5	4	3	2	1	0	KÓD	SZINTAXIS	CIKLUSIDŐ
	1	1	1	címzés		0	0				
közvetlen				0	0	0			\$E0	CPX \$#xx	2 μ s
nulláslap				0	0	1			\$E4	CPX \$xx	3 μ s
abszolút				0	1	1			\$EC	CPX \$xxxx	4 μ s

Az X regiszter tartalmából kivonja a tárcím tartalmát. A következő esetek lehetségesek:

a) az X regiszter és a tárcím tartalma egyenlő: N=0, Z=1, C=1,

b) az X regiszter nagyobb: N=0, Z=0, C=1,

c) az X regiszter kisebb: N=1, Z=0, C=0.

CPY operandus

COMPARE YR — összehasonlítja az Y regiszter és a megcímzett byte tartalmát.

Regiszterek:

PS: N: az összehasonlítás eredményétől függően módosul

V: nem változik

B: nem változik

D: nem változik

I: nem változik

Z: az összehasonlítás eredményétől függően módosul

C: az összehasonlítás eredményétől függően módosul

N	V	B	D	I	Z	C
*	—	—	—	—	*	*

AC: nem változik

XR: nem változik

YR: nem változik

SP: nem változik

PC: PC=PC+2 (3)

Formátum:

	7	6	5	4	3	2	1	0	KÓD	SZINTAXIS	CIKLUSIDŐ
	1	1	0	címzés		0	0				
közvetlen				0	0	0			\$C0	CPY # \$xx	2 μ s
nulláslap				0	0	1			\$C4	CPY \$xx	3 μ s
abszolút				0	1	1			\$CC	CPY \$xxxx	4 μ s

Az Y regiszter tartalmából kivonja a tárcím tartalmát. A következő esetek lehetségesek:

- a) az Y regiszter és a tárcím tartalma egyenlő: $N=0, Z=1, C=1$,
- b) az Y regiszter nagyobb: $N=0, Z=0, C=1$,
- c) az Y regiszter kisebb: $N=1, Z=0, C=0$.

DAR byte

Tiltott kód

Logikai ÉS művelet; ciklikus forgatás jobbra, ha a Decimal alacsony.

Regiszterek:

PS: N: az eredménytől függően módosul
 V: az akku 5. és 6. bitje közötti
 EOR eredményét veszi át
 B: nem változik
 D: nem változik
 I: nem változik
 Z: az eredménytől függően módosul
 C: az akku 0. bitjét veszi át

N	V	B	D	I	Z	C
*	*	—	—	—	*	*

AC: itt keletkezik az eredmény

XR: nem változik

YR: nem változik

SP: nem változik

PC: $PC = PC + 2$

Formátum:

7 6 5 4 3 2 1 0
 0 1 1 0 1 0 1 1

KÓD SZINTAXIS CIKLUSIDŐ

\$6B DAR # \$xx 2 μ s

Ez az egyik legbonyolultabb utasítás. Csak akkor hajtódik végre, ha a Decimal flag alacsony. Ekkor előbb egy logikai ÉS műveletet végez az akku és a byte között. Ezután az akkun egy ROR hajtódik végre, ezzel egy időben az akku 5. és 6. bitje közötti EOR eredménye kerül az Overflow (V) jelzőbe. (A bitek sorszáma a ROR utáni állapotra érvényes.)

Az N és Z jelzők az akku tartalma szerint állnak be, a Carry a ROR miatt az akku művelet előtti 0. bitjét veszi át.

DEC operandus

DECREMENT — a tárcím tartalmát eggyel csökkenti. Az N és Z jelzőket a tárhely tartalma szerint állítja be.

Regiszterek

PS: N: ha a művelet eredményének legfelső bitje 1, magas
 V: nem változik
 B: nem változik
 D: nem változik
 I: nem változik
 Z: ha a művelet eredménye zero, magas
 C: nem változik

AC: nem változik
 XR: nem változik
 YR: nem változik
 SP: nem változik
 PC: $PC = PC + 2$ (3)

N	V	B	D	I	Z	C
*	—	—	—	—	*	—

Formátum:

	7	6	5	4	3	2	1	0
	1	1	0	címezés			1	0
nulláslap				0	0	1		
abszolút				0	1	1		
nulláslap,X				1	0	1		
abszolút,X				1	1	1		

KÓD SZINTAXIS CIKLUSIDŐ

\$C6	DEC \$xx	5 μ s
\$CE	DEC \$xxxx	6 μ s
\$D6	DEC \$xx,X	6 μ s
\$DE	DEC \$xxxx,X	7s

DEM operandus**Tiltott kód**

DECREMENT AND COMPARE — dekrementálás, majd összehasonlítás. A megcímezett byte-ot dekrementálja, s az eredményt a CMP utasítás szerint összehasonlítja az akkumulátorral (ld. még CMP).

Regiszterek:

PS: N: az eredménynek megfelelően módosul
 V: nem változik
 B: nem változik
 D: nem változik
 I: nem változik

N	V	B	D	I	Z	C
*	—	—	—	—	*	*

Z: az eredménynek megfelelően módosul

C: az eredménynek megfelelően módosul

AC: nem változik

XR: nem változik

YR: nem változik

SP: nem változik

PC: $PC = PC + 2$ (3)

Formátum:

7 6 5 4 3 2 1 0

1 1 0 címzés 1 1

indexelt indirekt

0 0 0

nulláslap

0 0 1

abszolút

0 1 1

indirekt, indexelt

1 0 0

nulláslap,X

1 0 1

abszolút,Y

1 1 0

abszolút,X

1 1 1

KÓD SZINTAXIS CIKLUSIDŐ

\$C3 DEM (\$xx,X) 8 μ s

\$C7 DEM \$xx 5 μ s

\$CF DEM \$xxxx 6 μ s

\$D3 DEM (\$xx),Y 8 μ s

\$D7 DEM \$xx,Y 6 μ s

\$DB DEM \$xxxx,Y 7 μ s

\$DF DEM \$xxxx,X 7 μ s

DEX

DECREMENT XR — az X regiszter dekrementálása. Ha \$01 volt, a Z jelző magasra vált. Ha \$00 volt, az N jelző vált magasra, az új tartalom pedig \$FF lesz.

Regiszterek:

PS: N: a művelet eredménye alapján módosul

V: nem változik

B: nem változik

D: nem változik

I: nem változik

Z: a művelet eredménye alapján módosul

C: nem változik

N	V	B	D	I	Z	C
*	—	—	—	—	*	—

AC: nem változik

XR: eggyel csökken

YR: nem változik

SP: nem változik

PC: $PC = PC + 1$

Formátum:

7	6	5	4	3	2	1	0
<hr/>							
1	1	0	0	1	0	1	0

<i>KÓD</i>	<i>SZINTAXIS</i>	<i>CIKLUSIDŐ</i>
\$CA	DEX	2 μ s

DEY

DECREMENT YR — az Y regiszter dekrementálása. Ha \$01 volt, a Z jelző magasra vált. Ha az YR \$00 volt, az új tartalom \$FF lesz, és az N jelző vált magasra.

Regiszterek:

PS: N: az eredménytől függően módosul
 V: nem változik
 B: nem változik
 D: nem változik
 I: nem változik
 Z: az eredménytől függően módosul
 C: nem változik

N	V	B	D	I	Z	C
*	—	—	—	—	*	—

AC: nem változik
 XR: nem változik
 YR: eggyel csökken
 SP: nem változik
 PC: PC=PC+1

Formátum:

7	6	5	4	3	2	1	0
<hr/>							
1	0	0	0	1	0	0	0

<i>KÓD</i>	<i>SZINTAXIS</i>	<i>CIKLUSIDŐ</i>
\$88	DEY	2 μ s

EOR operandus

EXCLUSIVE OR — kizáró VAGY (logikai művelet) az akku és a megcímzett byte között. Az eredmény az akkuban képződik, az N és Z jelzők módosulnak.

Regiszterek:

PS: N: a művelet eredményétől
 függően módosul
 V: nem változik
 B: nem változik
 D: nem változik
 I: nem változik

N	V	B	D	I	Z	C
*	—	—	—	—	*	—

Z: a művelet eredményétől függően módosul

C: nem változik

AC: az eredmény itt keletkezik

XR: nem változik

YR: nem változik

SP: nem változik

PC: $PC = PC + 2$ (3)

Formátum:

7 6 5 4 3 2 1 0

0 1 0 címzés 0 1

indexelt indirekt

0 0 0

nulláslap

0 0 1

közvetlen

0 1 0

abszolút

0 1 1

indirekt, indexelt

1 0 0

nulláslap,X

1 0 1

abszolút,Y

1 1 0

abszolút,X

1 1 1

KÓD SZINTAXIS CIKLUSIDŐ

\$41 EOR (\$xx,X) 6 μ s

\$45 EOR \$xx 3 μ s

\$49 EOR #\$xx 2 μ s

\$4D EOR \$xxxx 4 μ s

\$51 EOR (\$xx),Y 5 μ s (6)

\$55 EOR \$xx,X 4 μ s

\$59 EOR \$xxxx,Y 4 μ s (5)

\$5D EOR \$xxx,X 4 μ s (5)

INB operandus

Tiltott kód

INCREMENT AND SUBTRACT — inkrementálás és kivonás. A megcímzett byte-ot inkrementálja, majd az eredményt kivonja az akkumulátorból. A Carry jelző a kivonásnál megszokott szerepet játssza. Az N és Z jelzők a művelet eredményének megfelelő állapotot veszik fel.

Regiszterek:

PS: N: a művelet eredményétől függően módosul

V: nem változik

B: nem változik

D: nem változik

I: nem változik

Z: a művelet eredményétől függően módosul

C: a kivonás alulcsordulása

N	V	B	D	I	Z	C
*	—	—	—	—	*	*

AC: az eredmény itt képződik
 XR: nem változik
 YR: nem változik
 SP: nem változik
 PC: $PC = PC + 2$ (3)

Formátum:

	7	6	5	4	3	2	1	0	KÓD	SZINTAXIS	CIKLUSIDŐ
	1	1	1	címrész			1	1			
indexelt indirekt				0	0	0			\$E3	INB (\$xx,X)	6 μ s
nulláslap				0	0	1			\$E7	INB \$xx	5 μ s
abszolút				0	1	1			\$EF	INB \$xxxx	6 μ s
indirekt, indexelt				1	0	0			\$F3	INB (\$xx),Y	5 μ s
nulláslap,X				1	0	1			\$F7	INB \$xx,X	6 μ s
abszolút, Y				1	1	0			\$FB	INB \$xxxx,Y	7 μ s
abszolút,X				1	1	1			\$FF	INB \$xxxx,X	7 μ s

INC operandus

INCREMENT — a megcímezett tárhely tartalmát eggyel növeli. Az N jelző magasra vált, ha a tárhely tartalma meghaladta a \$7F-et. A Z jelző magas lesz, ha a művelet előtt a tárhely tartalma \$FF volt.

Regiszterek:

PS: N: a művelet eredményétől
 függően módosul
 V: nem változik
 B: nem változik
 D: nem változik
 I: nem változik
 Z: a művelet eredményétől függően módosul
 C: nem változik

N	V	B	D	I	Z	C
*	—	—	—	—	*	—

AC: nem változik
 XR: nem változik
 YR: nem változik
 SP: nem változik
 PC: $PC = PC + 2$ (3)

Formátum:

	7	6	5	4	3	2	1	0	KÓD	SZINTAXIS	CIKLUSIDŐ
	1	1	1	címzés		1	0				
nulláslap				0	0	1			\$E6	INC \$xx	5 μ s
abszolút				0	1	1			\$EE	INC \$xxxx	6 μ s
nulláslap,X				1	0	1			\$F6	INC \$xx,X	6 μ s
abszolút,X				1	1	1			\$FE	INC \$xxxx,X	7 μ s

INX

INCREMENT XR — az X regiszter tartalmát eggyel növeli. Ha eredetileg \$7F volt, az N jelző, ha eredetileg \$FF volt, a Z jelző vált magasra.

Regiszterek:

PS: N: ha az XR 7. bitje 1, magas
 V: nem változik
 B: nem változik
 D: nem változik
 I: nem változik
 Z: ha az XR zéró, magas
 C: nem változik

N	V	B	D	I	Z	C
*	—	—	—	—	*	—

AC: nem változik
 XR: eggyel növekszik
 YR: nem változik
 SP: nem változik
 PC: PC = PC + 1

Formátum:

7	6	5	4	3	2	1	0	KÓD	SZINTAXIS	CIKLUSIDŐ
1	1	1	0	1	0	0	0	\$E8	INX	2 μ s

INY

INCREMENT YR — az Y regiszter tartalmát eggyel növeli. Ha eredetileg \$7F volt, az N jelző, ha eredetileg \$FF volt, a Z jelző vált magasra.

Regiszterek:

PS: N: ha az YR 7. bitje 1, magasra állítódik
 V: nem változik
 B: nem változik

N	V	B	D	I	Z	C
*	—	—	—	—	*	—

D: nem változik
 I: nem változik
 Z: ha az YR zéró, magas
 C: nem változik

AC: nem változik
 XR: nem változik
 YR: eggyel növekszik
 SP: nem változik
 PC: $PC = PC + 1$

Formátum:

7	6	5	4	3	2	1	0
1	1	0	0	1	0	0	0

KÓD	SZINTAXIS	CIKLUSIDŐ
\$C8	INY	2 μ s

JMP cím

JUMP — feltétel nélküli ugrás a megadott címre. Ezt a címet a programszámlálóba (PC) tölti, s a program végrehajtását onnan folytatja. Az indirekt címezés egy vektorra mutat. Jelzők nem változnak.

Regiszterek:

PS: N: nem változik
 V: nem változik
 B: nem változik
 D: nem változik
 I: nem változik
 Z: nem változik
 C: nem változik

N	V	B	D	I	Z	C
—	—	—	—	—	—	—

AC: nem változik
 XR: nem változik
 YR: nem változik
 SP: nem változik
 PC: átveszi a cím értékét

Formátum:

	7	6	5	4	3	2	1	0
	0	1	0	1	1	0	0	
abszolút			0					
indirekt			1					

KÓD	SZINTAXIS	CIKLUSIDŐ
\$4C	JMP \$xxxx	3 μ s
\$6C	JMP (\$xxxx)	5 μ s

JSR cím

JUMP TO SUBROUTINE — szubrutinhívás: feltétel nélküli ugrás a megadott címre a visszatérés biztosításával. Az ugrás végrehajtása előtt a programszámláló pillanatnyi értékéhez hozzáad kettőt, majd a címet alsó-felső byte formában elhelyezi a veremben. A jelzők nem változnak.

Regiszterek:

PS: N: nem változik
 V: nem változik
 B: nem változik
 D: nem változik
 I: nem változik
 Z: nem változik
 C: nem változik

AC: nem változik
 XR: nem változik
 YR: nem változik
 SP: SP = SP - 2
 PC: átveszi a cím értékét

N	V	B	D	I	Z	C
—	—	—	—	—	—	—

Formátum:

7	6	5	4	3	2	1	0
<hr/>							
0	0	1	0	0	0	0	0

KÓD	SZINTAXIS	CIKLUSIDŐ
\$20	JSR \$xxxx	6μs

LDA operandus

LOAD ACCU — megcímzett byte tartalmát betölti az akkumulátorba. Az N és Z jelzők módosulhatnak.

Regiszterek:

PS: N: magas, ha az akku 7. bitje 1
 V: nem változik
 B: nem változik
 D: nem változik
 I: nem változik
 Z: magas, ha az akku zéró
 C: nem változik

N	V	B	D	I	Z	C
*	—	—	—	—	*	—

AC: felveszi a tárcím tartalmát
 XR: nem változik
 YR: nem változik
 SP: nem változik
 PC: $PC = PC + 2$ (3)

Formátum:

	7	6	5	4	3	2	1	0	KÓD	SZINTAXIS	CIKLUSIDŐ
	1	0	1	címzés			0	1			
indexelt indirekt				0	0	0			\$A1	LDA (\$xx,X)	6 μ s
nulláslap				0	0	1			\$A5	LDA \$xx	3 μ s
közvetlen				0	1	0			\$A9	LDA #\$xx	2 μ s
abszolút				0	1	1			\$AD	LDA \$xxxx	4 μ s
indirekt, indexelt				1	0	0			\$B1	LDA (\$xx),Y	5 μ s (6)
nulláslap,X				1	0	1			\$B5	LDA \$xx,X	4 μ s
abszolút,Y				1	1	0			\$B9	LDA \$xxxx,Y	4 μ s (5)
abszolút,X				1	1	1			\$BD	LDA \$xxxx,X	4 μ s (5)

LDT operandus*Tiltott kód*

A megcímzett byte-ot betölti az akkuba, majd továbbítja az X regiszterbe. Az akku és az X regiszter tartalmától függően módosulnak az N és Z jelzők.

Regiszterek:

PS: N: az akku tartalmától függően módosul
 V: nem változik
 B: nem változik
 D: nem változik
 I: nem változik
 Z: az akku tartalmától függően módosul
 C: nem változik

N	V	B	D	I	Z	C
*	—	—	—	—	*	—

AC: átveszi a megcímzett byte tartalmát
 XR: átveszi az akku tartalmát
 YR: nem változik
 SP: nem változik
 PC: $PC = PC + 2$ (3)

Formátum:

	7	6	5	4	3	2	1	0	KÓD	SZINTAXIS	CIKLUSIDŐ
	1	0	1	címrés			1	1			
indexelt indirekt				0	0	0			\$A3	LDT (\$xx,X)	6 μ s
nulláslap				0	0	1			\$A7	LDT \$xx	3 μ s
abszolút				0	1	1			\$AF	LDT \$xxxx	4 μ s
indirekt, indexelt				1	0	0			\$B3	LDT (\$xx),Y	5 μ s (6)
nulláslap,Y				1	0	1			\$B7	LDT \$xx,Y	4 μ s
abszolút,Y				1	1	1			\$BF	LDT \$xxxx,Y	5 μ s

LDX operandus

LOAD XR — az X regiszterbe tölti a megcímezett byte tartalmát. Az N és Z jelzők ennek megfelelően módosulnak.

Regiszterek:

PS: N: az XR tartalmától függően módosul
 V: nem változik
 B: nem változik
 D: nem változik
 I: nem változik
 Z: az XR tartalmától függően módosul
 C: nem változik

N	V	B	D	I	Z	C
*	—	—	—	—	*	—

AC: nem változik

XR: átveszi a megcímezett byte tartalmát

YR: nem változik

SP: nem változik

PC: PC=PC+2 (3)

Formátum:

	7	6	5	4	3	2	1	0	KÓD	SZINTAXIS	CIKLUSIDŐ
	1	0	1	címrés			1	0			
közvetlen				0	0	0			\$A2	LDX #\$xx	2 μ s
nulláslap				0	0	1			\$A6	LDX \$xx	3 μ s
abszolút				0	1	1			\$AE	LDX \$xxxx	4 μ s
nulláslap,Y				1	0	1			\$B6	LDX \$xx,Y	4 μ s
abszolút,Y				1	1	1			\$BE	LDX \$xxxx,Y	4 μ s (5)

LDY operandus

LOAD YR — az Y regiszterbe tölti a megcímzett byte tartalmát. Az N és Z jelzők ennek alapján módosulnak.

Regiszterek:

PS: N: az YR tartalmától függően módosul
 V: nem változik
 B: nem változik
 D: nem változik
 I: nem változik
 Z: az YR tartalmától függően módosul
 C: nem változik

N	V	B	D	I	Z	C
*	—	—	—	—	*	—

AC: nem változik

XR: nem változik

YR: átveszi a megcímzett byte tartalmát

SP: nem változik

PC: $PC = PC + 2$ (3)

Formátum:

	7	6	5	4	3	2	1	0
	1	0	1	címzés	0	0		
közvetlen				0	0	0		
nulláslap				0	0	1		
abszolút				0	1	1		
nulláslap,X				1	0	1		
abszolút,X				1	1	1		

KÓD SZINTAXIS CIKLUSIDŐ

\$A0	LDY #\$xx	2 μ s
\$A4	LDY \$xx	3 μ s
\$AC	LDY \$xxxx	4 μ s
\$B4	LDY \$xx,X	4 μ s
\$BC	LDY \$xxxx,X	4 μ s (5)

LSE operandus*Tiltott kód*

LOGICAL SHIFT RIGHT AND EXCLUSIVE OR — a bitek jobbralejtetése, majd kizáró VAGY művelet. Először a megcímzett byte bitjeit tolja jobbra egy helyiértékkel, a 0. bit a Carry-be kerül. Ezután az eredmény és az akku tartalma között végrehajt egy kizáró VAGY műveletet. Az eredmény az akkuba kerül. Az N és Z jelzők az akku tartalmától függően módosulnak.

Regiszterek:

PS: N: magas, ha az eredmény 7. bitje 1
 V: nem változik
 B: nem változik
 D: nem változik

N	V	B	D	I	Z	C
*	—	—	—	—	*	*

I: nem változik
 Z: magas, ha az eredmény nulla
 C: átveszi a forrásbyte 0. bitjét

AC: az eredmény itt keletkezik

XR: nem változik

YR: nem változik

SP: nem változik

PC: $PC = PC + 2$ (3)

Formátum:

	7	6	5	4	3	2	1	0	KÓD	SZINTAXIS	CIKLUSIDŐ
	0	1	0	címkzés			1	1			
indexelt indirekt				0	0	0			\$43	LSE (\$xx,X)	8 μ s
nulláslap				0	0	1			\$47	LSE \$xx	5 μ s
abszolút				0	1	1			\$4F	LSE \$xxxx	6 μ s
indirekt, indexelt				1	0	0			\$53	LSE (\$xx),Y	8 μ s
nulláslap,X				1	0	1			\$57	LSE \$xx,X	6 μ s
abszolút,Y				1	1	0			\$5B	LSE \$xxxx,Y	7 μ s
abszolút,X				1	1	1			\$5F	LSE \$xxxx,X	7 μ s

LSR operandus

LOGICAL SHIFT RIGHT — a megcímezett byte-on (regiszteren) jobbra tolja a biteket. A legalsó (0.) bit a Carry-be kerül. Az utasítás matematikai értelme a 2-vel való osztás, a Carry a maradékot tartalmazza. Az N és Z jelzők az eredménytől függően módosulnak. A 7. bit mindig 0 lesz.

Regiszterek:

PS: N: az eredmény 7. bitjétől függően
 magas vagy alacsony
 V: nem változik
 B: nem változik
 D: nem változik
 I: nem változik
 Z: az eredménytől függően módosul
 C: a forrásregiszter 0. bitjét veszi át

N	V	B	D	I	Z	C
*	—	—	—	—	*	*

AC: csak beleértett címzés esetén módosul
 XR: nem változik
 YR: nem változik
 SP: nem változik
 PC: $PC = PC + 2$ (3)

Formátum:

	7	6	5	4	3	2	1	0
	0	1	0	címzés			1	0
nulláslap				0	0	1		
beleértett				0	1	0		
abszolút				0	1	1		
nulláslap,X				1	0	1		
abszolút,X				1	1	1		

KÓD SZINTAXIS CIKLUSIDŐ

\$46	LSR \$xx	5 μ s
\$4A	LSR A	2 μ s
\$4E	LSR \$xxxx	6 μ s
\$56	LSR \$xx,X	6 μ s
\$5E	LSR \$xxxx,X	7 μ s

NOP, NO1, NO2, NO3

NO OPERATION — nincs művelet.

Regiszterek:

PS: a jelzők egyike sem változik
 AC: nem változik
 XR: nem változik
 YR: nem változik
 SP: nem változik
 PC: $PC = PC + 1$ (2, 3)

N	V	B	D	I	Z	C
—	—	—	—	—	—	—

Formátum:

7	6	5	4	3	2	1	0
1	1	1	0	1	0	1	0
0	0	0	1	1	0	1	0
0	0	1					
0	1	0					
0	1	1					
1	1	0					
1	1	1					
1	0	0	0	0	0	0	0
1	0	0	0	1	0	0	1
1	0	0	0	0	0	1	0
1	1	0					
1	1	1					
0	0	0	0	0	1	0	0
0	0	0	1	0	1	0	0
0	0	1					
0	1	0	0	0	1	0	0
0	1	0	1				
0	1	1	0				
0	1	1	1				
1	1	0	1				
1	1	1	1	0	1	0	0
0	0	0	0	1	1	0	0
0	0	0	1				
0	0	1	1				
0	1	0	1				
0	1	1	1				
1	1	0	1				
1	1	1	1	1	1	0	0

KÓD	SZINTAXIS	CIKLUSIDŐ
\$EA	NOP	2 μ s
\$1A	NO1	2 μ s
\$3A	NO1	2 μ s
\$5A	NO1	2 μ s
\$7A	NO1	2 μ s
\$DA	NO1	2 μ s
\$FA	NO1	2 μ s
\$80	NO2 (\$xx,X)	2 μ s
\$89	NO2 #\$xx	2 μ s
\$82	NO2 (\$xx,X)	2 μ s
\$C2	NO2 (\$xx,X)	2 μ s
\$E2	NO2 (\$xx,X)	2 μ s
\$04	NO2 \$xx	3 μ s
\$14	NO2 \$xx,X	4 μ s
\$34	NO2 \$xx,X	4 μ s
\$44	NO2 \$xx	3 μ s
\$54	NO2 \$xx,X	4 μ s
\$64	NO2 \$xx	3 μ s
\$74	NO2 \$xx,X	4 μ s
\$D4	NO2 \$xx,X	4 μ s
\$F4	NO2 \$xx,X	4 μ s
\$0C	NO3 \$xxxx	4 μ s
\$1C	NO3 \$xxxx,X	4 μ s (5)
\$3C	NO3 \$xxxx,X	4 μ s (5)
\$5C	NO3 \$xxxx,X	4 μ s (5)
\$7C	NO3 \$xxxx,X	4 μ s (5)
\$DC	NO3 \$xxxx,X	4 μ s (5)
\$FC	NO3 \$xxxx,X	4 μ s (5)

ORA operandus

OR ACCU — logikai VAGY művelet az akku és a megcímzett byte között. Az eredmény az akkuba kerül, és értékének megfelelően módosulnak az N és Z jelzők.

Regiszterek:

PS: N: a művelet eredményétől
függően módosul
V: nem változik
B: nem változik

N	V	B	D	I	Z	C
*	—	—	—	—	*	—

D: nem változik
 I: nem változik
 Z: a művelet eredményétől függően módosul
 C: nem változik

AC: az eredmény itt képződik

XR: nem változik

YR: nem változik

SP: nem változik

PC: $PC = PC + 2$ (3)

Formátum:

	7	6	5	4	3	2	1	0
	0	0	0	címzés	0	1		
indexelt indirekt				0	0	0		
nulláslap				0	0	1		
közvetlen				0	1	0		
abszolút				0	1	1		
indirekt, indexelt				1	0	0		
nulláslap,X				1	0	1		
abszolút,Y				1	1	0		
abszolút,X				1	1	1		

KÓD SZINTAXIS CIKLUSIDŐ

\$01	ORA (\$xx,X)	6 μ s
\$05	ORA \$xx	3 μ s
\$09	ORA #\$xx	2 μ s
\$0D	ORA \$xxxx	4 μ s
\$11	ORA (\$xx),Y	5 μ s (6)
\$15	ORA \$xx,X	4 μ s
\$19	ORA \$xxxx,Y	4 μ s (5)
\$1D	ORA \$xxxx,X	4 μ s (5)

PHA

PUSH ACCU — az akku verembe mentése. A veremmutató eggyel csökken. A jelzők nem változnak.

Regiszterek:

PS: N: nem változik
 V: nem változik
 B: nem változik
 D: nem változik
 I: nem változik
 Z: nem változik
 C: nem változik

N	V	B	D	I	Z	C
—	—	—	—	—	—	—

AC: nem változik

XR: nem változik

YR: nem változik

SP: $SP = SP - 1$

PC: $PC = PC + 1$

Formátum:

7	6	5	4	3	2	1	0
<hr/>							
0	1	0	0	1	0	0	0

<i>KÓD</i>	<i>SZINTAXIS</i>	<i>CIKLUSIDŐ</i>
------------	------------------	------------------

\$48	PHA	3 μ s
------	-----	-----------

PHP

PUSH PROCESSOR STATUS — az állapotregiszter mentése a verembe. A veremmutató eggyel csökken. A jelzők nem változnak.

Regiszterek:

PS: N: nem változik
 V: nem változik
 B: nem változik
 D: nem változik
 I: nem változik
 Z: nem változik
 C: nem változik

N	V	B	D	I	Z	C
—	—	—	—	—	—	—

AC: nem változik
 XR: nem változik
 YR: nem változik
 SP: SP = SP - 1
 PC: PC = PC + 1

Formátum:

7	6	5	4	3	2	1	0
<hr/>							
0	0	0	0	1	0	0	0

<i>KÓD</i>	<i>SZINTAXIS</i>	<i>CIKLUSIDŐ</i>
------------	------------------	------------------

\$08	PHP	3 μ s
------	-----	-----------

PLA

PULL ACCU — a veremmutató által kijelölt byte betöltődik az akkuba. Az N és Z jelzők a byte értékének megfelelő állapotot vesznek fel. A veremmutató értéke eggyel növekszik.

Regiszterek:

PS: N: az akku tartalmának
 megfelelően módosul
 V: nem változik
 B: nem változik
 D: nem változik
 I: nem változik

N	V	B	D	I	Z	C
*	—	—	—	—	*	—

Z: az akku tartalmának megfelelő értéket vesz fel

C: nem változik

AC: átveszi a veremből kiemelt byte értékét

XR: nem változik

YR: nem változik

SP: $SP = SP + 1$

PC: $PC = PC + 1$

Formátum:

7 6 5 4 3 2 1 0

0 1 1 0 1 0 0 0

KÓD SZINTAXIS CIKLUSIDŐ

\$68

PLA

4 μ s

PLP

PULL PROCESSOR STATUS — az állapotregiszter visszaolvasása a veremből. A veremmutató értéke eggyel nő. Minden jelző változhat.

Regiszterek:

PS: N: változhat

V: változhat

B: változhat

D: változhat

I: változhat

Z: változhat

C: változhat

N	V	B	D	I	Z	C
*	*	*	*	*	*	*

AC: nem változik

XR: nem változik

YR: nem változik

SP: $SP = +1$

PC: $PC = PC + 1$

Formátum:

7 6 5 4 3 2 1 0

0 0 1 0 1 0 0 0

KÓD SZINTAXIS CIKLUSIDŐ

\$28

PLP

4 μ s

RAD operandus*Tiltott kód*

ROTATE RIGHT AND ADDITION WITH CARRY — ciklikus forgatás jobbra, összeadás. A megcímzett byte-on végrehajt egy ROR műveletet, majd az eredményt hozzáadja az akkuhoz. A Carry jelző ugyanazt a szerepet játssza, amit a ROR és az ADC utasításnál. Az N és Z jelzők az akku tartalma szerint módosulnak.

Regiszterek:

PS: N: magas, ha az akku 7. bitje 1.
 V: nem változik
 B: nem változik
 D: nem változik
 I: nem változik
 Z: magas, ha az akku nulla
 C: a forrásregiszter 0. bitjét veszi át

N	V	B	D	I	Z	C
*	—	—	—	—	*	*

AC: az eredmény itt képződik

XR: nem változik

YR: nem változik

SP: nem változik

PC: $PC = PC + 2$ (3)

Formátum:

7 6 5 4 3 2 1 0

0 1 1 címzés 1 1

indexelt indirekt

0 0 0

nulláslap

0 0 1

abszolút

0 1 1

indirekt, indexelt

1 0 0

nulláslap,X

1 0 1

abszolút,Y

1 1 0

abszolút,X

1 1 1

KÓD SZINTAXIS CIKLUSIDŐ

\$63 RAD (\$xx,X) 8 μ s

\$67 RAD \$xx 5 μ s

\$6F RAD \$xxxx 6 μ s

\$73 RAD (\$xx),Y 8 μ s

\$77 RAD \$xx,X 6 μ s

\$7B RAD \$xxxx,Y 7 μ s

\$7F RAD \$xxxx,X 7 μ s

RAN operandus*Tiltott kód*

ROTATE LEFT — AND WITH ACCU — ciklikus forgatás balra logikai ÉS művelettel kombinálva. A megcímzett byte-on végrehajt egy ROL utasítást, majd az eredmény és az akku között egy logikai ÉS műveletet. Az eredmény az akkuba kerül. Az N és Z jelzők az akku tartalmának megfelelő értéket vesznek fel. A Carry jelző a ROL utasításnál ismert szerepet tölti be.

Regiszterek:

PS: N: magas, ha az akku 7. bitje 1
 V: nem változik
 B: nem változik
 D: nem változik
 I: nem változik
 Z: magas, ha az akku zéró
 C: átveszi a forrásregiszter 0. bitjét

N	V	B	D	I	Z	C
*	—	—	—	—	*	*

AC: az eredmény itt képződik

XR: nem változik

YR: nem változik

SP: nem változik

PC: $PC = PC + 2$ (3)

Formátum:

	7	6	5	4	3	2	1	0
	0	0	1	címzés			1	1
indexelt indirekt				0	0	0		
nulláslap				0	0	1		
abszolút				0	1	1		
indirekt, indexelt				1	0	0		
nulláslap,X				1	0	1		
abszolút,Y				1	1	0		
abszolút,X				1	1	1		

KÓD SZINTAXIS CIKLUSIDŐ

\$23	RAN (\$xx,X)	8 μ s
\$27	RAN \$xx	5 μ s
\$2F	RAN \$xxxx	6 μ s
\$33	RAN (\$xx),Y	8 μ s
\$37	RAN \$xx,X	6 μ s
\$3B	RAN \$xxxx,Y	7 μ s
\$3F	RAN \$xxxx,X	7 μ s

ROL operandus

ROTATE LEFT — ciklikus forgatás balra. A megcímzett byte tartalmát a Carry bevonásával egy helyiértékkel balra lépteti. A 0. bit helyére a Carry tartalma, a 7. bit pedig a Carry-ba kerül. Az N és Z jelzők az eredménynek megfelelően állítódnak be.

Regiszterek:

PS: N: az eredménytől függően módosul
 V: nem változik
 B: nem változik
 D: nem változik
 I: nem változik
 Z: az eredménytől függően módosul
 C: átveszi a forrásregiszter 7. bitjét

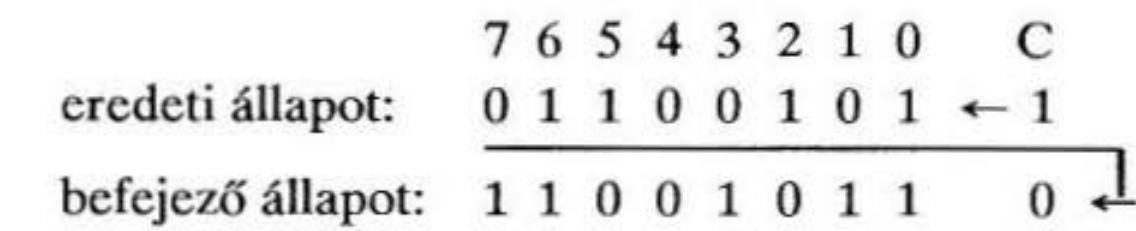
N	V	B	D	I	Z	C
*	—	—	—	—	*	*

- AC: csak beleértett címzés esetén változhat
- XR: nem változik
- YR: nem változik
- SP: nem változik
- PC: $PC = PC + 2$ (3)

Formátum:

	7	6	5	4	3	2	1	0	KÓD	SZINTAXIS	CIKLUSIDŐ
	0	0	1	címzés			1	0			
nulláslap				0	0	1			\$26	ROL \$xx	5µs
beleértett				0	1	0			\$2A	ROL A	2µs
abszolút				0	1	1			\$2E	ROL \$xxxx	6µs
nulláslap,X				1	0	1			\$36	ROL \$xx,X	6µs
abszolút,X				1	1	1			\$3E	ROL \$xxxx,X	7µs

A balra forgatás mechanizmusa:



ROR operandus

ROTATE RIGHT — ciklikus forgatás jobbra. A megcímzett byte tartalmát a Carry bevonásával egy helyiértékkel jobbra lépteti. A 7. bit helyére a Carry tartalma, a 0. bit pedig a Carry-be kerül. Az N és Z jelzők az eredménynek megfelelő állapotot vesznek fel.

Regiszterek:

- PS: N: az eredménynek megfelelően módosul
- V: nem változik
- B: nem változik
- D: nem változik
- I: nem változik
- Z: az eredménynek megfelelően módosul
- C: átveszi a forrásregiszter 0. bitjét

N	V	B	D	I	Z	C
*	—	—	—	—	*	*

AC: csak beleértett címzésnél változhat
 XR: nem változik
 YR: nem változik
 SP: nem változik
 PC: $PC = PC + 2$ (3)

Formátum:

	7	6	5	4	3	2	1	0	KÓD	SZINTAXIS	CIKLUSIDŐ
	0	1	1	címzés			1	0			
nulláslap				0	0	1			\$66	ROR \$xx	5 μ s
beleértett				0	1	0			\$6A	ROR A	2 μ s
abszolút				0	1	1			\$6E	ROR \$xxxx	6 μ s
nulláslap,X				1	0	1			\$76	ROR \$xx,X	6 μ s
abszolút,X				1	1	1			\$7E	ROR \$xxxx,X	7 μ s

A jobbra forgatás mechanizmusa:

	C	7	6	5	4	3	2	1	0
eredeti állapot:	0	→	0	1	1	0	0	1	1
	↓								
befejező állapot:	1		0	0	1	1	0	0	1

RTI

RETURN FROM INTERRUPT — visszatérés megszakításból. A megszakítóprogram utolsó utasítása. Hatására visszatöltődik az állapotregiszter és a programszámláló megszakítás előtti utolsó értéke, s a program tovább folytatódhat.

Regiszterek:

PS: N: változhat
 V: változhat
 B: változhat
 D: változhat
 I: változhat
 Z: változhat
 C: változhat

N	V	B	D	I	Z	C
*	*	*	*	*	*	*

AC: nem változik
 XR: nem változik
 YR: nem változik
 SP: $SP = SP + 3$
 PC: a regiszter felveszi a megszakítás előtti értékét

Formátum:

7	6	5	4	3	2	1	0
<hr/>							
0	1	0	0	0	0	0	0

KÓD	SZINTAXIS	CIKLUSIDŐ
-----	-----------	-----------

\$40	RTI	6 μ s
------	-----	-----------

RTS

RETURN FROM SUBROUTINE — visszatérés szubrutinból. Kiemeli a veremből és a programszámlálóba tölti a szubrutinhívás előtti utolsó értéket, amelyet a JSR mentett el. A veremmutató közben 2-vel növekszik. A visszatöltött programszámlálóhoz hozzáad egyet. A jelzők nem változnak.

Regiszterek:

PS: N: nem változik
 V: nem változik
 B: nem változik
 D: nem változik
 I: nem változik
 Z: nem változik
 C: nem változik

N	V	B	D	I	Z	C
—	—	—	—	—	—	—

AC: nem változik
 XR: nem változik
 YR: nem változik
 SP: $SP = SP + 2$
 PC: előbb a hívás előtti érték, majd $PC = PC + 1$

Formátum:

7	6	5	4	3	2	1	0
<hr/>							
0	1	1	0	0	0	0	0

KÓD	SZINTAXIS	CIKLUSIDŐ
-----	-----------	-----------

\$60	RTS	6 μ s
------	-----	-----------

SBC operandus

SUBTRACT WITH CARRY — a megcímezett byte-ot kivonja az akku tartalmából. Az eredmény az akkuba kerül. Az N és Z jelzők az eredménytől függően módosulnak. A Carry magas állapota jelzi, hogy nem volt túlszordulás.

Regiszterek:

PS: N: az eredménytől függően módosul
 V: magas, ha a művelet során bináris túlszordulás keletkezett
 B: nem változik

N	V	B	D	I	Z	C
*	*	—	—	—	*	*

D: nem változik
 I: nem változik
 Z: az eredménytől függően módosul
 C: magas, ha nem volt átvitel

AC: az eredmény itt képződik
 XR: nem változik
 YR: nem változik
 SP: nem változik
 PC: $PC = PC + 2$ (3)

Formátum:

	7	6	5	4	3	2	1	0	KÓD	SZINTAXIS	CIKLUSIDŐ
	1	1	1	címzés		0	1				
indexelt indirekt				0	0	0			\$E1	SBC (\$xx,X)	6 μ s
nulláslap				0	0	1			\$E5	SBC \$xx	3 μ s
közvetlen				0	1	0			\$E9	SBC #\$xx	2 μ s
abszolút				0	1	1			\$ED	SBC \$xxxx	4 μ s
indirekt, indexelt				1	0	0			\$F1	SBC (\$xx),Y	5 μ s (6)
nulláslap,X				1	0	1			\$F5	SBC \$xx,X	4 μ s
abszolút,Y				1	1	0			\$F9	SBC \$xxxx,Y	4 μ s (5)
abszolút,X				1	1	1			\$FD	SBC \$xxxx,X	4 μ s (5)

A kivonás összeadásra vezethető vissza. A megcímzett byte tartalmát az ALU invertálja és hozzáadja az akkuhoz, majd a szintén invertált Carry-vel teszi ugyanezt. Az összeadás 9. bitje kerül a Carry-be.

SBC byte

Tiltott kód

SUBTRACT WITH CARRY — bináris kivonás. A standard MOS-assembler nem alkalmazta, ezért a tiltott kódok közé soroltuk. Teljes mértékben megegyezik a standard SBC közvetlen címzésű változatával.

Regiszterek

PS: N: a művelet eredményétől
 függően módosul
 V: ha volt túlsordulás, magas
 B: nem változik
 D: nem változik
 I: nem változik
 Z: a művelet eredményétől függően módosul
 C: magas, ha nem volt átvitel

N	V	B	D	I	Z	C
*	*	—	—	—	*	*

AC: az eredmény ide kerül
 XR: nem változik
 YR: nem változik
 SP: nem változik
 PC: $PC = PC + 2$

Formátum:

7 6 5 4 3 2 1 0
 1 1 1 0 1 0 1 1

KÓD SZINTAXIS CIKLUSIDŐ
 \$EB SBC #\$xx 2 μ s

SE_x

SET x — a megfelelő jelző beállítása.

Regiszterek:

PS: N: nem változik
 V: nem változik
 B: nem változik
 D: változhat
 I: változhat
 Z: nem változik
 C: változhat

	N	V	B	D	I	Z	C
SEC	—	—	—	—	—	—	1
SED	—	—	—	1	—	—	—
SEI	—	—	—	—	1	—	—

AC: nem változik
 XR: nem változik
 YR: nem változik
 SP: nem változik
 PC: $PC = PC + 1$

Formátum:

7 6 5 4 3 2 1 0
 0 0 1 1 1 0 0 0
 1 1 1
 0 1 1

KÓD SZINTAXIS CIKLUSIDŐ
 \$38 SEC 2 μ s
 \$F8 SED 2 μ s
 \$78 SEI 2 μ s

SEC: beállítja az átvitelbitet
 SED: beállítja a decimális üzemmódot
 SEI: beállítja az Interrupt bitet, tiltja a maszkolható megszakításokat.

STA operandus

STORE ACCU — az akkumulátor tartalmának tárolása a megadott címen. A jelzők nem változnak.

Regiszterek:

PS: N: nem változik
 V: nem változik
 B: nem változik
 D: nem változik
 I: nem változik
 Z: nem változik
 C: nem változik

AC: nem változik
 XR: nem változik
 YR: nem változik
 SP: nem változik
 PC: $PC = PC + 2$ (3)

N	V	B	D	I	Z	C
—	—	—	—	—	—	—

Formátum:

	7	6	5	4	3	2	1	0
	1	0	0	címzés	0	1		
indexelt indirekt				0	0	0		
nulláslap				0	0	1		
abszolút				0	1	1		
indirekt, indexelt				1	0	0		
nulláslap,X				1	0	1		
abszolút,Y				1	1	0		
abszolút,X				1	1	1		

KÓD SZINTAXIS CIKLUSIDŐ

\$81	STA (\$xx,X)	6 μ s
\$85	STA \$xx	3 μ s
\$8D	STA \$xxxx	4 μ s
\$91	STA (\$xx),Y	6 μ s
\$95	STA \$xx,X	4 μ s
\$99	STA \$xxxx,Y	5 μ s
\$9D	STA \$xxxx,X	5 μ s

STX operandus

STORE XR — az X regiszter tartalmának tárolása a megadott címen. A jelzők nem változnak.

Regiszterek:

PS: N: nem változik
 V: nem változik
 B: nem változik
 D: nem változik
 I: nem változik

N	V	B	D	I	Z	C
—	—	—	—	—	—	—

Z: nem változik
 Z: nem változik
 C: nem változik

AC: nem változik
 XR: nem változik
 YR: nem változik
 SP: nem változik
 PC: $PC = PC + 2$ (3)

Formátum:

7 6 5 4 3 2 1 0
 1 0 0 címzés 1 0

nulláslap
 abszolút
 nulláslap, Y

0 0 1
 0 1 1
 1 0 1

KÓD SZINTAXIS CIKLUSIDŐ

\$86 STX \$xx 3 μ s
 \$8E STX \$xxxx 4 μ s
 \$96 STX \$xx, Y 4 μ s

STY operandus

STORE YR — az Y regiszter tárolása a megadott címen. A jelzők nem változnak.

Regiszterek:

PS: N: nem változik
 V: nem változik
 B: nem változik
 D: nem változik
 I: nem változik
 Z: nem változik
 C: nem változik

N	V	B	D	I	Z	C
—	—	—	—	—	—	—

AC: nem változik
 XR: nem változik
 YR: nem változik
 SP: nem változik
 PC: $PC = PC + 2$ (3)

Formátum:

7 6 5 4 3 2 1 0
 1 0 0 címzés 0 0

nulláslap
 abszolút
 nulláslap, X

0 0 1
 0 1 1
 1 0 1

KÓD SZINTAXIS CIKLUSIDŐ

\$84 STY \$xx 3 μ s
 \$8C STY \$xxxx 4 μ s
 \$94 STY \$xx, X 4 μ s

TAN byte*Tiltott kód*

Logikai ÉS művelet az X regiszter és a byte között. Az X regiszter tartalma először átkerül az akkuba, majd végrehajtódik az AND. A jelzők nem változnak.

Regiszterek:

PS: N: nem változik
 V: nem változik
 B: nem változik
 D: nem változik
 I: nem változik
 Z: nem változik
 C: nem változik

N	V	B	D	I	Z	C
—	—	—	—	—	—	—

AC: az eredmény itt képződik
 XR: nem változik
 YR: nem változik
 SP: nem változik
 PC: $PC = PC + 2$

Formátum:

7 6 5 4 3 2 1 0
 1 0 0 0 1 0 1 1

KÓD SZINTAXIS CIKLUSIDŐ
 \$8B TAN # \$xx 2 μ s

TAX, TAY, TXA, TYA

Regisztertranszfer műveletek.

Regiszterek:

PS: N: magas, ha a 7. bit magas
 V: nem változik
 B: nem változik
 D: nem változik
 I: nem változik
 Z: magas, ha minden bit nulla
 C: nem változik

	N	V	B	D	I	Z	C
TAX	*	—	—	—	—	*	—
TAY	*	—	—	—	—	*	—
TXA	*	—	—	—	—	*	—
TYA	*	—	—	—	—	*	—

AC: TAXA és TYA esetén felveszi a regiszter értékét, egyébként nem változik
 XR: TAX esetén felveszi az akku értékét, egyébként nem változik
 YR: TAY esetén felveszi az akku értékét, egyébként nem változik
 SP: nem változik
 PC: $PC = PC + 1$

Formátum:

7 6 5 4 3 2 1 0	KÓD	SZINTAXIS	CIKLUSIDŐ
1 0 1 0 1 0 1 0	\$AA	TAX	2 μ s
1 0 1 0 1 0 0 0	\$A8	TAY	2 μ s
1 0 0 0 1 0 1 0	\$8A	TXA	2 μ s
1 0 0 1 1 0 0 0	\$98	TYA	2 μ s

TAX: az akku tartalma az X regiszterbe kerül.

TAY: az akku tartalma az Y regiszterbe kerül.

TXA: az akku átveszi az X regiszter tartalmát.

TYA: az akku átveszi az Y regiszter tartalmát.

Az N és Z jelzők az eredménytől függően változnak.

TSA operandus*Tiltott kód*

A veremmutató áthelyezése az akkuba. A megcímezett byte tartalma és a veremmutató közötti logikai ÉS művelet eredménye az X regiszterbe kerül, majd onnan az akkuba és a veremmutatóba. A jelzők nem változnak.

Regiszterek:

PS: N: nem változik
 V: nem változik
 B: nem változik
 D: nem változik
 I: nem változik
 Z: nem változik
 C: nem változik

N	V	B	D	I	Z	C
—	—	—	—	—	—	—

AC: az eredmény ide kerül az X regiszterből

XR: a művelet eredménye először ide kerül

YR: nem változik

SP: átveszi az X regiszter tartalmát

PC: PC=PC+3

Formátum:

7 6 5 4 3 2 1 0	KÓD	SZINTAXIS	CIKLUSIDŐ
1 0 1 1 1 0 1 1	\$BB	TSA \$xxxx,Y	5 μ s

TSX és TXS

A veremmutató transzfer műveletei.

Regiszterek:

PS: N: csak TSX esetén változhat
 V: nem változik
 B: nem változik
 D: nem változik
 I: nem változik
 Z: csak TSX esetén változhat
 C: nem változik

	N	V	B	D	I	Z	C
TSX	*	—	—	—	—	*	—
TXS	—	—	—	—	—	—	—

AC: nem változik

XR: TSX: átveszi a veremmutató tartalmát, egyébként nem változik

YR: nem változik

SP: TXS: átveszi az X regiszter tartalmát, egyébként nem változik

PC: PC=PC+1

Formátum:

7 6 5 4 3 2 1 0
 1 0 1 1 1 0 1 0
 1 0 0

KÓD SZINTAXIS CIKLUSIDŐ

\$BA TSX 2μs
 \$9A TXS 2μs

TSX: a veremmutató az X regiszterbe kerül. Az N és Z jelzők a veremmutató tartalma szerint változnak.

TXS: az X regiszter tartalma a veremmutatóba kerül. A jelzők nem változnak, mert a veremmutató nem munkaregisztere a processzornak.

XAS byte

Tiltott kód

Az X regiszter és az akku között logikai ÉS, ennek eredményéből kivonja a megadott byte tartalmát, de úgy, hogy a Carry-t nem veszi figyelembe. Az eredmény az X regiszterbe, a Carry magas állapotba kerül. Az N és Z jelzők az eredménynek megfelelően módosulnak.

Regiszterek:

PS: N: az eredménytől függően módosul
 V: nem változik
 B: nem változik
 D: nem változik
 I: nem változik

N	V	B	D	I	Z	C
*	—	—	—	—	*	1

Z: az eredménytől függően módosul
C: magas értéket vesz fel

AC: nem változik
XR: az eredmény ide kerül
YR: nem változik
SP: nem változik
PC: $PC = PC + 2$

Formátum:

7	6	5	4	3	2	1	0
1	1	0	0	1	0	1	1

KÓD	SZINTAXIS	CIKLUSIDŐ
\$CB	XAS #\$xx	2 μ s

XYA operandus

Tiltott kód

Logikai ÉS művelet az X regiszter és a byte között; ennek eredménye kerül a megcímzett byte-ba. A jelzők nem változnak.

Regiszterek:

PS: N: nem változik
V: nem változik
B: nem változik
D: nem változik
I: nem változik
Z: nem változik
C: nem változik

N	V	B	D	I	Z	C
—	—	—	—	—	—	—

AC: nem változik
XR: nem változik
YR: nem változik
SP: nem változik
PC: $PC = PC + 3$

Formátum:

7	6	5	4	3	2	1	0
1	0	0	1	1	1	1	0

KÓD	SZINTAXIS	CIKLUSIDŐ
\$9E	XYA \$xxxx, Y	5 μ s

YXA operandus*Tiltott kód*

Logikai ÉS művelet az Y regiszter és a cím felső byte-ja között; ennek eredménye kerül a megcímzett byte-ba. A jelzők nem változnak.

Regiszterek:

PS: N: nem változik
 V: nem változik
 B: nem változik
 D: nem változik
 I: nem változik
 Z: nem változik
 C: nem változik

AC: nem változik
 XR: nem változik
 YR: nem változik
 SP: nem változik
 PC: PC=PC+3

N	V	B	D	I	Z	C
—	—	—	—	—	—	—

Formátum:

7 6 5 4 3 2 1 0
 1 0 0 1 1 1 0 0

KÓD SZINTAXIS CIKLUSIDŐ
 \$9C YXA \$xxxx,Y 5μs

2.3. Gépi kódú programozás, assembly programozás

A processzor gépi nyelvű programozásához elengedhetetlenül fontos ismerni az assemblereket, az assembly szintű programozást. A C64-es bőséges szoftverkínálata az assemblerekre is érvényes. Az assemblerek különböző fajtáinak ismertetése magában is megtöltene egy könyvet, ezért itt a legegyszerűbb és legelterjedtebb assemblert ismertetjük, a HELP PLUS BASIC-bővítés assemblerét. A könyvben szereplő assembler programok is mind ezzel a programmal dolgozhatók fel.

Az assembler lényegében egy fordítóprogram, amely az assembly forrásnyelvi programot gépi kóddá fordítja. A feldolgozás két lépésben történik: az első lépésben az assembler végigolvassa a forrásnyelvi programot, ellenőrzi a szintaxist, megállapítja a változók helyét és tárolja azokat. A második lépésben történik az igazi fordítás, az assembler lefordítja az utasí-

tásokat gépi kódra és tárolja azokat. A tárolás a **HELP PLUS** esetében a memóriában történik.

Az assembly programozáshoz nemcsak a gépi nyelv építőelemeit szükséges ismerni. Az assemblerek kezeléséhez tisztában kell lenni az általuk nyújtott szolgáltatásokkal is, amelyek programonként eltérőek lehetnek, de néhány alapfeltételben megegyeznek.

A program kezdőcímét a * operátorral szokás megadni. Az operátorról később még lesz szó. Az assemblerek többsége egyformán tudja kezelni a bináris, decimális és hexadecimális értékeket. Az assembler számára címeket és adatokat helyettesíthet valamilyen címke is. A címke hossza általában 6 karakter lehet, tartalmazhat számokat és betűket, de nem kezdődhet számjeggyel. A címke nem lehet mnemonik, mert a fordító ezt utasításnak fogja vélni. A fordítás menetét szabályozó direktívák a mnemonikok helyére kerülnek, de hogy mind a mnemonikoktól, mind a címkéktől meg lehessen őket különböztetni, ponttal kezdődnek.

A **HELP PLUS** a következő direktívákat ismeri fel:

- BYTE:** egybyte-os operandusok következhetnek utána. Elválasztásuk vesszővel történik. A tárgyprogramba a byte-ok értéke épül be.
- WORD:** címpoperandusok következhetnek utána. A tárgyprogramba alsó, felső byte formájában épülnek be.
- TEXT:** a direktívát követő, idézőjelek közé zárt szöveg ASCII kódjai kerülnek a tárgyprogramba.
- DISP:** mint az előző, de nem az ASCII kódok, hanem a képernyőkódok kerülnek a programba.
- LOAD:** a megadott file fordításával folytatódik az assemblálás.
- END:** utasítja az assemblert a fordítás befejezésére.

Más assemblerek természetesen tartalmazhatnak más direktívákat is.

Az assembly programozás néhány vonatkozásban hasonlít a BASIC programozásra. E hasonlóság különösen a fejlesztés időszakában mutatkozik meg: a programot BASIC sorként helyezhetjük el a tárban, és mint BASIC programot mentjük lemezre. Az assembler a lemezre mentett forrásnyelvi programot dolgozza fel.

Az assembly programozás előnye a BASIC vagy más magasszintű programozással szemben főként abban mutatkozik meg, hogy a gépi nyelvű programozás optimálisan kihasználja a processzort, annak teljesítményét. Ennek legfontosabb ismérve a sebesség. De van bőven olyan programozási feladat is, amely vagy nem valósítható meg BASIC-ben, vagy olyan lassú, hogy nem tekinthető megoldásnak. Ha assemblyben tanulunk programozni, számos nehézség elé kell néznünk, amelyeket elsősorban a processzor működésének alapos ismerete segít leküzdeni. A nehézségeket bizonyára mindenkinek más és más jelenti. De ami a legjobban feladja a leckét, az az, hogy egyszerre kell gyorsan bánni a decimális, bináris és hexadecimális számrendszerekkel, megjegyezni az utasítások működését, és közben nem veszteni szem elől az összefüggéseket. Az is kemény dió, hogy a processzor nem ad ki hibaüzeneteket, emiatt az eleinte igen gyakori hibák okaira csak áttételesen lehet következtetni, ha egyáltalán lehet. Mert az első időkben bizony a processzor igen gyakran „kiakad”.

Jól megtanulni a processzor nyelvét csak gyakorlás útján lehet. Legjobb, bár igen fáradságos módszer a tanulásra, ha már működő programokat tanulmányozunk alaposan, esetleg itt-ott módosítani próbáljuk azokat. Tanulmányozásra maga a ROM a legalkalmasabb. Alapos tanulás ellenére is előfordulhat, hogy programunk fordítás után nem működik. Már az is hatalmas eredmény az első időkben, ha egyáltalán lefut, bár nem azt csinálja, amire kitaláltuk. Ennél sokkal gyakoribb a már említett kiakadás, amit követhet a hosszadalmas hibakeresés. Mások programjában a hibát rendszerint hamarabb észrevesszük, de a program a miénk, így a hiba is, felfedezése tehát jóval nehezebb. Különösen a logikai hibák deríthetők fel nehezen! Sokszor a hiba megkeresése több időbe kerül, mint maga a program kitalálása, begépelése és kijavítása.

Sok időt megtakaríthatunk, ha betartunk néhány fontos szabályt:

- szinte minden programozási feladat részekre bontható. E részek lehetőleg minél kisebbek legyenek. Ezeket a részeket írjuk meg és teszteljük külön-külön. Egy kisebb és egy nagyobb programozási feladat programozási ideje között exponenciális kapcsolat van. A program növekedtével a ráfordított idő hatványozottan növekszik;
- lehetőleg alakítsunk ki egységes jelölésrendszert. Ez többet segít, mint gondolnánk;
- a program szerkezete legyen jól áttekinthető és elkülöníthető.

Az ismertetett alapelvek a moduláris programozás alapelvei. A moduláris programozás alkalmazása sok-sok bosszúságtól kíméli meg a programozót. Ha moduljainkat hibátlanul dolgoztuk ki, nagy esélyünk van rá, hogy a modulok összeillesztése nem okoz majd problémákat.

2.3.1. A programok ellenőrzése, a MOVMON monitorprogram

A processzor eredményes programozásához a forrásnyelvi programokat gépi kódra fordító assembler nem elegendő. Ellenőriznünk kell a lefordított program helyességét, de ez pusztán futtatással sokszor nem lehetséges. Gyakran van szükség arra, hogy láthassuk, mit művel a tárban a program. Az is fontos, hogy gyorsan vissza tudjunk keresni a memóriában egyes programrészleteket. Erre való a *monitorprogram*, amely disassemblálja (visszafejti) a gépi kódot. A példaként vett HELP PLUS programnak is van disassemblere, de az nem üti meg egy monitorprogram színvonalát, ezért a monitorok közötti általánosan elfogadott szabványnak megfelelő MOVMON-t ismertetjük.

A MOVMON meglehetősen kényelmes disassembler. A kurzorral a listában nemcsak előre, hanem hátra is mozoghatunk. A decimális számokat átszámítja hexadecimálissá és viszont. Assembly programutasításokat is képes lefordítani. Két tesztelő üzemmódja is van; továbbá kereső és törlő funkciója is.

A MOVMON nem rendelkezik fix kezdőcímmel. Betöltés közben tudakolja, hogy hová helyezze a monitorprogramot.

A MOVMON utasításkészlete:

A	Assembler	A 6000 LDA #\$01	Assemblálja az utasítást.
B	Breakpoint Set	B 6002 02	Megállítja a programot, ha futás közben már kétszer „áthaladt” itt a programszámláló.
C	Compare	C 6000 6008 5000	Összehasonlítja a 6000 — 6008 közötti tartományt az 5000 — 5008 közötti tartománnyal. Az eltérések címeit a program kijelzi.
D	Disassembler	D 8000 (8060)	A megadott határok között disassemblálja a tár tartalmát.
F	Fill	F C000 CFFF EA	A C000-CFFF tartományt feltölti \$EA kóddal.
G	GO	G FCE2	Az FCE2 címen kezdődő rutinra adja a vezérlést.
H	Hunt	H 6000 6FFF EA	A megadott tartományban megkeresi a \$EA kódokat, és a címeket kiírja.
L	Load	L „XYZ” 08	Betölti az XYZ programot a helyére.
M	Memory	M 8000 (8060)	A megadott határok között kijelzi a memória tartalmát hexa számokban.
Q	Quick Trace	Q 8000	Gyors nyomkövetés. Ebben az üzemmódban érvényesül a B utasítás.
R	Registers	R	Kijelzi a processzor regisztereinek tartalmát.
S	Save	S „XYZ”,08,8000,8060	Lemezre menti XYZ néven a megadott tartomány tartalmát.
T	Transfer	T 6000 6020 6040	Áthelyezés. A 6000 — 6020 tartomány tartalmát áthelyezi a 6040-től kezdődően.
W	Walk	W 8000	Lassú nyomkövetés. Lépésenként hajtja végre a \$8000-től kezdődő programot. A léptetés egy-egy billentyűre történik. Kilépni STOP-pal lehet.
X	Exit	X	Kilépés a monitorprogramból.
#	#32768		Megadja a szám hexadecimális formáját.
\$	\$8000		Megadja a szám decimális formáját.

Más monitorprogramokban ennél kevesebb vagy több utasítás is lehet. A monitorprogramok akkor hasznosak, ha a már elkészült programot kell „belőni”. A jól sikerült programokon is szokott lenni igazítanivaló.

2.3.2. Gyakorlati programozás

Ciklusszervezés

A gépi nyelvben éppúgy fontosak a *ciklusok*, mint a BASIC-ben, a ciklusszervezés azonban különböző. Assemblerben ciklusváltozónak az X vagy az Y indexregisztert szokás használni. A ciklus lehet előre- vagy hátraszámláló. A hátraszámláló ciklusokat akkor használjuk, amikor a lépések száma kisebb, mint 256. A ciklus három részre tagolható:

- a ciklusfej, ahol a kezdőértéket adjuk meg,
- a ciklusmag, ahol a tényleges műveletek történnek,
- a ciklusvég, ahol a visszatérési feltételt ellenőrzi a processzor.

Vizsgáljunk meg egy-egy példát előre- és hátraszámláló ciklusokra.

Előreszámláló:

```
LDX    #$00    ciklusfej
AA0    TXA
STA    $0400,X  ciklusmag
INX
BNE    AA0      ciklusvég
```

A ciklus egészen addig fut, amíg az X regiszter újra nulla nem lesz. Ha meghatározott lépésszámmig akarjuk a ciklust futtatni (pl. 120-ig), akkor a ciklusvég némileg módosul:

```
INX
CPX    #$78
BNE    AA0
```

A ciklusmag természetesen lehet egészen összetett is, de arra nagyon vigyázzunk, hogy a ciklusmagban ne alkalmazzunk olyan utasítást, amely az X regiszternek értéket ad. Bonyolult programok bonyolult ciklusokat is megkívánhatnak. Ilyenkor előfordulhat, hogy a ciklusváltozó regiszterét mégis használnunk kell a ciklusmagban. A káosz ekkor is elkerülhető: a ciklusváltozó pillanatnyi értékét célszerűen megválasztott segédregiszterbe kell helyezni és csak a ciklusvégből visszaolvasni.

Hátraszámláló:

```
LDY    #$3F    ciklusfej
LDA    #$FF
AA1    CLC      ciklusmag
ADC    #$01
STA    $1000,Y
```

```

EOR   #$FF
STA   $1040,Y
DEY                               ciklusvég
BPL   AA1

```

A példákból látható, hogy a két ciklusfajta nem sokban tér el egymástól, csupán működésük iránya más.

Összehasonlítás

Az összehasonlítást a processzor aritmetikai művelettel (közelebbről kivonással) végzi, mégpedig valamelyik munkaregisztere (AC, XR, YR) és a memória egy byte-ja között. Az összehasonlítás során a munkaregiszterből kivonja a megcímzett byte tartalmát, de sem a regiszter, sem a memória tartalma nem változik. A kivonás eredménye nem tárolt, viszont az állapotregiszter egyes bitjei az eredménynek megfelelő értéket vesznek fel. E bitek alapján dönthetjük el, hogy az összehasonlítás eredménye mi volt.

Kisebb:	C = 0 BCC
Nagyobb vagy egyenlő:	C = 1 BCS
Egyenlő:	Z = 1 BEQ
Nem egyenlő:	Z = 0 BNE
Negatív:	N = 1 BMI
Pozitív:	N = 0 BPL

Ha kizárólag a *nagyobb* feltételnek megfelelő elágazást kívánunk létrehozni, a BCS utasítás előtt az egyenlőséget vizsgálni kell, például:

```

CMP   #$23
BEQ   AA1      egyenlő
BCS   AA3      nagyobb

```

Ha egyenlő, az AA1-en folytatódik a program, ha nagyobb, az AA3-on, ha kisebb, nincs ugrás.

Aritmetikai műveletek

Bináris összeadás — ADC:

a megadott tárcím tartalma hozzáadódik az akkumulátor tartalmához. Az eredmény az akkumulátorban jelenik meg.

A bináris összeadás elve:

$$\begin{aligned} 0 + 0 &= 0 \\ 0 + 1 &= 1 \\ 1 + 0 &= 1 \\ 1 + 1 &= 0 + \text{átvitel} \end{aligned}$$

Két nyolcbites szám összeadásakor előfordulhat, hogy az eredmény nem ábrázolható 8 biten. Ilyenkor átvitel keletkezik, amely az állapotregiszter (Carry, C) átvitelbitjén jelentkezik. Az összeadás előtt az állapotregiszter átvitelbitje hozzáadódik az akku tartalmához, ezért az első művelet előtt ezt törölni kell (CLC).

Példák bináris összeadásra:

%0110 0010	C = 0	\$62	#98
+ %1001 0010		\$92	#146
<hr/>			
%1111 0100	C = 0	\$F4	#244
%1000 1111	C = 0	\$8F	#143
+ %1111 0000		\$F0	#240
<hr/>			
%0111 1111	C = 1	\$17F	#383
%0110 0010	C = 1	\$62	#98
+ %1001 0010		\$92	#146
<hr/>			
%1111 0101	C = 0	\$F5	#245
%1000 1111	C = 1	\$8F	#143
+ %1111 0000		\$F0	#240
<hr/>			
%1000 0000	C = 1	\$180	#384

Az átvitel (túlcsordulás) fogalma még érthetőbbé válik, ha az összeadást nem egy byte-on, hanem 32 bites bináris egészen mutatjuk be. Egy jellemző példa:

LDY	#\$00	
LDA	\$1000,Y	
CLC		<i>az első összeadás előtt az átvitelbitet törölni kell</i>
ADC	\$1004,Y	
STA	\$1000,Y	
INY		
LDA	\$1000,Y	
ADC	\$1004,Y	
STA	\$1000,Y	
INY		

```
LDA    $1000,Y
ADC    $1004,Y
STA    $1000,Y
INY
LDA    $1000,Y
ADC    $1004,Y
STA    $1000,Y
```

A program a \$1000 — 1003 címeken elhelyezkedő értékekhez hozzáadja a \$1004-1007 cí-
meken található értékeket. Ciklusba szervezve jóval rövidebb.

```
LDY    #$00
CLC
AA2:   LDA    $1000,Y
      ADC    $1004,Y
      STA    $1000,Y
      INY
      CPY    #$04
      BNE    AA2
```

Az átvitelbitet csak az első összeadás előtt kell törölni, mert az összeadás egyébként hely-
telen eredményre vezetne.

Kivonás — SBC:

az akkumulátor és a megcímzett byte különbsége az akkumulátorba kerül.

A bináris kivonás szabályai:

- 0 – 0 = 0
- 1 – 0 = 1
- 1 – 1 = 0
- 0 – 1 = 1 + átvitel

A bináris kivonás összeadásra vezethető vissza, s valójában a processzor is így végzi. A
kivonandó szám kettes komplementjét és a C jelző inverzét hozzáadja az akku tartalmához.
A kettes komplement képzésére a legegyszerűbb módszer, ha minden bitet invertálunk (el-
lenkezőjére fordítunk), majd hozzáadunk egyet. A kettes komplement fogalmát és előállítá-
sát a függelék részletesen tárgyalja.

Példa a kivonásra:

%1111 0011	C = 1	\$F3	#243
— %01001 1000		\$58	#88

a kettes komplementum:

$$\%0101\ 1000\ \%1010\ 0111 + 1 = \%1010\ 1000$$

a helyettesítő összeadás:

$\%1111\ 0011$	$C = 0$	$\$F3$	$\#243$
$+ \%1010\ 1000$		$\$A8$	$\#168$
<hr/>			
$\%1001\ 1011$	$C = 1$	$\$9B$	$\#155$

Az átvitelt itt a C jelző alacsony állapota mutatja, mert a processzor kivonás közben a C jelző inverzét használja. Emiatt kivonás előtt magasra kell állítani, mert a kivonás másként nem ad helyes eredményt. A kivonás az állapotregiszter N, Z, V és C jelzőit befolyásolja.

32 bites számok kivonásakor biztosítani kell, hogy két kivonási művelet között az átvitelbit értéke ne változhasson meg. Ez általában nem okoz problémát. Az első kivonás előtt az átvitelbitet magasra kell állítani. Ez a SBC utasítás működési feltétele. A programozási megoldás hasonlít ahhoz, amit az összeadásnál mutattunk be:

```

LDY    #$00
SEC
AA3    LDA    $1000,Y
      SBC    $1004,Y
      STA    $1000,Y
      INT
      CPY    #$04
      BNE    AA3

```

Szorzás kettővel — egy byte bitjeinek balratolása: ASL és ROL.

Az ASL 8, a ROL 9 bittel dolgozik. Mindkét utasításra igaz, hogy a megcímzett byte tartalmát egy bittel jobbra lépteti, ezáltal a byte 7. bitje a Carry jelzőbe kerül. Az ASL a 0. bitet alacsonyra állítja, a ROL a Carry jelző művelet előtti tartalmát tölti át. Példák az ASL működésére:

$\%0010\ 1011$	$\#43$	$C = 1,$	művelet után: $\%0101\ 0110$	$\#86$	$C = 0$
$\%1001\ 1110$	$\#158$	$C = 0,$	művelet után: $\%0011\ 1100$	$\#60$	$C = 1\ (\#316)$

Az ASL az N, Z és C jelzőket befolyásolja.

A ROL utasítás csak annyiban különbözik a fentiektől, hogy a 0. bit a Carry jelző művelet előtti tartalmát veszi át, ezért több byte-ból álló számok szorzásánál hasznos. Példa négy-byte-os, 32 bites egész kettővel való szorzására:

```

ASL    $1000
ROL    $1001
ROL    $1002
ROL    $1003

```


Osztás kettővel — egy byte bitjeinek jobbratolása: LSR és ROR.

Az LSR után a 7. bit mindig nulla lesz, a ROR után felveszi a művelet előtti Carry állapotát. Az LSR tehát 8, a ROR 9 bittel dolgozik. Mindkét utasításra igaz, hogy a 0. bit a Carry jelzőbe kerül.

Példák az LSR működésére:

%0110 0111	#103	C = 0,	művelet után: %0011 0011	#51	C = 1
%0111 1000	#120	C = 1,	művelet után: %0011 1100	#60	C = 0

Mindkét utasítás az N, Z és C jelzőket befolyásolja.

A ROL segítségével több byte-ból álló számokat forgathatunk (oszthatunk) jobbra. Példa kettővel való osztásra 4 byte-on:

```

LSR    $1003
ROR    $1002
ROR    $1001
ROR    $1000

```

A műveletet a legfelső byte-on kell kezdeni, ezért a szorzáshoz képest ellenkező irányú a feldolgozási sorrend.

Szubrutinok

A gépi nyelvű programozásban a szubrutin nagyon fontos építőelem. Segítségével az összetettebb programok áttekinthetővé és optimálisan rövidde tehetők. Ha egy jól körülhatárolható feladatot többször is el kell végeznie a programnak, azt a feladatot célszerű szubrutinnal megoldani. A főprogramban ezáltal a feladat egy utasításra, a szubrutinra való hivatkozásra zsugorodott.

Gyakorlatilag bármely programra igaz, hogy strukturált, hiszen még a szervezettség hiánya is valamilyen szervezési elvnek tekinthető. A strukturált jelzőt azonban mégis azok a programok kapják, amelyekben a rendszer könnyen felismerhető. Az egyik ilyen rendszer a moduláris programozás. E programozási módszer azért is ajánlott gépi nyelvű programozás esetén, mert az assembly programok átalakítása meglehetősen időigényes. A modulok cseréje viszont egyszerűbb, mint átírni a programot. A moduláris programozás fő jellemzője, hogy a főprogram szinte csak szubrutinhívásokból áll. Azok a programrészek is szubrutinként szerepelhetnek, amelyeket valójában csak egyszer kell végrehajtani. E látszólag haszontalan dolognak mégis van értelme: a főprogram nagyobb léptékű lépéseinek halmaza gyorsan áttekinthető; illetve módosítás esetén a megváltozott részek nincsenek kihatással a főprogram többi részére.

További jelentős előny, hogy a modulok sorrendje gyorsan megváltoztatható. Végeredményben a magasabb szintű programnyelvek is mind-mind moduláris rendszerű gépi nyelvű programok, amelyekben a modulokra az utasítások nevével hivatkozhatunk.

A processzor szubrutinhívó JSR utasításának hatására a programszámláló aktuális értéke a verembe kerül, majd a vezérlés a szubrutinra adódik. A visszatérés az RTS utasítással lehetséges. Ekkor a veremben lévő mutató visszakerül a programszámlálóba. A JSR utasítás 2 byte-ot foglal a veremben, ezért nem lehet tetszőleges számú szubrutint egymásba skatulyázni, hiszen a verem közben megtelhet. Ez a határ azonban inkább elméleti, mint gyakorlati, mert ritka az olyan programozási feladat, amelyben az egymásba ágyazott szubrutinok száma több húsznál.

Input-output műveletek

A számítógép programozásában a kommunikáció irányítása az egyik legterjedelmesebb terület. Az input-output műveletek lebonyolítására a processzor nem biztosít utasításokat, de az alaputasítások segítségével bármilyen adatátviteli rutin megvalósítható, amely a hardver szempontjából nem kizárt. A KERNAL operációs rendszer bőven kínál ilyen szubrutinokat. Ezeket a szubrutinokat a függelékben ismertetjük.

2.3.3. Gépi kódú programok a gyakorlatban

a) Hogyan lehet programból észlelni a lemezegység jelenlétét?

Észlelni, lehetőleg anélkül, hogy a program valamilyen módon megállna (a BASIC hiba-üzenettel, a gép lemerevedéssel). Ez a kis program megoldja ezt a problémát:

LDA #\$0F	logikai file-szám
LDX #\$08	egységszám
LDY #\$0F	parancscsatorna, másodlagos cím
JSR SETFLS	a fileparaméterek beállítása
LDA #\$00	a név hossza nulla
JSR SETNAM	a file-név paraméterek beállítása
JSR OPEN	a csatorna megnyitása
LDX #\$0F	logikai file-szám
JSR CHKOUT	az output egység kijelölése
LDA \$90	a státusz
PHA	ideiglenes tárolása
JSR CLRCHN	az output kijelölés megszüntetése
LDA #\$0F	logikai file-szám
JSR CLOSE	a csatorna lezárása
PLA	a státusz visszatöltése
BMI ERROR	ha a 7. bit magas, DEVICE NOT PRESENT
RTS	

ERROR LDA ... hibakezelés

A megoldás azon alapul, hogy az IEC (soros) busz DATA vonalát a külső egység 1 ms-on belül köteles alacsonyra állítani azután, hogy feléje irányuló címzést érzékelt. Amennyiben ez nem következik be, a CHKOUT rutin alatt a státuszbyte (\$90) 7. bitje magasra vált. Ez azt jelenti, hogy a megcímzett egység nincs a buszon (egyáltalán nincs is vagy csak nincs bekapcsolva).

b) A lemez tartalomjegyzékének megjelenítése

Amikor a lemezegység utasítást kap, hogy a \$ file-t olvassa (amelyet egyébként csak olvashat, írni nem írhat), az operációs rendszerében levő program a lemezen tárolt adatok alapján összeállítja a lemez tartalomjegyzékét. Abban a formában, amiben a lemezen van, a BASIC interpreter nem tud vele mit kezdeni. A lemezegység így nem is továbbíthatja, mert ha betöltődik a számítógép programtárába, a LIST rutin nem tudja megjeleníteni. A tartalomjegyzék szerkezete meg kell egyezzen a BASIC program tárbeli szerkezetével. A lemezegység ezért erre a formátumra hozza a tartalomjegyzék adatait. A BASIC programok tárbeli elhelyezkedését a BASIC programozással foglalkozó fejezetben részletesen ismertetjük. Az alábbi program — bár nem tölti be a tárba a tartalomjegyzéket — kénytelen figyelembevenni az említett speciális felépítés sajátosságait.

	LDA #\$01	logikai file-szám
	LDX #\$08	egységszám
	LDY #\$00	másodlagos cím
	JSR SETFLS	a file paramétereinek beállítása
	LDA #" "	a file nevének
	STA \$033C	átmeneti tárolása
	LDA #\$01	a file-név hossza
	LDX #\$3C	a file-név címe, alsó byte
	LDY #\$03	a file-név címe, felső byte
	JSR SETNAM	a file-név paraméterek beállítása
	JSR OPEN	a file megnyitása
	LDX #\$01	a megnyitott file száma
	JSR CHKIN	az input-mód kijelölése
	LDA #\$00	
	STA \$90	a státusz törlése
	LDY #\$03	a kezdőcím és a sorláncolás átugrása
AA0	STY \$FB	a számláló ideiglenes tárolása
	JSR INPUT	egy byte beolvasása
	STA \$FC	a byte ideiglenes tárolása
	LDY \$90	file vége ?
	BNE AA3	igen
	JSR INPUT	egy byte beolvasása
	LDY \$90	file vége ?
	BNE AA3	igen
	LDY \$FB	a számláló
	DEY	csökkentése
	BNE AA0	még nem zero, vissza
	LDX \$FC	a sorszám (blokkszám) alsó byte-ja
	JSR \$BD0D	a sorszám kiírása, felső byte az AC-ban
	LDA #\$20	egy szóköz
	JSR PRINT	kiírása
AA1	JSR INPUT	egy byte beolvasása
	LDX \$90	file vége ?
	BNE AA3	igen
	TAX	vége a sornak ?
	BEQ AA2	igen, új sort kezdeni
	JSR PRINT	a byte kiírása
	JMP AA1	és folytatás
AA2	LDA #\$0D	Carriage Return
	JSR PRINT	kiírása

	LDY #\$02	az átolvasandó byte-ok száma 4 (2*2)
	BNE AA0	feltétel nélküli ugrás
AA3	JSR CLRCHN	az input csatorna törlése
	LDA #\$01	a logikai file-szám
	JMP CLOSE	a file lezárása

c) Utasítás a lemezegységnek

Az előző példában a lemezegységtől adatokat vártunk. Némileg más a helyzet, ha parancsot továbbítunk a lemezegység felé. A következő program, amely ezt mutatja be, bizonyos tekintetben hasonlít a HELP PLUS @I parancsához. A példa egy byte hosszúságú parancsot továbbít, de több byte hosszúságú adatot is küldhetünk a program segítségével. Ehhez csak néhány módosítás szükséges. A töltő és mentő rutinokkal megkerülhető a BASIC LOAD és SAVE utasítása, ezáltal átléphetők a korlátaik is.

LDA # "I"	inicializálás parancs
STA \$033C	
LDA #\$01	a paraméter hossza
LDX #\$3C	a paraméter címe, alsó byte
LDY #\$03	a paraméter címe, felső byte
JSR SETNAM	
LDX #\$08	egységszám
LDY #\$6F	másodlagos cím, 15-ös csatorna (+\$60)
JSR SETFLS	
LDY #\$00	a státusz
STY \$90	törlése
LDA \$BA	egységszám
JSR LISTEN	
LDA \$B9	másodlagos cím
JMP \$F3EA	a paraméter kiírása az IEC-buszra

Program betöltése gépi programmal:

LDX #\$08	egységszám
LDY #\$01	másodlagos cím (lehet \$00 is)
JSR SETFLS	
LDA #\$04	a név hossza
LDX #NAME<	a név címe, alsó byte
LDY #NAME>	a név címe, felső byte
JSR SETNAM	
LDA #\$00	LOAD/VERIFY kapcsoló
STA \$9D	
JSR LOAD	a betöltés végrehajtása
BCS ERROR	kilépés, ha töltés közben hiba volt
RTS	

NAME	.TEXT "TEST"	file-név
ERROR	LDA ...	hibakezelés

Program mentése gépi programmal:

```
LDX #$08      egységszám
JSR SETFLS
LDA #$04      a név hossza
LDX #NAME<    a név címe, alsó byte
LDY #NAME>    a név címe, felső byte
JSR SETNAM
LDA #$00      a kezdőcím: $8000
LDX #$80
STA $FD      a kezdőcím mutatói
STX $FE      a nulláslapra
LDA $FD      mutató a nulláslapon levő címre
LDX #$FF      végcím, alsó byte
LDY #$8F      végcím, felső byte
JSR SAVE      a mentés végrehajtása
BCS ERROR    kilépés, ha mentés közben hiba volt
RTS
```

```
NAME .TEXT "TEST" file-név
```

```
ERROR LDA ... hibakezelés
```

Számtalan felhasználási lehetőséget kínál a ROM (úgy a BASIC, mint a KERNAL) is. Mivel e két programegyüttest profi programozók írták, rendkívül sok és hasznos példát nyújt programozási feladatok megoldására. A kezdő programozó sokat tanulhat belőlük.

A ROM egyik legjellemzőbb sajátossága, hogy a legfontosabb rendszerrutinok indirekt címzésű ugróutasításokkal hívhatók. E megoldás teszi lehetővé azt, hogy a ROM-ok fix, nem változtatható programjuk ellenére igenis módosíthatók. A ROM-rutinok egy ugrótábla módosításával egyszerűen „kicserélhetők” általunk írt rutinokkal.

Az ugrótábla a memória 3. lapján (\$0300-tól) helyezkedik el. Az, hogy tetszés szerint módosíthatók a mutatók, egy különleges megoldást kínál. Ez a megoldás a töltés (LOAD) utáni automatikus programindítás vagy röviden autostart. Az autostart programok nagyjából két csoportra oszthatók. Az egyik részük a töltés befejezése után aktivizálódik, a másik pedig már töltés közben működésbe lép.

A LOAD rutin lefutása után működésbe lépő autostart program azt használja ki, hogy a töltés után a vezérlés a BASIC-hibaüzenet rutinjára kerül. A rutin vektora pedig a \$0300 — 0301 címeken helyezkedik el. A betöltött program ezt írja át, ezért a töltés befejezése után a módosított vektor alapján a betöltött program elindulhat.

A másik változat ugyanezen az elven alapul, de más a gyakorlati megoldása. Itt ugyanis a STOP billentyű lenyomását ellenőrző rutin vektorát változtatja meg a program. A LOAD rutin a működése során ciklikusan ellenőrzi a STOP billentyű lenyomását. Amint a vektort átírta a betöltött program, a LOAD rutin működése felfüggeszthető. Erre a megoldásra kiváló példa MOVMON tármonitor, amely betöltés közben tudakolja, hogy hová helyezze a monitorprogramot.

A töltés után aktivizálódó autostart programoknak van egy különleges válfaja, amelyik a processzor visszatérési vermét írja felül.

Egy autostart program:

```

02C4 12 06 0A 00 93 22 53 54 BASIC program:
02CC 41 52 54 22 2C 38 2C 31 LOAD "START",8,1
02D4 00 00 00

02D7 A3 8B LDA #$8B a hibauzenet vektor
02D9 A2 E3 LDX #$E3
02DB 8D 00 03 STA $0300 visszaállítása
02DE 8E 01 03 STX $0301
02E1 A3 01 LDA #$01 logikai file-szám
02E3 A6 TAY másodlagos cím az YR-ben
02E4 A2 03 LDX #$03 egységszám
02E6 20 BA FF JSR $FFBA SETFLS
02E8 A3 04 LDA #$04 a név hossza
02EB A2 FC LDX #$FC a név címe, alsó byte
02ED A0 02 LDY #$02 a név címe, felső byte
02EF 20 BD FF JSR $FFBD SETNAM
02F2 A3 00 LDA #$00 LOAD/VERIFY kapcsoló ($00=LOAD)
02F4 85 9D STA $9D
02F6 20 D5 FF JSR $FFD5 LOAD rutin
02F8 4C 10 08 JMP $0810 a betöltött program elindítása

02FC 54 45 53 .TEXT "TEST" a betöltendő program neve

0300 D7 .BYTE $D7 a hibauzenet vektor az
0301 02 .BYTE $02 indítóprogramra mutat

```

A program — amelynek neve START legyen — betölthető ,8 opcióval is, mert ha BASIC-területre kerül, az elején lévő BASIC program RUN után helyesen betölti a gépi programot. Ez a változat külön hívja a TEST nevű főprogramot. Elképzelhető azonban olyan megoldás is, amelyiknél az autostart a főprogram elején van. Az ilyen megoldás egyszerűbb, de kevésbé helytakarékos, mert a programállománynak tartalmaznia kell az indító- és a főprogram között elhelyezkedő tárterületeket is (kazettapuffer, képernyő stb.). Autostart-generátor sok létezik, s ezek kellően változatosak is. Azonban megfelelő programozási gyakorlat és a gépi nyelv ismerete birtokában saját magunk is állíthatunk elő testre szabott autostart programot.

A programozók mindig is törekedtek arra, hogy a programokhoz minél nehezebben férjenek hozzá illetéktelenek, minél nehezebb legyen a programot feltörni, lényegét ellopní. Persze a C64-es nyitott hardvere miatt nem sok sikerrel jártak ezek a próbálkozások. Összességében leszögezhető: tökéletes védelem nincs, csak kellően bonyolult. A programozók ravaszsága növekedtével a crackerek (programtolvajok) ügyessége is növekedett. A kettejük közötti küzdelem érdekes módon ért véget. A C64-es kiment a divatból. Kiszorították az IBM PC-k és klónjaik. Így már nem érte meg védeni az elavult technológiához tartozó programokat.

Tulajdonképpen e küzdelem eredménye volt az autostart programok megjelenése is. De hasonló okok miatt vált fontossá a hardver RESET kivédése is. A processzor ismertetésénél említettük, hogy a RESET lábra érkező alacsony jel a processzort alapállapotba hozza. A folyamat eredménye, hogy a processzor az utasításszámlálóba tölti a memória (ROM) \$FFFC—FFFD címén található byte-okat. E byte-ok a \$FCE2 címet adják; itt kezdődik az ún. RESET program. A program némi előkészítés után megvizsgálja, hogy a memória meghatározott területén (\$8003—8008) megtalálható-e a CBM80 kulcsszó. Ha megvan, akkor nem a ROM-ban folytatja a RESET program végrehajtását, hanem végrehajt egy indirekt

ugrást a \$8000 — 8001 címeken található értékek felhasználásával. Éppen ez az, amit kihasználhat a programozó. Ugyanis a \$8000 — 8001 tárcímek tartalmát a programozó szabadon módosíthatja. (Itt meg kell jegyezni, hogy ez a lehetőség — az indirekt ugrás — nem a programozók kedvéért került be az operációs rendszerbe. Erre azért volt szükség, hogy a bővítőportba helyezett memóriaegység programja a gép bekapcsolása után automatikusan elinduljon.) Így a RESET program lefutását a programozó saját elképzelése szerint módosíthatja.

Egy példa RESET-rutinra:

8000	09 80		RESET-vektor: \$8009
8002	46 80		NMI-vektor : \$8046
8004	C3 C2 CD 38 30		CBM80 azonosító
8009	A2 05	LDX #\$05	
800B	20 A3 FD	JSR \$FDA3	a megszakítás inicializálása
800E	20 50 FD	JSR \$FD50	a BASIC munkaterület inicializálása
8011	A9 80	LDA #\$80	a BASIC RAM felső határa a \$80. lap
8013	8D 84 02	STA \$0284	
8016	20 15 FD	JSR \$FD15	KERNAL I/O vektorok beállítása
8019	20 5B FF	JSR \$FF5B	Video-reset
801C	59	CLI	
801D	20 53 E4	JSR \$E453	a BASIC vektorok beállítása
8020	20 BF E3	JSR \$E3BF	a BASIC RAM inicializálása
8023	20 44 E4	JSR \$E444	BASIC NEW utasítás
8026	A2 FB	LDX #\$FB	a verem inicializálása
8028	9A	TXS	
8029	20 2F 80	JSR \$802F	a BASIC-bővítés bekapcsolása
802C	4C 86 E3	JMP \$E386	BASIC melegstart
802F	A9 00	LDA #\$00	a szín fekete - képernyő
8031	A2 0F	LDX #\$0F	a szín szürke - Karakterek
8033	8D 20 D0	STA \$D020	a Keret színe
8036	8D 21 D0	STA \$D021	a háttér színe
8039	8E 86 02	STX \$0286	a betűk színe
803C	20 44 E5	JSR \$E544	képernyőtörlés
803F	20 E8 80	JSR \$80E8	a bővítés bekapcsolása
8042	60	RTS	
8043	4C 72 FE	JMP \$FE72	KERNAL NMI-rutin
8046	20 BC F6	JSR \$F6BC	STOP ellenőrzés előkészítése
8049	20 ED F6	JSR \$F6ED	a STOP billentyű ellenőrzése
804C	D0 F5	BNE \$8043	nincs lenyomva
804E	20 A3 FD	JSR \$FDA3	a megszakítás inicializálása
8051	20 18 E5	JSR \$E518	Video-reset
8054	20 2F 80	JSR \$802F	a bővítés bekapcsolása
8057	4C 7B E3	JMP \$E37B	BASIC NMI-rutin

Ez a RESET rutin alkalmas a HELP PLUS inicializálására. Mind RESET, mind NMI után bekapcsolja a bővítést, inicializálja a megszakítást, beállítja a képernyő színét. Ha a BASIC bővítést bekapcsoló rutint (\$80E8) alkalmas más rutinra cseréljük, elérhetjük azt, hogy a tárban levő program hardver RESET után mindig újrainduljon.

A programvédelem igen változatos trükkjei közül ez csak kettő. Aki többre kíváncsi, átfogó összefoglalást találhat dr. Lengyel — Varga: *Lakat alatt a C64-es* című könyvében (NOVOT-RADE Kiadó, 1988).

3. A BASIC

„A BASIC univerzális nyelv: majdnem minden feladat jobban programozható más nyelven, de semmilyen másik nyelvben nem lehet majdnem mindent ilyen jól programozni.”

Frank Ostrowski

A BASIC-et 1962 — 64 között fejlesztette ki Thomas Kurtz és John Kemeny az amerikai Dartmouth University számítástechnikai laboratóriumában. Az új programnyelv koncepciója 1962-ben Thomas Kurtz fejében született meg. Akkoriban még nem létezett olyan számítógép, amelyet egyidejűleg akár csak két felhasználó használhatott volna. Kemény (magyar származása miatt a továbbiakban magyarul írjuk a nevét) és Kurtz először ezt a műszaki problémát oldotta meg. Az ő laboratóriumukban hozták létre a világ első időosztásban (time-sharing) működő számítógépét. Az időosztásos rendszer teszi lehetővé, hogy a számítógép központi egysége látszólag egy időben több terminált kiszolgáljon. (Valójában a központi egység precízen beosztott időzítés szerint először az egyik terminállal foglalkozik, majd az ott futó programot megszakítva, és a változókat ideiglenesen elmentve, sorra veszi a következő terminált, és így tovább. Ha a központi egység ezt elég gyorsan teszi, a terminál előtt ülő ember úgy érzi, mintha a számítógép egyedül az ő feladatával foglalkozna.) A műszaki nehézségek leküzdése után Kurtz és Kemeny kidolgozták a BASIC elméleti koncepcióját. Ennek a koncepciónak a lényegét nyolc szabályban foglalták össze.

1. A nyelv kezdők számára könnyen elsajátítható legyen.
2. A nyelv bővíthető legyen, hogy a profik is használhassák, de ez ne akadályozza a kezdőket.
3. A parancsokkal a lehető legkülönbözőbb programok is létrehozhatók legyenek.
4. A nyelv viszonylagos előnyre tegyen szert abból, hogy a felhasználó és a számítógép közvetlenül kommunikálhat egymással.
5. A hibaüzenetek érthetőek legyenek.
6. A kis programok feldolgozása rövid ideig tartson.
7. A programozáshoz ne kelljen hardverismereteket tanulni.
8. A programozónak ne kelljen az operációs rendszer különlegességeivel foglalkoznia.

A BASIC, mint láthatjuk, elsősorban kezdők számára íródott. Tulajdonképpen a neve is innen származik (BASIC = Beginner's All-purpose Symbolic Instruction Code — kezdők általános célú szimbolikus utasításkódja). Kurtz és Kemeny következetesen meg is valósította elképzelésének nyolc pontját. Születésének napja 1964. május 1., amely a BASIC későbbi hatását figyelembe véve, történelmi dátum. Ekkor mutatták be először az új operációs rendszert egy új elven működő számítógépen. A siker nem váratott magára túl sokáig, a BASIC rövid idő alatt igen népszerűvé vált a diákok között. Kurtz és Kemeny szándékosan nem szabadal-

maztatták találmányukat, hogy a nyilvánosság tovább növelje népszerűségét és ezáltal elterjedtségét.

Az első BASIC operációs rendszer még nem nagyon hasonlított a mai utódokra. Először is nem a manapság igen elterjedt interpreter, azaz értelmező program, hanem sokkal inkább szerkesztő-fordító (editor-compiler) program volt. De a nyelv utasításainak alapkészlete azonos volt a mai BASIC interpreterek utasításkészletével, szerepelt benne az INPUT, a LET, a PRINT, a RETURN, az END, a FOR — NEXT, a GOTO, a REM, az IF — THEN, a READ, a DATA, a DIM, a DEF és a GOSUB utasítás.

Az eredeti BASIC-nek néhány éven belül új verziói jelentek meg; nagyjából tíz év múlva megjelent az első interpreter, amely tovább „szelidítette” a számítógépet a felhasználóhoz. A mikroszámítógépek világában csaknem szabvány, hogy a géppel együtt valamilyen BASIC-verziót is szállítanak.

A BASIC „életének” első 25 évében hallatlanul sokat fejlődött, de az a nyolc pont, amely alapelveinek lényege, tulajdonképpen ma is meghatározza a BASIC alatti programozás szellemét.

A Commodore BASIC 2.0, amely a C64-es alapnyelve, a késői BASIC verziók egyike. Ez a verzió, 1982-es megjelenése után a C64-es hihetetlen népszerűsége miatt nagy ismertségre tett szert. Magára az alapverzióra is számos bővítés készült (Simon's Basic, SUPERGRAPHIC, GRAPHICS BASIC, DBASIC, MASTER '64 stb.), de megjelentek a C, PASCAL, FORTH programnyelvek operációs rendszerei is. Az egyébként is elég jól sikerült alapnyelv a bővítésekkel együtt igazán sokoldalúvá tette a C64-est. Hogy ez az alkalmazott BASIC verzió érdeme, vagy az átgondolt hardverkoncepcióé, azt egyértelműen nem lehet eldönteni egyik javára sem. De a sok éven át tartó siker önmagáért beszél.

3.1. A Commodore BASIC 2.0 alapelemei

A Commodore BASIC nem a szabványként elfogadott ASCII jelkészletet használja, a különbség azonban nem jelentős.

Alfanumerikus jelek:

0 1 2 3 4 5 6 7 8 9

a b c d e f g h i j k l m n o p q r s t u v w x y z

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Műveleti, írás- és speciális jelek:

+ - * / < > = ↑
 ! () : ; , . ? szóköz [] '
 " # \$ % @ π & £ ←

A *grafikus jelek* többsége nem tartozik az ASCII kódkészlethez, a C64-esben a grafikai programok támogatására szolgálnak.

3.1.1. Változók és állandók

A programokban használt változó és állandó fogalma nem választható élesen ketté, mert ami a program számára állandó, az az interpreter számára változó. A változók és állandók kétféleképpen csoportosíthatók. Szerepük szerint megkülönböztetünk egyszerű és indexes változókat; értelmezési módjuk alapján pedig egész típusú, lebegőpontos, illetve szöveges változókat. Az utóbbiak mindhárom típusára igaz, hogy lehetnek egyszerű és indexelt változataik.

Az egész típusú változó értékének -32768 és $+32767$ közé kell esnie. A pozitív egész számok előjele elhagyható, a negatív számokat azonban minden esetben jelölni kell. A lebegőpontos változó abszolút értékének a $[2.93873588 \cdot 10^{-39}, 1.70141183 \cdot 10^{38}]$ intervallumba kell esnie. Ennél kisebb vagy nagyobb számok ábrázolása nem lehetséges. A 0,01-nél kisebb és a 999 999 999-nél nagyobb számokat az interpreter exponenciális alakban írja ki, pl. $1E10 = 10\,000\,000\,000$.

A szöveges változó maximum 255 alfanumerikus és/vagy grafikus karaktert tartalmazhat, de tartalmának megadásánál figyelembe kell venni, hogy a képernyőről egy művelettel csak maximum 80 karakter olvasható be a változóba (elméletileg, mert a változó nevének megadása és a műveleti jel miatt ez 72—75-re csökken).

A változónevek elvileg 255 karakter hosszúak is lehetnek, de az interpreter az első két betű alapján különbözteti meg őket. A változók neve állhat számokból és betűkből is, de sohasem kezdődhet számjeggyel. A típust a változónév után álló jellel különböztethetjük meg. Jelzés hiányában a változót valós típusú numerikus változónak kell tekinteni. Ha a név mögött % jel áll, akkor egész típusú, ha \$ jel áll, az szöveges típusú változót jelent. A változónév mögött helyezkedhet el a tömbváltozók indexe. Az index több mutatóból is állhat. Egy mutató esetén a tömböt vektornak, két vagy több mutató esetén két- vagy többdimenziós mátrixnak nevezzük. A vektor vagy mátrix elemeire az indexekkel hivatkozhatunk.

Változóinknak gyakorlatilag bármilyen kétbetűs nevet adhatunk, de a BASIC interpreter miatt ki kell hagynunk a következő kombinációkat:

IF ON GO TI OR FN TO TI\$ ST. Ugyanígy nem használhatjuk változónévként a BASIC utasítások neveit sem. CL változónk például lehet, de CLOSE változónk nem. Ilyenkor hibaüzenetet kapunk.

BASIC-en belüli változók, ST, TI, TI\$

Az interpreter három változót fenntart saját céljaira. Ezeket a változókat a programozó nem definiálhatja, nem adhat nekik értéket (a TI\$ ez alól kivétel).

Az ST a státuszváltozó, tartalma az input-output műveletek eredménye szerint változik. Értéke alapján következtethetünk a művelet alatt előfordult hibára. Lehetséges értékei:

- 0: minden rendben
- 1: a külső egység nem nyugtázta az adat átvételét
- 2: a külső egység nem küldött adatot
- 64: a file véget ért
- 128: a megcímzett egység nincs jelen a buszon

A TI és a TI\$ az interpreter belső órája; lényegében a hardvermegszakításokat számlálja. Mivel ezek kb. 60 ms-onként ismétlődnek, a számláló alkalmas idő mérésére; pontossága azonban meg sem közelíti a CIA TOD óráinak pontosságát. A számlálókat a nulláslap \$A0–A2 címein találhatjuk meg. A TI\$ ebből a számlálóból képződik, hatjegyű, szöveges típusú változó. Mivel a TI\$ írható, ezért közvetetten a TI is módosítható, mert az interpreter a TI\$ értékét átszámítja a léptetési ütemnek megfelelően. A TI\$ az időt óra — perc — másodperc alakban tartalmazza.

A változónév ábrázolása

Az interpreter a változók számára kétbyte-os azonosítókat engedélyez, a tárban a változó tényleges tartalma előtt ezt a két byte-ot tárolja. Hogy a három változótípus megkülönböztethető legyen, a változónév második vagy mindkét byte-jának 7. bitjét magasra állítja. A változóterületen így, kódolt formában tárolja a változókat, és az egyes típusokat a névbyte-ok 7. bitje alapján különbözteti meg.

Valós típusú változó esetén a változónév mindkét byte-jának ASCII-kódját tárolja, módosítás nincs. Ha a változónév egybyte-os, a második byte tartalma \$00.

Egész típusú változó esetén a változónév mindkét byte-jának legfelső (7.) bitje magas. Ha a változónév egybyte-os, a második byte \$80.

Szöveg típusú változó esetén a változónév első byte-ja az eredeti ASCII-kód. A második byte az ASCII-kód magasra állított 7. bittel. Ha nincs második byte, a kód \$80.

A programozó által definiált függvény megadására szolgáló FN utasítás változóját szintén a változóterületen tartja az interpreter, tartalma azonban nem érték, hanem mutató. Mutató a függvényre, amely a BASIC programban van. Az FN változónevének első byte-ja az ASCII-kód, magasra állított 7. bittel. A második byte vagy az ASCII-kód, vagy \$00.

A tömbváltozók neveit ugyanilyen módon oszthatjuk típusokra, a különbség csupán annyi, hogy ott nincs FN változó.

3.1.2. Kifejezések

Tartalmuk, összetevőik alapján megkülönböztetünk aritmetikai, illetve szöveges kifejezéseket. Az aritmetikai kifejezésekben aritmetikai, logikai és relációs operátorok szerepelhetnek.

Aritmetikai operátorok (alapműveletek):

- + összeadás
- kivonás
- * szorzás
- / osztás
- ↑ hatványozás

Logikai operátorok:

- AND logikai ÉS (szorzás)
- OR logikai megengedő VAGY (összeadás)
- NOT logikai NEM (tagadás)

A logikai operátorokkal végzett műveletek a logikai algebra részét képezik, amelyet részletesen tárgyalunk a függelék megfelelő részében.

Relációs operátorok:

- < kisebb, mint
- = egyenlő
- > nagyobb, mint
- <= kisebb vagy egyenlő
- >= nagyobb vagy egyenlő
- <> nem egyenlő

A relációs operátorok alkalmazása esetén az eredmény vagy -1, vagy 0 lehet, attól függően, hogy az összehasonlítás eredménye igaz-e vagy hamis. Az igaz állítás értéke -1.

A műveletek közötti elsőbbségi rend:

1. ↑ hatványozás
2. - előjelváltás
3. * / szorzás, osztás
4. + - összeadás, kivonás
5. < = > relációs operátorok
6. NOT logikai NEM
7. AND logikai ÉS
8. OR logikai VAGY

Az azonos szintű műveletek között a balról jobbra haladás szabálya érvényes. Az elsőbbségi sorrend zárójelezéssel megváltoztatható.

Néhány példa aritmetikai kifejezésekre:

$$B = -A$$

$$C = A \uparrow B$$

$$A = 2 * B + C - D$$

$$E = (A + B) * (A - B)$$

$$D = (-B + (B * B - 4 * A * C) \uparrow (1/2)) / (2 * A)$$

$$A = B \text{ AND NOT } C$$

$$A = B \text{ OR } C$$

$$A = 18 < > 6 * 2$$

$$C = (A = B)$$

A szövegfüggvények által végzett műveleteket nem tekintve, a szöveges kifejezésekkel csak kétféle művelet végezhető: az összeadás és az összehasonlítás.

Az összeadás valójában összefűzés, mert két szöveg összeadása egymáshoz kapcsolásukat jelenti. Például:

```
A$ = "OSSZE": B$ = "ADAS" : PRINT A$ + B$
OSSZEADAS
```

Az összehasonlítást az interpreter karakterenként végzi. A betűk és számok nagyságát az ASCII kódtáblázatban elfoglalt helyük (pozíciójuk) határozza meg, ennek alapján viszonyítja egymáshoz. A művelet eredménye itt is 0 vagy -1 lehet.

3.1.3. Értékadás

A programozás során nem kerülhető el, hogy bizonyos változóknak kezdőértéket adjunk. Ez megvalósítható a LET utasítás segítségével. A Commodore BASIC-ben, mint a legtöbb BASIC-verzióban, a LET utasításszót nem szükséges kiírni. Emiatt az értékadó kifejezés kétféleképpen nézhet ki:

a) 500 A = 10 [sorszám] [változó] = [érték vagy kifejezés]

b) 500 LET A = 10 [sorszám] LET [változó] = [érték vagy kifejezés]

Az értékadó utasítás hatására megtörténik a változó azonosítása, a kifejezés kiértékelése, értékének meghatározása, valamint az érték hozzárendelése a változóhoz.

3.1.4. Utasításcsoportok

Rendszervező utasítások

CONT, END, LIST, LOAD, NEW, RUN, STOP, SAVE, VERIFY

- a program betöltésével kapcsolatos utasítások (LOAD, SAVE, VERIFY)
- a programot törölő és listázó utasítások (NEW, LIST)
- a program futását vezérlő utasítások (CONT, END, RUN, STOP)

A parancsok többsége általában csak parancsmódban használatos, bár mindegyik beépíthető programba is.

Adatforgalmat lebonyolító utasítások

DATA, RESTORE, INPUT, READ, PRINT, SPC, TAB, GET

- input utasítások (INPUT, READ, GET)
- output utasítás (PRINT)
- kiegészítő utasítások (DATA, RESTORE, SPC, TAB)

Az input utasításoknak azért van több változata, mert a BASIC alkotói a paraméterezést szerették volna elkerülni, s inkább új utasításokat hoztak létre. Az SPC és a TAB önállóan nem léteznek, csak a PRINT argumentumaként.

Ciklusszervező utasítások

FOR, NEXT, STEP, TO

A programozás során rendkívül fontosak a ciklusok, amelyeket szinte egyetlen program sem nélkülözhet. Az utasítások a ciklusfej (FOR, TO, STEP) és ciklusvég (NEXT) csoportokra bonthatók. A ciklusszervezés opcionális eleme a STEP utasítás, amelynek elhagyása automatikusan egységnyi (1) lépésközt feltételez. A Commodore BASIC nem követeli meg a NEXT utasítás után álló változónév megadását. Ki lehet tenni, de nem szükséges.

Vezérlésátadó utasítások

GOTO, GO, ON, TO

Bizonyára feltűnő, hogy a TO utasítás itt is szerepel. Az interpreter TO rutinja érzékelni tudja, hogy honnan hívták, a FOR vagy a GO utasításból. Ezek az utasítások a programvégrehajtó rutint feltétel nélküli ugrásra kényszerítik. A feltétel nélküli jelző az ON GOTO esetében ugyan nem igaz maradéktalanul, de a lényegtől itt sem tér el nagyon. A programvégrehajtás az utasítás után álló sorszám szerinti programsoron folytatódik.

*Szubrutinkezelő utasítások***GOSUB, RETURN**

A szubrutinok a strukturált, de egyáltalán minden programozás elengedhetetlen elemei. A Commodore BASIC a szubrutinokat két utasítással kezeli: a hívó GOSUB-bal és a lezáró RETURN-nel. Szubrutin hívásakor az interpreter elmenti a verembe az aktuális visszatérési címet, majd a RETURN feldolgozásakor visszakeresi azt.

*Változókezelő utasítások***CLR, DIM, LET**

Törlő (CLR) és deklaráló (DIM, LET) utasítások. A LET és a DIM létrehozza, a CLR megszünteti a listázható változókat.

*Szövegkezelő utasítások***ASC, CHR\$, LEN, LEFT\$, MID\$, RIGHT\$, STR\$, VAL**

Ezek az utasítások valójában szövegfüggvények, amelyek a paraméterek függvényében hajtanak végre különböző műveleteket az argumentumban megadott szövegen.

*Függvénykezelő utasítások***AND, ABS, ATN, COS, DEF, EXP, FN, INT, LOG, OR, POS, RND, SGN, SQR, SIN, TAN**

- logikai operátorok (AND, OR, NOT)
- matematikai függvények (ABS, ATN, COS, EXP, INT, LOG, SGN, SQR, SIN, TAN)
- programozható függvény (DEF, FN)
- egyéb függvények (POS, RND)

A matematikai függvények egyargumentumos függvények, amelyek kiszámítják a matematikai függvény értékét a megadott argumentumnál. A programozható függvényeket a programozó saját maga építheti fel a DEF FN utasítás segítségével. A későbbiekben a saját függvényre az FN utasítással lehet hivatkozni.

*Perifériakezelő input-output utasítások***CLOSE, CMD, INPUT#, GET#, OPEN, PRINT#**

Az utasítások lényegében file-ok kezelésére szolgálnak. Az OPEN és a CLOSE a file-ok megnyitására és lezárására hivatott. A CMD az adat irányítását változtatja meg, de csak output esetén. A megváltozást nem ellenkező irányra váltásként kell értelmezni, hanem más egységre váltásként. Az INPUT#, GET#, PRINT# utasítások az INPUT, GET, PRINT utasítások file-kezelő módosulatai.

*Feltételes elágazást létrehozó utasítások***IF, THEN**

A feltételes elágazások fontosak a program szerkezetében. Segítségükkel lehet végrehajtani előre megadott feltételeken alapuló döntéseket. Ha az IF mögött álló feltétel igaznak bizonyul, a THEN mögött álló utasítás(ok) lesz(nek) végrehajtva. Ha nem igaz az állítás, a vezérlés a soron következő programsorra adódik.

*Egyéb utasítások***FRE, POKE, PEEK, REM, SYS, USR, WAIT**

Az utasítások egy része függvényként is értelmezhető (FRE, PEEK, USR, WAIT). A többi három valamilyen szempont szerint elüt az összes többi utasítástól. Például a REM nem csinál semmit. A SYS gépi programot hív. A POKE tárhelyet tölt fel.

3.2. A BASIC 2.0 utasításai

A Commodore 64-esen a BASIC nyelv utasításai általában kétbetűs, rövidebb alakban is begépelhetők. Ilyenkor a második betűt Shifttel együtt kell leütni. Könyvünben ezt jelöltük. A függvények után felírt argumentumot minden esetben zárójelek között kell megadni — ahogyan a szintaxisban jelezzük is.

ABS

Token: \$B6 #182
Belépési pont: \$BC58

Aritmetikai függvény; rövidítése: AB

Szintaxis: ABS (*num. kifejezés*)

Példa: ABS(X) : ABS(2*I)

Az utasítás az argumentum abszolút értékét veszi, azaz megfosztja azt esetleges negatív előjelétől. Az utasítás egy kifejezést igényel, működése során kiszámítja a kifejezés előjelhelyes értékét, majd előjelét pozitívvá változtatja.

AND

Token: \$AF #175
Belépési pont: \$AFE9

Logikai-aritmetikai művelet; rövidítése: AN

Szintaxis: *num. kifejezés* AND *num. kifejezés*

Példa: A = A AND B : IF (A AND B) < 23 THEN ...

Az utasításszó jobb és bal oldalán álló argumentumok között logikai ÉS műveletet hajt végre. A logikai ÉS igazságtáblázata, ha a művelet bitenként kerül végrehajtásra:

0 AND 0 = 0

0 AND 1 = 0

1 AND 0 = 0

1 AND 1 = 1

ASC

Token: \$C6 #198

Belépési pont: \$B78B

Szöveges függvény; rövidítése: AS

Szintaxis: ASC (szövegkifejezés)

Példa: ASC(A\$) : ASC("X") : ASC("XYZ")

Az argumentumban felírt szövegkifejezés első karakterének ASCII-kódját adja meg. Az argumentum több karakterből is állhat.

ATN

Token: \$C1 #193

Belépési pont: \$E30E

Aritmetikai függvény; rövidítése: AT

Szintaxis: ATN (num. kifejezés)

Példa: ATN(5) : ATN(A * 0.7) : ATN(A + B + C)

A matematikai arkusz tangens függvény. Argumentuma bármilyen valós érték lehet, amelyet a C64-es ábrázolni képes. Ellenkező esetben ?OVERFLOW ERROR üzenetet kapunk.

CHR\$

Token: \$C7 #199

Belépési pont: \$B6EC

Szöveges függvény; rövidítése: CH

Szintaxis: CHR\$ (num. kifejezés)

Példa: CHR\$(32) : CHR\$(A + B) : CHR\$(12 + C)

A függvény argumentuma mutatja azt az ASCII karaktert, amelyet felhasználni kívánunk. Az argumentum értéke 0-tól 255-ig terjedhet. A függvény szöveges típusú eredményt hoz létre.

CLOSE

Token: \$A0 #160

Belépési pont: \$E1C7

Utasítás; rövidítése: CLO

Szintaxis: CLOSE logikai file-szám

Példa: CLOSE 2 : CLOSE A

Lezárja a megnyitott logikai file-t, ha a paraméternek megfelelő bejegyzés létezik a megnyitott file-ok táblázatában. Ha nem létezik, hibaüzenet nélkül fut le. A paraméter 0-tól 255-ig tetszőleges egész szám lehet.

CLR

Token: \$9C #156
Belépési pont: \$A65E

Parancs; rövidítése: CL

Szintaxis: CLR

Inicializálja a változóterületet. Alkalmazása után nem férhetünk hozzá előzőleg használt változóinkhoz.

CMD

Token: \$9D #157
Belépési pont: \$AA86

Parancs; rövidítése: CM

Szintaxis: CMD *logikai file-szám*, kifejezés

Példa: CMD 2 : CMD A : CMD 2"XYZ"

Az elsődleges output eszköz kijelölése. A paraméterben megadott logikai file-szám jelöli ki az egységet, amelyre az output irányul. A kijelölés áttételes, mert a kiviteli eszközt valójában az OPEN-ben megadott egységszám jelöli ki. A CMD a PRINT# helyett is használható. A CMD hatását a PRINT# utasítás szünteti meg. A CMD nyomtatáskor lehet hasznos.

CONT

Token: \$9A #154
Belépési pont: \$A857

Parancs; rövidítése: CO

Szintaxis: CONT

A STOP utasítással vagy a RUN/STOP billentyűvel megszakított BASIC programot a megszakítás helyétől folytatja. Különösen tesztelésnél hasznos.

COS

Token: \$BE #190
Belépési pont: \$E264

Aritmetikai függvény; nem rövidíthető

Szintaxis: COS (*num. kifejezés*)

Példa: COS(π) : COS($A*2 + \pi/2$)

A függvény kiszámítja az argumentum koszinuszát. Az argumentumot radiánban kell megadni.

DATA	Token:	\$83 #131
	Belépési pont:	\$A8F8

Utasítás; rövidítése: DA

Szintaxis: DATA *adatok*

Példa: DATA 1,2,3,... : DATA "A","B","C"

Adatok elkülönítésére szolgál; segítségével a program önmagában is tárolhat adatokat. Az adatokat vesszővel kell elválasztani, típusuk lehet szöveges vagy numerikus, illetve ezek keveréke. Parancsmódban nem adható ki. Előfordulhat, hogy az adat vesszőt vagy pontosvesszőt tartalmaz. Hogy a READ utasítás ne elválasztójelként értelmezze, az ilyen adatot feltétlenül tegyük idézőjelek közé.

DEF	Token:	\$96 #150
	Belépési pont:	\$B3B3

Utasítás; rövidítése: DE

Szintaxis: DEF FN...

Példa: DEF FN F(x) = $x \uparrow 2$

Felhasználói függvény definiálása. Parancsmódban nem adható ki.

DIM	Token:	\$86 #134
	Belépési pont:	\$B081

Utasítás; rövidítése: DI

Szintaxis: DIM *változólista*

Példa: DIM A(20) : DIM A(20),B(20),C\$(10,10)

Tömbváltozók deklarálása. Egyaránt alkalmazható vektorok és mátrixok dimenzionálására. Az utasítás hatására az interpreter helyet foglal a tömbváltozó elemei számára. A változó indexeinek számát csak a rendelkezésre álló hely korlátozza, ha az kevesebb 255-nél. Ez azonban csak elméleti korlát, a gyakorlatban sohasem érhetjük el. A tömb indexei mindig a nulla értéktől indulnak.

END	Token:	\$80 #128
	Belépési pont:	\$A831

Utasítás; rövidítése: EN

Szintaxis: END

Megszünteti a programvégrehajtást. Parancsmódban nem alkalmazható.

EXP	Token:	\$BD #189
	Belépési pont:	\$BFED

Aritmetikai függvény; rövidítése: EX

Szintaxis: EXP (*num. kifejezés*)

Példa: EXP(0) : EXP(A) : EXP(A + C)

Matematikai függvény, az Euler-féle szám ($e = 2,718281828$) hatványozása. A hatványkitevő a megadott argumentum.

FN	Token:	\$A5 #165
	Belépési pont:	\$B3F4

Felhasználói függvény

Szintaxis: FN *függvényazonosító (változó)*

Példa: A = FN F(x)

A DEF FN utasítással definiált függvény értékét számítja ki az argumentum által megadott pontban. Az utasítás parancsmódban is működik, ha a függvényt a program már előzőleg definiálta. Ellenkező esetben ?UNDEF'D FUNCTION ERROR hibaüzenetet kapunk.

FOR	Token:	\$81 #129
	Belépési pont:	\$A742

Utasítás; rövidítése: FO

Szintaxis: FOR *változónév* = *kezdőérték* TO *végérték*

Példa: FOR I= 1 TO 100

Ciklusszervező utasítás, a ciklusfejet definiálja. Parancs- és programmódban egyaránt használható. Az utasítás után csak valós típusú változó állhat, ellenkező esetben hibaüzenetet (SYNTAX ERROR) kapunk.

FRE	Token:	\$B8 #184
	Belépési pont:	\$B37D

Függvény; rövidítése FR

Szintaxis: FRE (*num. kifejezés*)

Példa: FRE(0) : FRE(X)

Megadja a szabad tárterületet byte-okban. Az argumentumnak az eredményre nincs befolyása. A FRE másik értelmezése szerint tárhelyet szabadít fel azzal, hogy a szöveges változókat összefüggő területre tömöríti.

GET

Token: \$A1 #161
 Belépési pont: \$AB7B

Utasítás; rövidítése: GE

Szintaxis: GET *változó*

Példa: GET X\$: GET A

a) Egy byte beolvasása a billentyűzetről; a byte a megadott változóba kerül. Parancsmódban nem használható. Ha a billentyűzetről nem érkezett adat, a változó értéke üres szöveg (" ") vagy nulla lesz.

b) GET#

Szintaxis: GET# *logikai file-szám, változó*

Példa: GET#2, A\$

Rövidítés: GE#

Egy byte beolvasása a lemezegységről vagy kazettás egységről. Működése hasonló a GET-hez. Ha a háttértárolóról nem érkezik adat, a változó értéke hasonlóan alakul, mint fent. Egy eltérés mégis van: ha a háttértároló CHR\$(0) adatot küld (szöveges mód), a változó értéke üres szöveg lesz (" "). Ez hibaüzenetet okozhat a programban, ha az esetleg nincs felkészítve üres szövegekre. Ilyenkor a beolvasott változóhoz mindig adjunk hozzá egy CHR\$(0) karaktert: A\$=A\$+CHR\$(0). Így annak hossza nem lehet nulla.

GO

Token: \$CB #203
 Belépési pont: \$A80E

Utasítás

Szintaxis: GO TO *sorszám*

Példa: GO TO 100

A TO kulcsszóval összekapcsolva feltétel nélküli ugrást hajt végre a paraméterben megadott sorra. A sorszám nem helyettesíthető változóval.

GOSUB

Token: \$8D #141
 Belépési pont: \$A883

Utasítás; rövidítése: GOS

Szintaxis: GOSUB *sorszám*

Példa: GOSUB 2000

Szubrutinhívás. A sorszám nem helyettesíthető változóval vagy kifejezéssel.

GOTO

Token: \$89 #137
 Belépési pont: \$A8A0

Utasítás; rövidítése: GO

Szintaxis: GOTO *sorszám*

Példa: GOTO 65

Feltétel nélküli ugrás a megadott sorra. A sorszám nem helyettesíthető változóval vagy kifejezéssel.

IF	Token:	\$8B #139
	Belépési pont:	\$A928

Utasítás

Szintaxis: IF *feltétel* THEN *következmény*

Példa: IF A < 100 THEN B = 20 : IF A < 100 THEN GOTO 200

Feltételes elágazás. A feltétel teljesülése esetén a THEN mögött álló utasítás(ok) végrehajtása következik. Ha a feltétel nem teljesül, a vezérlés a következő sorra adódik. A feltétel teljesülésének az tekinthető, ha a kifejezés értéke -1 . A feltétel lehet szöveges vagy numerikus kifejezés is. A THEN mögött álló GOTO elhagyható, például: IF A < 100 THEN 200. De ugróutasítás esetén a THEN felcserélhető a GOTO-val, például: IF A < 100 GOTO 200. Mindkét verzió működik. Az utasítás parancsmódban is használható, de nem sok értelme van úgy alkalmazni.

INPUT	Token:	\$85 #133
	Belépési pont:	\$ABBF

Utasítás

Szintaxis: INPUT *változólista*

Példa: INPUT A : INPUT A,B,C : INPUT "ADAT:"; A\$

Adatbeolvasás billentyűzetről. A maximális beolvasható szöveghossz 88 karakter. Ha a megjelenő kérdőjel mögött a vártnál több adatot adunk meg, az INPUT az ?EXTRA IGNORED üzenetet küldi. ?REDO FROM START üzenet jelenik meg, ha az INPUT több adatot vár, mint amennyit megadtunk. Az adatmezőket vesszővel választhatjuk el egymástól. Az INPUT és ezáltal a program könnyen zavarba hozható amiatt, hogy az INPUT nem tiltja a kurzor le-föl mozgatását. Az INPUT a beolvasandó változó értékét nem változtatja meg, ha csak RETURN-t nyomunk válaszként. Az INPUT és a változók közötti szöveg a képernyőre kerül, a beolvasás mögötte kezdődik.

INPUT#	Token:	\$84 #132
	Belépési pont:	\$ABA5

Utasítás; rövidítése: IN

Szintaxis: INPUT# *logikai file-szám, változólista*

Példa: INPUT#2, A\$; INPUT#3, A,B,C : INPUT#C,A\$,B\$

Adatbeolvasás perifériáról. Mivel a BASIC interpreter nem tartalmaz parancsot a beviteli egység kijelölésére, ezért a megfelelő periférián logikai file-t kell nyitni. A maximálisan beolvasható karakterlánc hossza 88 karakter.

INT	Token:	\$B5 #181
	Belépési pont:	\$BCCC

Aritmetikai függvény

Szintaxis: INT (*num. kifejezés*)

Példa: INT (A) : INT (A * 100)/100 : INT(A + B + 0.5)

Matematikai egészrész függvény. Az argumentum értékének egész részét képezi.

LEFT\$	Token:	\$C8 #200
	Belépési pont:	\$B700

Szöveges függvény; rövidítése: LEF

Szintaxis: LEFT\$ (*szöv. kifejezés, num. kif.*)

Példa: A\$ = LEFT\$(X\$,2) : X\$ = LEFT\$(A\$,A)

Egy karakterlánc bal oldalának meghatározott részét adja. A rész hosszát a kifejezés második paramétere adja meg. Az első paraméter a forrásszöveg azonosítója.

LEN	Token:	\$C3 #195
	Belépési pont:	\$B77C

Szöveges függvény

Szintaxis: LEN (*szöveges kifejezés*)

Példa: A = LEN(A\$) : A = LEN (A\$ + B\$)

Megadja a kifejezésben szereplő szöveges változó hosszát.

LET	Token:	\$88 #136
	Belépési pont:	\$A9A5

Utasítás; rövidítése: LE

Szintaxis: LET *értékadó kifejezés*

Példa: LET A = 20 : LET A = B

Változók értékadása. A Commodore BASIC nem követeli meg használatát, ezért értékadásnál el is lehet hagyni.

LIST	Token:	\$9B #155
	Belépési pont:	\$A69C

Parancs; rövidítése: LI

Szintaxis: LIST vagy LIST *sorszám*

Példa: LIST : LIST 20 : LIST 20 – : LIST – 20 : LIST 20 – 40

A paraméterben megadott sorszámú programsorokat listázza. A paraméter elmaradása esetén az egész programot kilistázza. Bár programba is beépíthető, alkalmazása esetén a program megszakad.

LOAD	Token:	\$93 #147
	Belépési pont:	\$E168

Parancs; rövidítése: LO

Szintaxis: LOAD *programnév, egységszám, másodlagos cím*

Példa: LOAD "ABC",1 : LOAD"XYZ",8,1

Program betöltése háttértárolóról. A névben kerülni kell a kettőspont, a pontosvessző és a vessző alkalmazását. A másodlagos cím, amely 0 vagy 1 — 255 lehet, határozza meg a töltés kezdőcímét. A másodlagos cím elhagyása esetén a betöltés mindig a BASIC-tár elejére történik. Ha megadunk valamilyen másodlagos címet, a program mindig ugyanoda kerül, ahonnan kimentettük. (A másodlagos címmel részletesebben is foglalkozunk a háttértárolóknál.)

LOG	Token:	\$BC #188
	Belépési pont:	\$B9EA

Aritmetikai függvény

Szintaxis: LOG (*num. kifejezés*)

Példa: LOG(1) : LOG(A) : LOG(A + B + C)

A természetes (Euler-) alapú logaritmus értékét számítja ki. Az argumentum nem lehet negatív szám.

MID\$	Token:	\$CA #202
	Belépési pont:	\$B737

Szöveges függvény; rövidítése: MI

Szintaxis: MID\$ (*szövegkifejezés, pozíció, hossz*)

Példa: MID\$(A\$,2) : MID\$(A\$,2,4)

A szövegfüggvény a megadott karakterláncból mintegy kiemeli a megadott hosszúságú részletet. A paramétereknek a lehetőségek (a szöveg hossza, a pozíció és a hossz viszonya) határain belül kell maradniuk. Ellenkező esetben ?ILLEGAL QUANTITY ERROR hiba-üzenetet kapunk.

NEW

Token: \$A2 #162
 Belépési pont: \$A642

Parancs

Szintaxis: NEW

Törli a tárban levő BASIC program mutatóit, inicializálja a változóterületet, és végrehajt egy melegstartot. A tárban levő program megmenthető, de ez csak a mutatók helyreállítása és a program újraláncolása után lehetséges. A legtöbb BASIC-bővítés tartalmaz ilyen parancsot. A program visszaállítása csak akkor lehetséges, ha a NEW kiadása után nem végeztünk semmi olyan műveletet, amely igénybe veszi a tárat. (A törölt program visszaállítására bemutattunk egy gépi nyelvű programot a BASIC és a gépi nyelv együttes használatáról szóló fejezetben.)

NEXT

Token: \$82 #130
 Belépési pont: \$AD1E

Utasítás; rövidítése: NE

Szintaxis: NEXT *ciklusváltozók*

Példa: NEXT : NEXT I : NEXT I,J,K

A FOR-ral megkezdett ciklusok lezáró utasítása. Feladata annak megvizsgálása, hogy a ciklusváltozó elérte vagy meghaladta-e már a kijelölt végértéket. Ha még nem haladta meg, a programvezérlést visszairányítja a ciklusmag elejére. Ez mindaddig ismétlődik, amíg a változó nem éri el a végértéket. A ciklusváltozót nem kötelező kitenni a kulcsszó mögé, ilyenkor mindig az utoljára megkezdett ciklus lép tovább. Ha több változót sorolunk fel, az egymásba skatulyázott ciklusokat jelent. A ciklus lefutása után a ciklusváltozó a lehetséges maximumnál eggyel nagyobb értéket vesz fel.

NOT

Token: \$A8 #168
 Belépési pont: \$AED4

Logikai-aritmetikai művelet; rövidítése: NO

Szintaxis: NOT *numerikus kifejezés*

Példa: NOT 1+2 : NOT (A*B) : NOT A

Az operandus értékének logikai komplementjét állítja elő. Az operandus legnagyobb értéke 32 767, legkisebb értéke -32 768 lehet. A NOT X eredménye a következő eljárás alapján számítható ki:

$$\text{NOT } X = -(X+1)$$

ON	Token:	\$91 #145
	Belépési pont:	\$A94B

Utasítás

Szintaxis: ON *változó* GOTO *sorszámok*

Példa: ON A GOTO 100,200,300 : ON A GOSUB 100,200,300

Feltételhez kötött vezérlésátadó utasítás. A megadott paraméter egész részének megfelelő sorszámú sorra adódik a vezérlés, akár GOTO, akár GOSUB útján. A példa szerinti működés: ha az A értéke 1, akkor a 100-as sorra, ha 2, a 200-as sorra, ha 3, akkor a 300-as sorra kerül a végrehajtás. Ha A értéke 4 vagy nagyobb, ?UNDEF'D STATEMENT ERROR hibaüzenetet kapunk.

OPEN	Token:	\$9F #159
	Belépési pont:	\$E1BE

Utasítás; rövidítése: OP

Szintaxis: OPEN *logikai file-szám, egységszám, másodlagos cím, utasítás*

Példa: OPEN 4,4 : OPEN 2,8,2,"ABC,S,W"

Megnyitja a logikai file-t a megadott számú egységen. Ha utasítást (nevet) nem alkalmazunk, a másodlagos cím sem kötelező. A másodlagos címet egyes nyomtatók igénylik, pl. MPS 802. A másodlagos cím ilyenkor formátumutasítások küldését készítheti elő, vagy a nyomtató üzemmódját változtathatja meg.

A megadható egységszámok:

- 1 — kazettás egység
- 2 — RS232
- 3 — képernyő
- 4 — nyomtató #1
- 5 — nyomtató #2
- 8 — lemezegység #1
- 9 — lemezegység #2
- 10 — lemezegység #3
- 11 — lemezegység #4

OR	Token:	\$B0 #176
	Belépési pont:	\$AFE6

Logikai-aritmetikai művelet

Szintaxis: *numerikus kifejezés* OR *numerikus kifejezés*

Példa: A OR B : 120 OR 11

Logikai VAGY művelet, amely a két operandus között jön létre. Az operandusok értékének a -32768 és 32767 tartományba kell esnie.

Az OR művelet értelmezése:

0 OR 0 = 0

0 OR 1 = 1

1 OR 0 = 1

1 OR 1 = 1

PEEK

Token: \$C2 #194

Belépési pont: \$B80D

Függvény; rövidítése: PE

Szintaxis: PEEK (*tárcím*)

Példa: PEEK (32768) : PEEK (X)

A függvény megadja az argumentumnak megfelelő tárcím tartalmát. Emiatt az argumentum nem lehet nagyobb 65535-nél.

POKE

Token: \$97 #151

Belépési pont: \$B824

Utasítás; rövidítése: PO

Szintaxis: POKE *tárcím*, *érték*

Példa: POKE 49152,123 : POKE A,10 : POKE 2049+1,A :

A RAM byte-jainak megváltoztatására szolgál. Az érték mindig a RAM-ba kerül, kivéve az I/O területeket.

POS

Token: \$B9 #185

Belépési pont: \$B39E

Függvény

Szintaxis: POS (*numerikus kifejezés*)

Példa: POS(0) : POS(2) : POS(X)

Visszaadja azt az oszloppozíciót, amelyben a kurzor utoljára tartózkodott. Csak programban van értelme alkalmazni, mert parancsmódban mindig nullát ad. A kifejezés értéke nem meghatározó, bármi lehet 0 — 255 között.

PRINT

Token: \$99 #153

Belépési pont: \$AAA0

Utasítás; rövidítése: ?

Szintaxis: PRINT *kifejezés*

Példa: PRINT A : PRINT "ABC" : PRINT A\$

Az adatforgalom, a nyomtatás, a megjelenítés átfogó utasítása. A mögötte álló kifejezést, illetve annak értékét, eredményét az éppen aktuális output egységre továbbítja. A PRINT

utasítással számos más feladat is megoldható; ehhez a vezérlőkérekek nyújtanak segítséget.

Speciális funkciók:

- \$08 : a karakterkészlet váltásának blokkolása
- \$09 : a karakterkészlet váltásának engedélyezése
- \$0E : kisbetű/nagybetű készlet beállítása
- \$11 : kurzor le
- \$12 : az inverz mód bekapcsolása
- \$13 : a kurzor a bal felső sarokba (Home)
- \$14 : egy karakter törlése (Delete)
- \$1D : kurzor jobbra
- \$8E : nagybetű/grafika készlet
- \$91 : kurzor fel
- \$92 : az inverz mód kikapcsolása
- \$93 : a képernyő törlése
- \$94 : egy karakter beszúrása (Insert)
- \$9D : kurzor balra

A színek beállítása:

- \$05 : fehér (white)
- \$1C : vörös (red)
- \$1E : zöld (green)
- \$1F : kék (blue)
- \$81 : narancs (orange)
- \$90 : fekete (black)
- \$95 : barna (brown)
- \$96 : piros (light red)
- \$97 : szürke1 (grey1)
- \$98 : szürke2 (grey2)
- \$99 : világoszöld (light green)
- \$9A : világoskék (light blue)
- \$9B : szürke3 (grey3)
- \$9C : lila (purple)
- \$9E : sárga (yellow)
- \$9F : türkiz (cyan)

A vezérlőkérekek pl. a CHR\$ függvény segítségével aktivizálhatók.

PRINT#

Token: \$98 #152
 Belépési pont: \$AA80

Utasítás; rövidítése: PR

Szintaxis: PRINT# *logikai file-szám, kifejezés*

Példa: PRINT#2,A\$: PRINT#4, 128 2-3

Funkciója azonos a PRINT utasításéval, a különbség közöttük az irányítottságban van. A PRINT# a megadott (és előzetesen megnyitott) logikai file-ba továbbítja a kiírandó kifejezést.

READ

Token: \$87 #135
 Belépési pont: \$AC06

Utasítás; rövidítése: RE

Szintaxis: READ *változó(kifejezés)(ek)*

Példa: READ X : READ A,B,C : READ A\$

Segítségével lehet a programban elhelyezett adatokat átolvasni változóba. Mindig a sorra következő DATA elemet olvassa be; amennyiben azok elfogytak, ?OUT OF DATA ERROR hibaüzenet jelenik meg.

REM

Token: \$8F #143
 Belépési pont: \$A93B

Utasítás

Szintaxis: REM *megjegyzés*

Példa: REM FOPROGRAM

Az utasítás megjegyzések programba építésére alkalmas, mivel az interpreter az utasítást követő szöveget nem dolgozza föl. A REM a sorban bárhol állhat, de az utána álló jeleket az interpreter a programsor végéig megjegyzésként értelmezi.

RESTORE

Token: \$8C #140
 Belépési pont: \$A81D

Utasítás; rövidítése: RES

Szintaxis: RESTORE

Utasítja az interpretert, hogy a DATA mutatót állítsa a program elejére — alaphelyzetbe — ez \$0800. A DATA mutató a nulláslap \$41 — 42 címén található.

RETURN	Token:	\$8E #142
	Belépési pont:	\$A8D2

Utasítás; rövidítése: RET

Szintaxis: RETURN

A szubrutin befejező utasítása. Hatására az interpreter a hívó utasítást követő utasításon folytatja a program végrehajtását.

RIGHT\$	Token:	\$C9 #201
	Belépési pont:	\$B72C

Szöveges függvény; rövidítése: RI

Szintaxis: RIGHT\$ (*szöv. kifejezés, hossz*)

Példa: RIGHT\$ (A\$,4) : \$ (A\$ + B\$,C)

A kijelölt szöveges változó jobb oldaláról a megadott hosszúságú karakterláncot elkülöníti. Az interpreter a továbbiakban ezzel dolgozhat.

RND	Token:	\$BB #187
	Belépési pont:	\$E097

Függvény; rövidítése: RN

Szintaxis: RND (*változó*)

Példa: RND (0) : RND (5) : RND (-1)

Pszudovéletlen számgenerátor. Ha az argumentum negatív, kicseréli a számításhoz szükséges állandót az argumentumból számított értékkel. Ha pozitív, értéke nem meghatározó. A számított érték egy 0 és 1 közé eső, valós típusú szám.

RUN	Token:	\$8A #138
	Belépési pont:	\$A871

Parancs; rövidítése: RU

Szintaxis: RUN *sorszám*

Példa: RUN : RUN 100

A BASIC program elindítása. Arra utasítja az interpretert, hogy a megadott sortól kezdve hajtsa végre a programot. Argumentumként változó vagy kifejezés nem alkalmazható. Az utasítás programból is kiadható, így érhető el például az újraindítás.

SAVE

Token: \$94 #148
 Belépési pont: \$E156

Parancs; rövidítése: SA

Szintaxis: SAVE *név, egységszám*

Példa: SAVE "ABC",1 : SAVE "XYZ",8

Az utasítás a BASIC programok háttértárolóra mentésére szolgál.

SGN

Token: \$B4 #180
 Belépési pont: \$BC39

Aritmetikai függvény; rövidítése: SG

Szintaxis: SGN (*num. kifejezés*)

Példa: SGN (-3) : SGN (A) : SGN (A*8)

Matematikai előjelfüggvény. Ha az argumentum negatív, értéke -1, ha pozitív, +1, ha nulla, akkor 0.

SIN

Token: \$BF #191
 Belépési pont: \$E26B

Aritmetikai függvény; rövidítése: SI

Szintaxis: SIN (*num. kifejezés*)

Példa: SIN ($\pi/2$) : SIN (200)

Matematikai szinusz függvény. A szöget radiánban kell megadni.

SPC(

Token: \$A6 #166
 Belépési pont: \$AAF8

Utasítás; rövidítése: SP

Szintaxis: SPC (*num. kifejezés*)

Példa: SPC (6) : SPC (A-8)

Tabulációs célokat szolgál a PRINT utasításnál. A kurzort a kifejezés értékének megfelelően jobbra mozgatja.

SQR

Token: \$BA #186
 Belépési pont: \$BF71

Aritmetikai függvény; rövidítése: SQ

Szintaxis: SQR (*num. kifejezés*)

Példa: SQR (2) : SQR (A) : SQR (A+B+C)

A kifejezés négyzetgyökét adja. Negatív számot nem fogad el.

STEP	Token: \$A9	#169
	Belépési pont:	

Utasítás; rövidítése: STE

Szintaxis: STEP *num. kifejezés*

Példa: FOR 1 = 1 TO 100 STEP 0.04 : FOR 1 = 100 TO 1 STEP -1

Az utasítás önmagában nem alkalmazható, csak a FOR utasítással kombinálva. A ciklus-változó léptetési ütemét (és irányát) adja meg.

STOP	Token:	\$90 #144
	Belépési pont:	\$A82F

Utasítás; rövidítése: ST

Szintaxis: STOP

Csak programban alkalmazható: megszakítja a program futását. A megállított program CONT-tal folytatható.

STR\$	Token:	\$C4 #196
	Belépési pont:	\$B465

Szöveges függvény; rövidítése: STR

Szintaxis: STR\$ (*num. kifejezés*)

Példa: STR\$ (5) : STR\$ (A)

Numerikus kifejezés eredményét alakítja át szöveges formátumra.

SYS	Token:	\$9E #158
	Belépési pont:	\$E12A

Utasítás; rövidítése: SY

Szintaxis: SYS *tárcím*

Példa: SYS 49152 : SYS A

Gépi program hívása BASIC-ből. Az argumentum értéke 65536-nál kisebb pozitív egész szám lehet. A 3. lapon (\$030C—0F) található négy tárhely segítségével a gépi rutin indítása előtt beállíthatjuk a processzor munkaregisztereit és az állapotregisztert. A szubrutin lefutása utáni állapotot szintén ezekben a regiszterekben találhatjuk meg.

TAB(Token:	\$A3 #163
	Belépési pont:	\$AAF8

Utasítás; rövidítése: TA

Szintaxis: TAB (*num. kifejezés*)

Példa: TAB (10) : PRINT TAB (A-6)

Tabulációs célokat szolgál, beállítja a kurzorpozíciót. A PRINT utasítás része, önmagában nem alkalmazható. Működésének eredménye hasonló az SPC(utasítás hatásához, de a pozíciót más módon számítja ki. Míg az SPC(mindig az utolsó kurzorpozíciótól számolja a kiírandó karakterek kezdőpozícióját, a TAB(a sor (a fizikai képernyősor) kezdőpozíciójához viszonyít. Például a PRINT "AB"; TAB(10); "CD" hatására az AB szöveg a sor elejére kerül, a CD szöveg viszont nem a 12. pozíciótól kezdődik, mint az SPC(esetén lenne, hanem a 10-től. Ha az AB szöveget egy tíz karakternél hosszabb karakterlánccal helyettesítjük, a CD a kiírt szöveg mögé kerül, és nem írja felül azt.

A TAB paramétere mindig a képernyősor kezdetéhez viszonyítva értelmezhető, az utána következő karakterek ettől a pozíciótól számítva jelennek meg, kivéve, ha korábban már írtunk a sorba a megadott paraméternél hosszabb szöveget. Ekkor közvetlenül mögötte folytatódik a kiírás.

TAN

Token: \$C0 #192
Belépési pont: \$E2B4

Aritmetikai függvény

Szintaxis: TAN (*num. kifejezés*)

Példa: TAN (10.2) : TAN (A)

Kiszámítja az argumentum tangensét. A szöveget radiánban kell megadni.

THEN

Token: \$A7 #167
Belépési pont: –

Utasítás; rövidítése: TH

Szintaxis: IF *feltétel* THEN *utasítás*

Példa: IF A = 100 THEN A = 10 : IF A < 10 THEN 120

Önmagában nem alkalmazható, csak az IF utasítással együtt. Jelentése szerint a mögötte álló utasítás(oka)t akkor hajtja végre a program, ha az előtte álló feltétel teljesült, igaz. Bizonyos esetekben a kulcsszó GOTO-val helyettesíthető.

TO

Token: \$A4 #164
Belépési pont: –

Utasítás

Szintaxis: TO

Önmagában nincs értelme. Csak a FOR és a GO utasításokkal együtt alkalmazható.

USR	Token:	\$B7 #183
	Belépési pont:	\$0310

Függvény; rövidítése: US

Szintaxis: USR (*num. kifejezés*) a zárójel használata kötelező

Példa: USR (5) : USR (X)

Felhasználói függvény, ugyanúgy működik, mint az aritmetikai függvények. Alkalmazásához gépi program szükséges, valamint a függvény ugrási vektorának (\$0311) beállítása.

VAL	Token:	\$C5 #197
	Belépési pont:	\$B7AD

Szöveges függvény; rövidítése: VA

Szintaxis: VAL (*szövegkifejezés*)

Példa: VAL ("123") : VAL (A\$)

A kifejezésben felírt szöveges típusú adat számértékét adja vissza. Ha nem számjeggyel, tizedesponnttal vagy negatív előjellel kezdődik, az eredmény mindig nulla.

VERIFY	Token:	\$95 #149
	Belépési pont:	\$E165

Utasítás; rövidítése: VE

Szintaxis: VERIFY *név, egységszám*

Példa: VERIFY "ABC", 8

Minden tekintetben azonos a LOAD utasítással, de a beolvasott adat-, illetve programbyte-okat nem tölti be a tárba, csupán összehasonlítja. Ha eltérnek, ?VERIFY ERROR üzenetet küld és leállítja az ellenőrzést. Egyezés esetén az OK felirat jelenik meg.

WAIT	Token:	\$92 #146
	Belépési pont:	\$B82D

Utasítás; rövidítése: WA

Szintaxis: WAIT *cím, paraméter₁, paraméter₂*

Példa: WAIT 198, 1 : WAIT 56576,1,255

Feladata mindaddig várakozni, míg a tárcím bitkombinációja egy várt értéket fel nem vesz. Működése: beolvassa a tárcím tartalmát és a paraméter₂ értékével kizáró VAGY (EXOR) kapcsolatot létesít. Ennek eredménye és az első paraméter között logikai ÉS kapcsolat létesül. Ha az eredmény nulla, a folyamat előlről kezdődik, egyébként az utasítás lefut. Ha nem adunk meg második paramétert, annak értéke automatikusan 0.

3.3. Az interpreter hibaüzenetei

Hibaüzenetek mind program-, mind parancsmódban keletkezhetnek. Programmódban a hibaüzenetet a hiba helyének megjelölése követi. Például ?SYNTAX ERROR IN 120. Ilyenkor a 120-as sorban szintaktikai hiba van. A hibaüzenet kiadása után a program nem folytatható a CONT utasítással. Az interpreter hibaüzenet után mindig parancsmódba tér vissza.

Hibaüzenet saját programból is generálható. A hivatkozás azt az értéket adja meg, amelyet az X regiszterbe töltve a \$A437 rutin használ ahhoz, hogy az adott hibaüzenetet írja ki. A hibaüzeneteket azonban közvetlenül is elérhetjük, ha a belépési pontra hivatkozunk a SYS utasításban. A VERIFY ez utóbbi alól kivétel, belépési pontját ezért tettük zárójelbe.

BAD SUBSCRIPT Belépési pont: \$B245 Hivatkozás: \$12

Az indexes változóban megadott index túllépi a definiáláskor megadott tömb (vektor vagy mátrix) méretét.

BREAK Belépési pont: \$E107 Hivatkozás: \$1E

A töltés közben STOP-pal megszakított LOAD-rutin írja ki. Tulajdonképpen nem hiba, hanem inkább üzenet.

CAN'T CONTINUE Belépési pont: \$A859 Hivatkozás: \$1A

A program nem folytatható. A CONT kiadása után érkezik.

DEVICE NOT PRESENT Belépési pont: \$F707 Hivatkozás: \$05

A file-nyitás során megcímezett egység nincs jelen a buszon.

DIVISION BY ZERO Belépési pont: \$BB8A Hivatkozás: \$14

Osztás nullával. A műveletben az osztó értéke nulla.

FILE DATA Belépési pont: \$AB66 Hivatkozás: \$18

A file-ből rossz adat érkezett, numerikus érték helyett szöveg.

FILE NOT FOUND Belépési pont: \$F704 Hivatkozás: \$04

Az olvasásra megnyitott file nem szerepel a háttértárolón, vagy rossz file-nevet adtunk meg.

FILE NOT OPEN Belépési pont: \$F701 Hivatkozás: \$03

Input-output utasításban olyan file-ra hivatkoztunk, amely még nincs megnyitva, vagy hibás a logikai file-szám.

- FILE OPEN** Belépési pont: \$F6FE Hivatkozás: \$02
A megnyitni kívánt logikai file már nyitott. Teendő: másik logikai file-számot kell választani.
- FORMULA TOO COMPLEX** Belépési pont: \$B4D0 Hivatkozás: \$19
A kifejezés túl összetett. A szöveges kifejezést a kiértékelő rutin nem tudja feldolgozni, ezért legalább két részre kell bontani.
- ILLEGAL DEVICE NUMBER** Belépési pont: \$F713 Hivatkozás: \$09
A soros buszon olyan egységet kíséreltünk meg elérni, amelyet az operációs rendszer nem tud kezelni.
- ILLEGAL DIRECT** Belépési pont: \$B3AB Hivatkozás: \$15
Olyan utasítást akartunk parancsmódban használni, amelyet az interpreter nem enged meg (GET, INPUT, DEF FN stb.).
- ILLEGAL QUANTITY** Belépési pont: \$B248 Hivatkozás: \$0E
Nem megengedhető praméter. A függvényargumentum túllépi a számaábrázolás lehetőségeit.
- LOAD** Belépési pont: \$E19C Hivatkozás: \$1D
A szalagon levő program hibás; a betöltést az operációs rendszer nem tudja végrehajtani.
- MISSING FILENAME** Belépési pont: \$F710 Hivatkozás: \$08
A file-név hiányzik az utasításból. Teendő: pótolni.
- NEXT WITHOUT FOR** Belépési pont: \$AD3A Hivatkozás: \$0A
A program futása közben az interpreter olyan NEXT utasítást talált, amelyhez nem tartozik FOR utasítás. Felül kell vizsgálni a program felépítését.
- NOT INPUT FILE** Belépési pont: \$F70A Hivatkozás: \$06
Olyan file-ból próbáltunk olvasni, amelyet előzőleg írásra nyitottunk meg.
- NOT OUTPUT FILE** Belépési pont: \$F70D Hivatkozás: \$07
Olyan file-ba próbáltunk írni, amelyet előzőleg olvasásra nyitottunk meg.
- OUT OF DATA** Belépési pont: \$ACBF Hivatkozás: \$0D
A READ utasítás nem talált megfelelő DATA utasítást. A DATA elemek száma kevesebb, mint a READ utasítások száma.

OUT OF MEMORY

Belépési pont: \$A435

Hivatkozás: \$10

A programtár megtelt. A definiálni kívánt változó számára nincs hely. A processzor visszatérési verme, amelyet az interpreter is használ, megtelt. Túl sok ciklust skatulyáztunk egymásba, vagy gépi program betöltése után nem adtunk NEW parancsot.

OVERFLOW

Belépési pont: \$B97E

Hivatkozás: \$0F

A szám túl nagy. A számítások során olyan szám keletkezne, amit az interpreter nem képes ábrázolni.

REDIM'D ARRAY

Belépési pont: \$B24D

Hivatkozás: \$13

A dimenzionálni kívánt tömböt (vektor vagy mátrix) már egyszer dimenzionáltuk. Különbösen autodimenzionálás után fordul elő.

RETURN WITHOUT GOSUB

Belépési pont: \$A8E0

Hivatkozás: \$0C

Az interpreter olyan szubrutin végére ért, amelyet nem szubrutinként hívott meg.

STRING TOO LONG

Belépési pont: \$A571

Hivatkozás: \$17

Az interpreter a szöveget nem képes kezelni, mert hosszabb, mint 255 karakter. Főleg szövegösszeadásnál fordul elő.

SYNTAX

Belépési pont: \$AF08

Hivatkozás: \$0B

Formai hiba. A leggyakoribb hibaüzenet. Teendő: ellenőrizni a szintaxist.

TOO MANY FILES

Belépési pont: \$F6FB

Hivatkozás: \$01

Tíznél több logikai file nem tartható nyitva. A tizenegyedik file megnyitásakor kapjuk ezt az üzenetet.

TYPE MISMATCH

Belépési pont: \$AD99

Hivatkozás: \$16

Típuskeveredés. Szövegfüggvényben numerikus argumentumot adtunk meg, vagy fordítva.

UNDEF'D FUNCTION

Belépési pont: \$B3AE

Hivatkozás: \$1B

Definiálatlan függvény. Az FN utasítás előzetesen nem definiált függvényre hivatkozik.

UNDEF'D STATEMENT

Belépési pont: \$A8E3

Hivatkozás: \$11

A GOTO, GOSUB vagy RUN utasítás után olyan sorszám áll, amelyhez nem tartozik programsor.

VERIFY

Belépési pont: (\$E17E)

Hivatkozás: \$1C

Az összehasonlított program nem azonos a háttértárolón levő file tartalmával.

Az INPUT utasítás üzenetei

REDO FROM START — numerikus adat helyett szöveges típusú érkezett. A beolvasást meg kell ismételni.

EXTRA IGNORED — több adatot adtunk meg (vesszővel elválasztva), mint amit az INPUT várt. A fölösleget a rutin figyelmen kívül hagyja.

Az interpreter egyéb üzenetei

OK — az interpreter a VERIFY utasítás hibamenetes lefutása után írja ki. Azt jelenti, hogy a két program azonos.

BREAK — a program futása STOP utasítás vagy a STOP billentyű hatására megszakadt. A program CONT-tal folytatható.

READY — készenléti állapot. A program futása vagy az utolsó művelet véget ért. Az interpreter várja a következő feladatot.

3.4. Tár- és változókezelés

A programozónak — különösen gépi programoknál — szüksége lehet a BASIC interpreter tár- és változókezelésének ismeretére, vagyis arra, hogy a BASIC interpreter milyen módon tárolja a RAM-ban a programot és a változókat. A következőkben ezt ismertetjük.

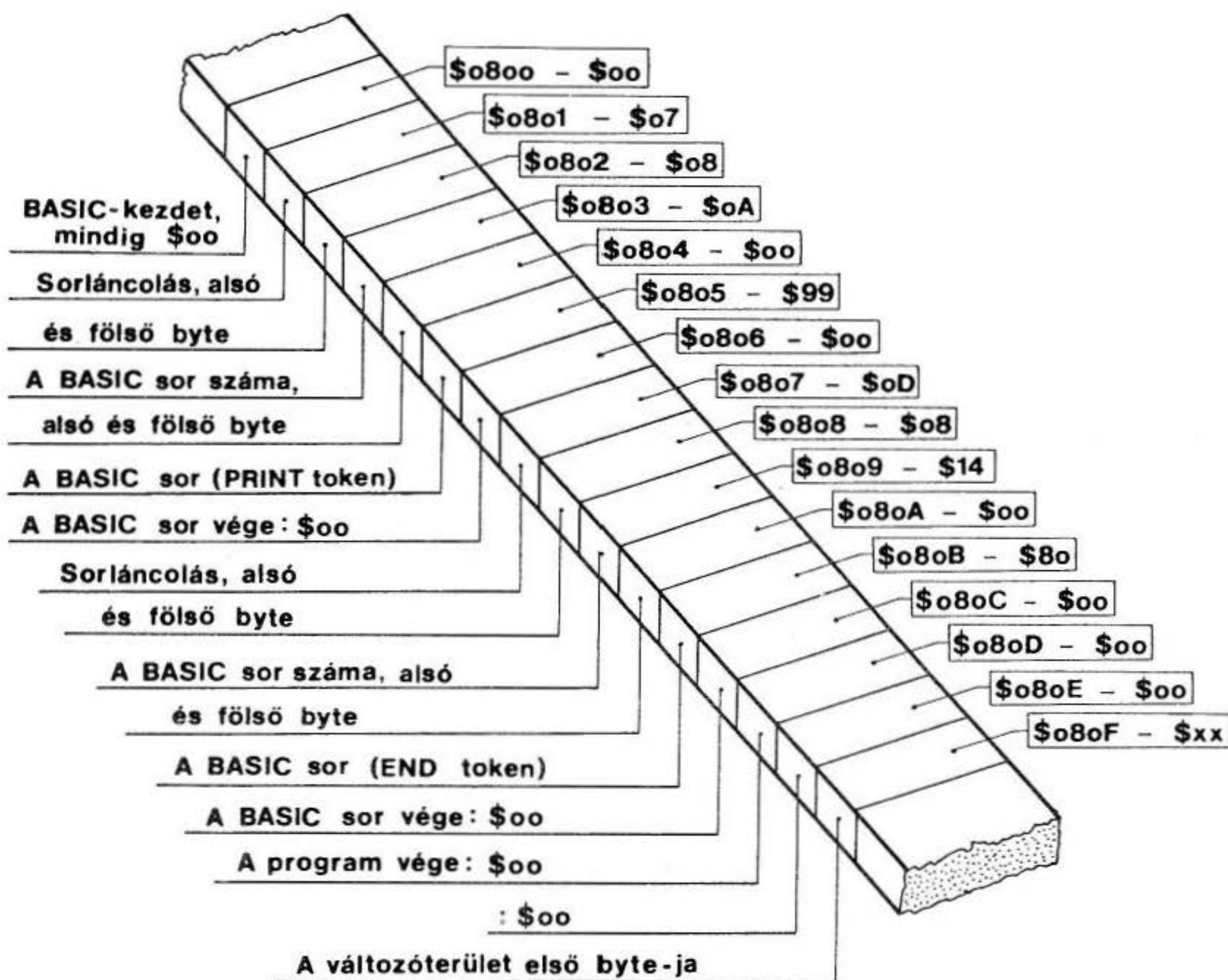
Programkezdet, programszerkezet, programvég

Az interpreter a BASIC-tár kezdetét a nulláslap \$2B — 2C regiszterpárján található mutató alapján számítja ki. A mutató bizonyos határok között szabadon változtatható. A BASIC programtár kezdetén mindig \$00 kell legyen, ellenkező esetben a RUN parancsra az interpreter ?SYNTAX ERROR üzenettel válaszol. A \$2B — 2C címeken levő mutató a \$00-t tartalmazó byte-ot követő tárhelyet címzi meg.

Nézzünk egy konkrét példát. Legyen programunk a következő:

```
10 PRINT  
20 END
```

A tárban a program a 25. ábrán látható módon helyezkedik el.



25. ábra. Példa a BASIC programok tárbeli elhelyezkedésére

A BASIC programsorok a tárban a következő elv szerint helyezkednek el.

1. A láncolási cím alsó byte-ja
2. A láncolási cím felső byte-ja
3. A programsor száma, alsó byte
4. A programsor száma, felső byte
5. A programsor (adatok, token stb.)
6. \$00 — az első sor vége
7. Láncolási cím, alsó byte
- ...
- ...
- x. \$00 — az utolsó sor vége
- x+1. \$00 — a láncolási cím alsó
- x+2. \$00 — a láncolási cím felső byte-ja, program vége

A programnak akkor van vége, ha a láncolási cím alsó és felső byte-ja is nulla. A program-sorok számozása 0-tól 63999-ig terjedhet. Egy program tehát maximálisan „csak” 64 000 sor-ból állhat, s nem 65 536-ból. Ennek oka, hogy az interpreter szerzői a sorszámellenőrzés mi-nél egyszerűbb módját kívánták beépíteni az alapszoftverbe, így csak a felső byte-ot ellenőrzi a program. Ha a felső byte-okat ellenőrzi az interpreter, az alsó byte-nak értelemszerűen \$00-nak kell lennie. A felső byte-nak a \$FA00-t választották, mert az kerek számot ad tízes számrendszerben, s így könnyen megjegyezhető.

A változók és a változóterületek

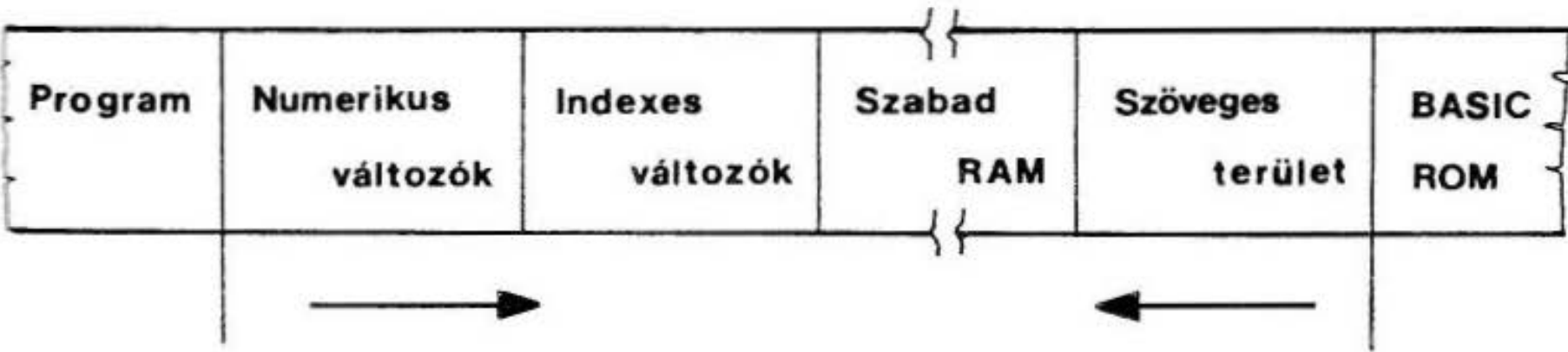
A Commodore BASIC három különböző változóterületet ismer. A tárban elfoglalt helyüket és sorrendjüket követve: numerikus változók, indexes változók és szöveges változók.

A numerikus változók közvetlenül a BASIC program után helyezkednek el. A terület kez-detének mutatója a nulláslapon a \$2D — 2E regiszterekben található. Az indexes változók a numerikus változóterület után következnek, a terület mutatóját a \$2F — 30 regiszterek tartal-ma adja. A numerikus és tömbváltozók az alacsonyabb tárcímtől a magasabb felé terjeszke-dnek, a szöveges változók pedig éppen ellenkezőleg. A szöveges terület eleje mindig a szabad BASIC RAM legfelső címe. A változók „hátrafelé” olvashatók. A változó tartalma továbbra is a megszokott sorrendet követi, de az a változó, amelyiket előbb deklaráltunk, magasabb címen kezdődik, mint az, amelyiket később.

A RAM tulajdonképpen olyan szendvics, amelyet kétfelől „esznek”: a numerikus változók balfelől, a szöveges változók jobbfelől. Meg kell azonban jegyezni, hogy míg a numerikus változók tartalma az azonosítójukat követi, addig a szöveges változók esetében ez nincs így. A szöveges változó azonosítója ugyanis a numerikus változóterületen helyezkedik el. A válto-zóterületeket az alábbi regiszterek határolják be:

- \$2D — 2E: az index nélküli változók területének kezdőcíme,
- \$2F — 30: az indexes változók területének kezdőcíme, az előbbi végcíme + 1,
- \$31 — 32: a szöveges terület végcíme, az indexes terület végcíme + 1,
- \$33 — 34: a szöveges terület kezdőcíme, a BASIC RAM vége.

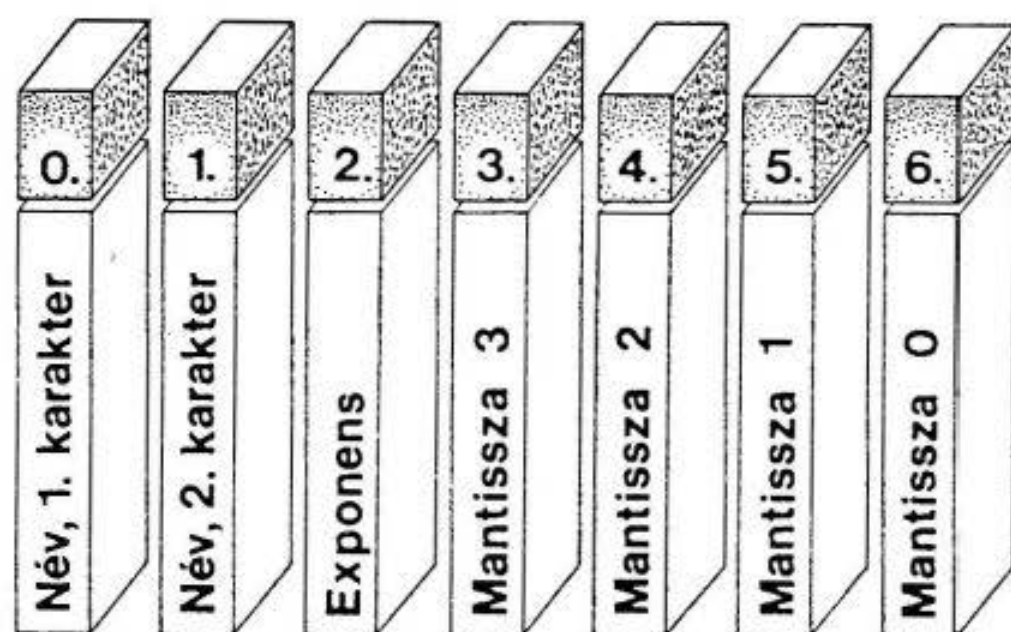
Grafikusan ábrázolva (a nyilak a terjeszkedés irányát jelzik):



A változók ábrázolása

Az *index nélküli* változók minden esetben egy *kétbyte-os blokkban* helyezkednek el. Erre a keresés egyszerűsítése miatt van szükség, mert pl. az egész típusú változó számára elegendő lenne 4 byte, vagy a szöveg számára 5 byte.

Valós változó



26. ábra. A valós típusú változó

A valós típusú változóknál lebegőpontos (floating-point) értéket tárol az interpreter. A lebegőpontos elnevezés a tizedespont elhelyezésére utal. Az ábrázolt szám 4 byte-ra és egy exponensre leképezett változatában ugyanis a tizedespontnak nincs kijelölt helye.

A számaábrázolás bináris rendszerben történik. A mantissza 32 bites bináris egész. Lebegőpontos valós típusúvá azáltal válik, hogy az exponens hatványkitevőként szerepel. A hatványkifejezés alapja pedig a mantissza. Az így keletkezett szám tizedes törtet is ábrázolhat. A lebegőpontos szám értékét a mantisszából és az exponensből az alábbi módon lehet kiszámítani:

$$Y = \text{mantissza} * 2^{\uparrow (-32 + \text{EXP} - 128)}$$

vagy

$$Y = \text{mantissza} * 2^{\uparrow (\text{EXP} - 160)}$$

A mantissza legfelső bitjét (a mantissza3 7. bitjét) a művelet előtt mindig magasra kell állítani ahhoz, hogy a művelet helyes eredményt adjon. Erre azért van szükség, mert a mantissza3 7. bitje tárolás közben a szám előjelét mutatja. Ha magas, a szám negatív. Amikor az interpreter beolvassa a változót a RAM-ből, ezt az előjelet egy kijelölt regiszterben külön tárolja; ezután az említett bitet magasra állítja.

Az exponens értékére a következők mérvadók:

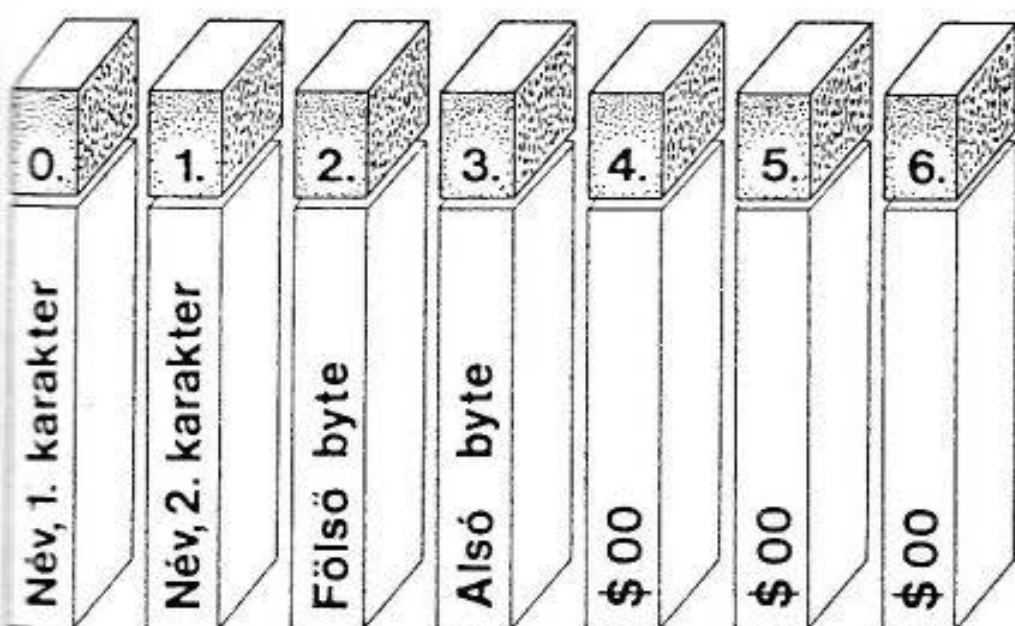
- ha kisebb \$81-nél, a szám abszolút értéke kisebb 1-nél;
- ha \$00, a szám értéke nulla, függetlenül attól, hogy mi a mantissza tartalma;
- ha nagyobb vagy egyenlő \$81-gyel, a szám abszolút értéke nagyobb vagy egyenlő 1-gyel.

Az interpreter a tárolt változót általában a lebegőpontos akkumulátorba (FAC — Floating Point Accumulator) olvassa be. Az előjelbit ilyenkor a \$66 regiszterbe (SGNFLG) kerül, a mantissza3 hetedik bitje pedig magasra vált.

Az említett ábrázolási korlátok (32 bites mantissza, egybyte-os exponens) miatt az ábrázolható szám a következő intervallumba eshet:

- ha pozitív előjelű: $2^{-128} - 2^{127}$
- ha negatív előjelű: $-2^{+127} - -2^{-128}$

Egész változó



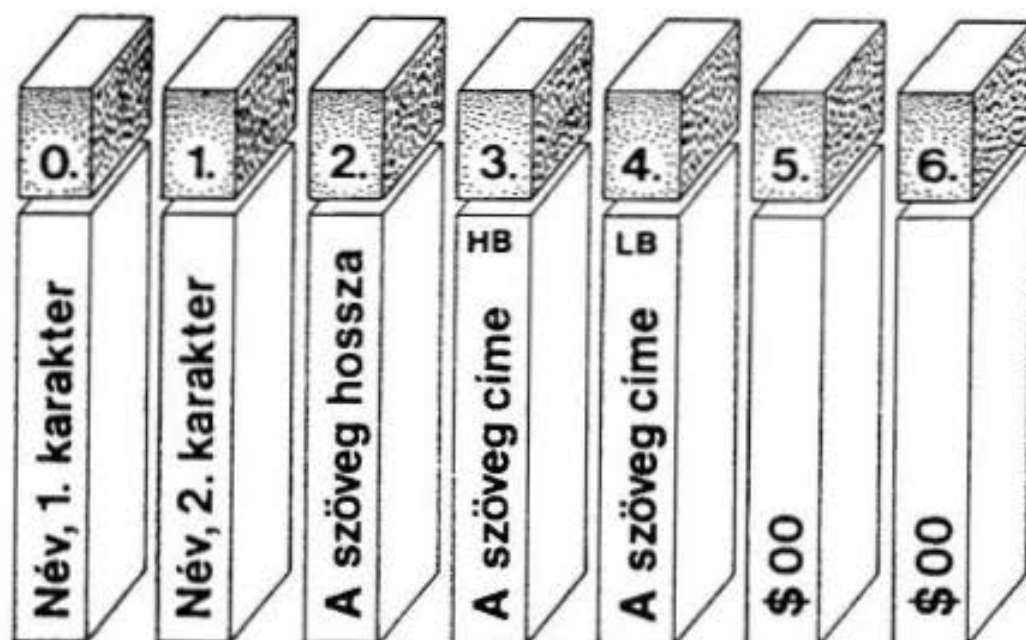
27. ábra. Az egész típusú változó

A tárolt szám 16 bites bináris egész. A szám legfelső bitje tárolja az előjelet, emiatt az ábrázolható legkisebb szám -32768 , a legnagyobb 32767 . Negatív előjel esetén (a felső byte 7. bitje magas) az ábrázolt szám abszolút értéke a tárolt szám kettes komplemente.

Szöveg változó

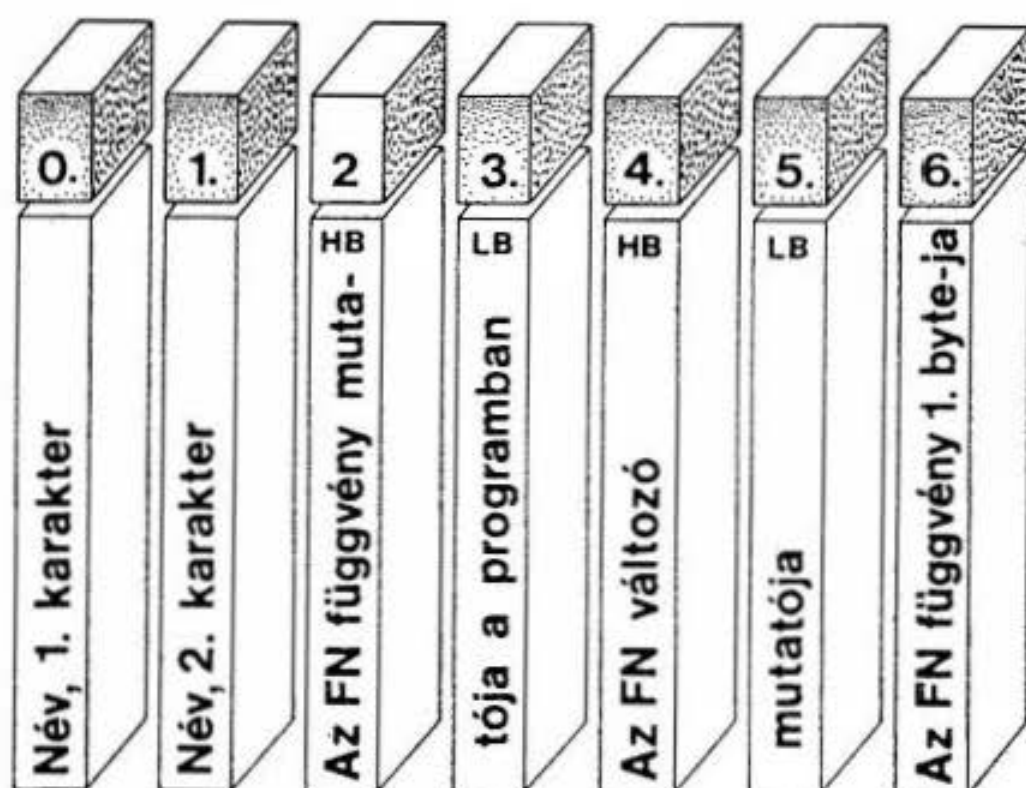
A szövegváltozó tartalma nem a fejlécet követően helyezkedik el, hanem a BASIC RAM végén, különálló területen.

A szöveg valódi címe mutathat a változóterületre és a programterületre egyaránt, attól függően, hogy milyen típusú értékadás történt. A szövegváltozó maximális hossza 255 (\$FF) karakter lehet.



28. ábra. A szöveg típusú változó

FN változó



29. ábra. Az FN változó

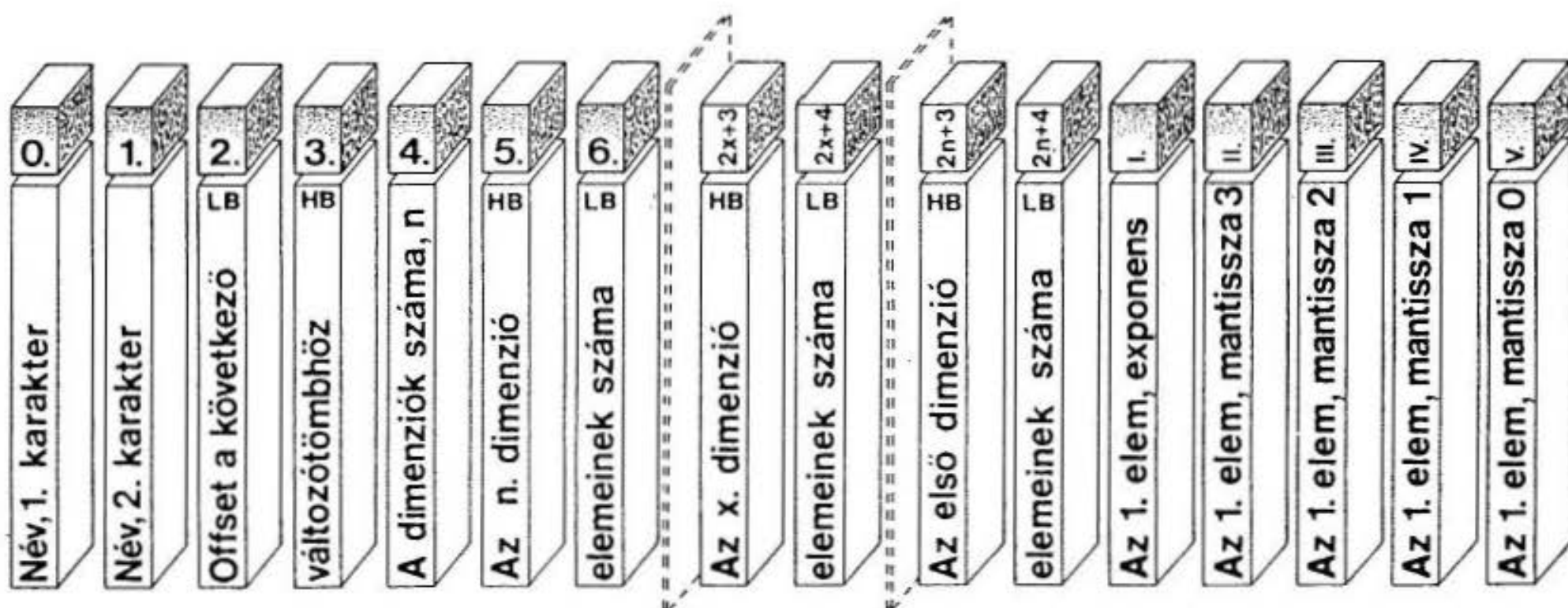
Az FN változó tartalmazza azt a mutatót, amely a függvény algoritmusára mutat a programterületen. Ez a mutató a 2. és 3. byte-on található. Az utána következő 3 byte a tárolás szempontjából semleges; szerepük az FN függvény feldolgozásakor van.

Az *indexes* vagy másképpen *tömbváltozók* olyan egyszerű változók, amelyek azonos változónévhez tartoznak, és megkülönböztetésük egy vagy több számparaméterrel, indexszel történik. Azokat a tömböket, amelyeket egy paraméter jellemez, vektoroknak, azokat pedig, amelyeket több paraméter határoz meg, mátrixoknak nevezzük.

Az indexek számozása mindig 0-tól kezdődik és a deklarációban (DIM) megadott határig terjed. Így egy DIM A (20) utasítás nem 20, hanem 21 elemű vektort jelöl ki. A többindexű mátrix elemeinek tárban elfoglalt sorrendje a következő szabály szerint alakul: az indexek közül mindig a balra álló változik gyorsabban. Eszerint egy 2x2-es mátrix elemei az alábbi sorrendben helyezkednek el:

$$A(0,0) : A(1,0) : A(0,1) : A(1,1).$$

Valós tömbök

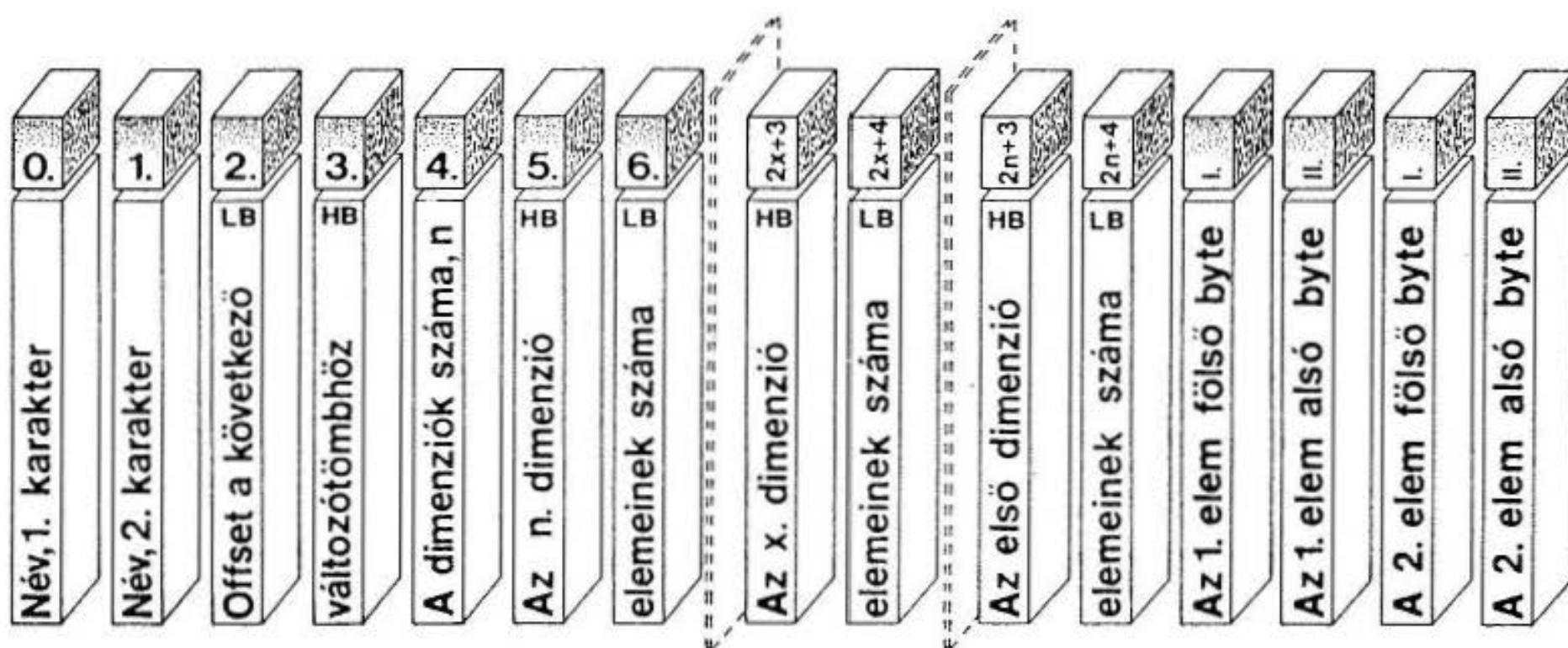


30. ábra. A valós típusú tömb

Ha a 0. byte címéhez hozzáadjuk a 2. és 3. byte tartalmát, megkapjuk a következő tömb kezdőcímét.

Egydimenziós tömb esetén a fejléc 7 byte hosszú, két dimenzió esetén 9 byte, n dimenzió $n*2+5$ byte hosszú. A fejléc után a változók következnek, amelyek egyenként 5 byte-ot foglalnak el.

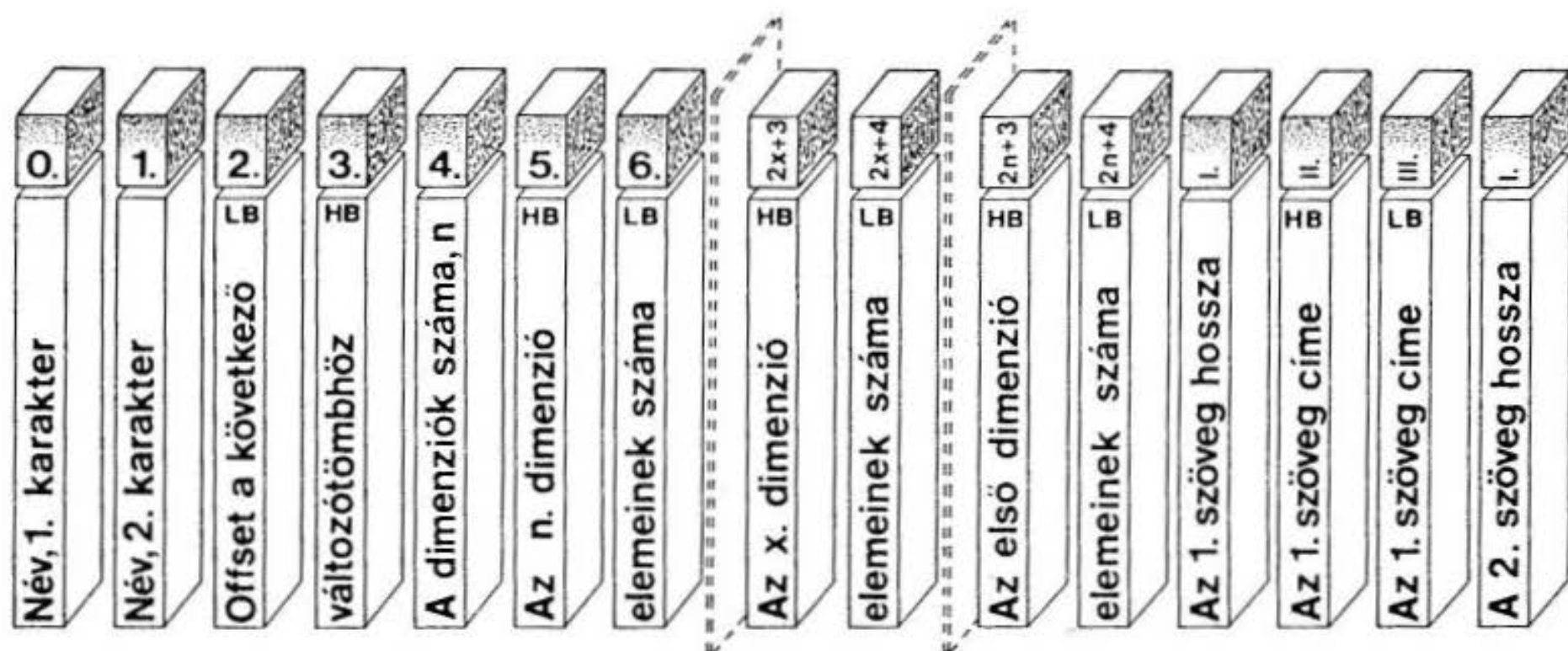
Egész tömbök



31. ábra. Az egész típusú tömb

A felépítés, főként a fejléc felépítése nem tér el a valós tömb felépítésétől. Az egyetlen különbség, hogy egy változó számára nem 5, hanem csak 2 byte van lefoglalva. Így az egész tömb azonos elemszám esetén is jóval rövidebb a valós tömbnél.

Szöveg tömbök



32. ábra. A szöveg típusú tömb

A tömb fejlécének felépítése azonos a valós tömbével. A tömb elemei helyett viszont mutatók találhatók. A mutatók 3 byte-ot foglalnak le elemenként. A valódi változók a szövegterületen helyezkednek el.

3.5. Perifériák és kezelésük

A Commodore 64 számos perifériát vezérelhet. E perifériák egy részét a soros buszra, más részét a bővítőportra, a user portra stb. lehet csatlakoztatni. A csatlakoztatható perifériák:

- a kazettaportra: szalagos egység
- a user portra: nyomtatók, más számítógépek
- a soros buszra: lemezegységek, nyomtatók
- a bővítőportra: winchester (merevlemez),
lemezegység (pl. SFD 1001), illetve
CP/M cartridge

Az operációs rendszer (KERNAL) a kazettaportra, a soros buszra és a felhasználói portra csatlakoztatott egységeket tudja kezelni. A felhasználói portra azonban csatlakoztatható olyan külső egység is, amelyhez külön szoftver szükséges. A bővítőportra olyan egységek csatlakoztathatók, amelyeket külön szoftver hajt meg. Ez a szoftver általában az IEEE — 488 cartridge beépített szoftvere. Az IEEE — 488 szabványos párhuzamos buszvezérlő 8 bites páru-

zamos adatátvitelhez. A KERNAL operációs rendszer a külső perifériákat az alábbi egység-számokkal azonosítja:

- 0 — billentyűzet
- 1 — szalagos egység
- 2 — RS232
- 3 — képernyő
- 4 — nyomtató #0
- 5 — nyomtató #1
- 8 — lemezmeghajtó #0
- 9 — lemezmeghajtó #1
- 10 — lemezmeghajtó #2
- 11 — lemezmeghajtó #3

Természetesen egészen különleges perifériák is elképzelhetők, mert a C64-es hardvere nyitott. Ezáltal a bővítőportra olyan külső, perifériának tekinthető egységek is csatlakoztathatók, amelyeknek elsődleges feladata nem az adatátvitel. Ilyen periféria lehet pl. egy ipari robotvezérlő vagy egy mérőműszer illesztője.

A C64-es perifériakezelésével csak érintőlegesen foglalkozunk, ez a téma kellő részletességgel önmagában is megtöltene egy kötetet. A gyakorlatban az amatőr programozó csak három külső perifériával kerülhet közelebbi kapcsolatba: a kazettás egységgel, a lemezegységgel és a nyomtatóval. (Nem említjük a billentyűzetet és a képernyőt, mert használatuk magától értetődik.)

3.5.1. A kazettás egység

A kazettás egység jelentősége viszonylagosan csekély. Elterjedtségét az magyarázza, hogy a gép kazettás magnóval együtt, készletben volt kapható. Bár viszonylag sok C64-tulajdonos rendelkezik kazettás magnóval, a lemezegység — számos előnye miatt — később kiszorította az alkalmazásokból. A kazettás egységgel az a legnagyobb baj, hogy lassú. A két háttértároló közötti különbség érzékelhetővé válik, ha paramétereiket összehasonlítjuk:

	Lemezegység	Kazettás egység
Tárolókapacitás	170 kbyte	változó, 40 — 80 kbyte
Átviteli sebesség	0,4 kbyte/s	0,05 kbyte/s
A létrehozható file-ok típusa	program szekvenciális relatív felhasználói	program szekvenciális
Az átvitel gyorsítása	megoldható	megoldható
A kapacitás növelése	nem lehetséges	lehetséges
Keresési megoldás	programozható, gyors	szekvenciális, lassú
Adathordozó	5 ¹ / ₄ " floppy disk	C60, C90, C120 audiokazetta
Befektetési költség	az alapgép árának mintegy 120 %-a	az alapgép árának mintegy 25 %-a

Amint az a fenti táblázatból is látható, a mérleg nyelve a lemezegység felé billen el; előnyei felülműlják az ára okozta hátrányt. Ezt a hátrányt az adathordozó olcsósága és elterjedtsége is csökkenteni némileg. A kazettás egység javára írható, hogy a felírás sebességének növelésével tárolókapacitása növelhető. Így egy kazettára kb. 4 lemezoldalnyi adat, program (680 kbyte) másolható.

A kazettás egység tárolási rendszere

A Commodore 64 a szalagon az egységnyi információt PPM-rendszerben tárolja (PPM — Pulse Position Modulation), vagyis a bitek közvetlenül kerülnek szalagra, impulzusok formájában. Ez a digitális tárolási rendszer nagyobb védelmet biztosít az adatok elvesztése ellen, mint az, ha hangfrekvenciás jelekkel kódolnánk az információt.

A jeltárolás technikája a következő. A szalagon levő mágneses réteg nem teszi lehetővé a 0 és 1 (alacsony, illetve magas) digitális jelek közvetlen rögzítését, ezért kódolásra van szükség. Mivel a szalagon a mágneses térerősség idő szerint változik, kézenfekvő, hogy a jeltárolásnál az időt vegyük figyelembe mint moduláló tényezőt. A digitális állandókat három különböző hosszúságú jel kombinációjából álló összetett jel ábrázolja. Az egyszerűség kedvéért jelöljük ezeket egy-egy betűvel; rövid: S (short), közepes: M (medium), hosszú: L (long). A digitális állandók tehát: MMSS a magas (1), SSMM az alacsony (0). A byte-ok elválasztásához a rendszer az LLMM jelet használja.

Az adatbyte-ok ezekből a jelsorozatokból állnak. Az adatbyte-ok 9 bitesek. A 9. bit az adatbyte paritása. A paritás az adatbyte bitein végzett EXOR művelet eredménye.

A szalagos tárolási rendszerben az adatok egymás után kétszer kerülnek fel a szalagra.

Ez a technikai megoldás az adatvesztés elkerülésére szolgál. Az ugyanis gyakorlatilag valószínűtlen, hogy az adatblokk és ismétlése ugyanazon a helyen sérüljön meg.

A programfile szerkezete:

- programfejléc (header),
- programfejléc ismétlése,
- programblokk,
- programblokk ismétlése, és így tovább.

A programfejléc szerkezete:

- a fejléc típusa, 1 byte,
- kezdőcím, 2 byte,
- végcím, 2 byte,
- programnév, 16 byte,
- kitöltő karakterek.

A programblokk felépítése:

- 2 s üres hely,
- 9 byte visszaszámlálás (countdown) \$89 — 81 (ismétlés: \$09 — 01),
- adatok,
- ellenőrző összeg (checksum), EXOR művelet minden byte-on,
- végjel (LLSS SSSS SSSS SSSS SSSS),
- 0,16 s üres hely.

Az adatfile felépítése ezzel teljesen megegyezik, viszont a kezdő- és végcím helyén \$0000 áll. A szalagos tárolási módszer főként azért olyan lassú, mert minden adatot kétszer kell beolvasni a tárb.

Betöltés és tárolás

A töltés (LOAD) és tárolás (SAVE) során fontos szerepet játszik az ún. kazettapuffer. A puffer a tár \$033C — 03FB (828 — 1019) területén helyezkedik el, de ez megváltoztatható. A puffer mutatója a nulláslap \$B2 — B3 címén található, ennek módosításával a RAM bármely szabad területére áthelyezhető.

A kazettapuffer kettős feladatot lát el. Egyrészt írás-olvasás esetén az adatok először ide kerülnek. Másrészt a program betöltése esetén itt található a programfile fejléce (header). Ilyenkor a kazettapuffer néhány címe rögzített szerepet lát el:

\$033C — a fejléc típusa

01: a file eltolható, azaz BASIC kezdet, általában \$0801.

02: nincs fejléc, azaz a blokk adatblokk.

03: a file nem tolható el. Töltési kezdetét a fejlécben megadott cím határozza meg.

04: az adatfile fejléce.

05: EOT (End Of Tape), szalagvégjel.

- \$033D — 033E — a program kezdőcíme (ha adatfile, \$0000)
- \$033F — 0340 — a program végcíme (ha adatfile, \$0000)
- \$0341 — 0350 — a program neve
- \$0351 — 03FF — nem használt terület

A kazettás egységgel — tárolási és olvasási rendszere miatt — csak soros (szekvenciális) file és programfile hozható létre. A soros file adattárolásra szolgál, de csak akkor tudjuk felhasználni programjainkban, ha programozástechnikailag meg tudjuk oldani azt, hogy a file-t elejétől végéig be kell olvasni a tárba. Ez gyakorlatilag azt jelenti, hogy a 13. rekordot csak az előző 12 átolvasása után érhetjük el.

A szalagos egység kezelése

A kazettás egységen található kezelőszervek funkciója azonos a kazettás magnókon megszokott billentyűk funkciójával. Elhelyezkedésük hasonlíthat is. PAUSE billentyű ugyan nincs, de nincs is rá szükség, mert az egység motorját a C64-es vezérli. Ha a kazettás egységen lenyomjuk valamelyik billentyűt, a motor elindul. Viszont a 0-ás tárcím 5. bitjének alacsonyra állítása után a motor megáll. A processzor ezen a vonalon keresztül vezérelni tudja a kazettás egység motorját, tehát PAUSE programból is elérhető.

Program betöltésekor nem kell feltétlenül megadni a program nevét. Elég egy LOAD utasítást kiadni. Meg kell azonban jegyezni, hogy gépi program így nem tölthető a helyére. A LOAD parancs kiadását az interpreter megkönnyíti, mert a SHIFT és a RUN/STOP billentyű együttes lenyomására a képernyőn megjelenik a LOAD felirat, majd alatta a PRESS PLAY ON TAPE. Ezután kapcsoljuk be a kazettás egységet, nyomjuk meg a PLAY gombot. A képernyő ekkor egyszínűvé és üressé válik, azaz a VIC ilyenkor „kikapcsol”. Erre azért van szüksége az interpreternek, mert a szalagról való olvasás pontos időzítést kíván, amit a VIC esetleges megszakításkérése zavarna. A LOAD rutin ezután beolvassa a fejléct. Amennyiben a LOAD parancsban nem adtunk meg programnevet, a gép úgy veszi, hogy a következő programot akarjuk beolvastatni. Amikor beolvasta a fejléct, kiírja a megtalált program nevét: FOUND programnév. Ilyenkor vár a C= billentyű lenyomására kb. 8,5 másodpercig, s ha azt eközben nem nyomjuk le, automatikusan folytatja a betöltést. Ha lenyomjuk a C= billentyűt, a töltés azonnal folytatódik. A háttér ismét felveszi a keret színét, és a program betöltődik a tárba. Ha a betöltés elindítására a SHIFT + RUN/STOP billentyűket alkalmazzuk, a betöltött program azonnal el is indul.

LOAD utasítást kiadhatunk — a már említett mellett — többféleképpen is:

```
LOAD
LOAD "NEV"
LOAD "NEV",1
LOAD "NEV",1,1
```

Az első három BASIC, a negyedik gépi programot tölt.

Program mentésekor a tennivalók azonosak: kiadjuk az utasítást, és megnyomjuk a kazet-tás egység billentyűjét (itt a PLAY és RECORD billentyűt egyszerre). A többi az operációs rendszer önmaga végzi el.

A mentés (SAVE) utasításának szintaxisa befolyásolja a program jellegét. Ha csak SAVE "NEV" vagy SAVE "NEV",1 utasítást adunk, a tárban levő program mint BASIC program kerül a szalagra. Ha azonban a SAVE "NEV",1,1 utasítást alkalmazzuk, ez a program később csak az eredeti helyére tölthető vissza. Az operációs rendszer a másodlagos cím értékéből értesül arról, hogyan kell kimentenie a programot.

Ha a szalagon nem akarunk több programot tárolni, bár van még rajta hely, az operációs rendszer lehetőséget kínál az utolsó file megjelölésére. Ez a jel az EOT (End Of Tape) jel, amely azt jelzi a töltő rutinnak, hogy a szalagon nincs több program. Az EOT jelet a fejléc 1 byte-ja tartalmazza.

Amikor a rendszer EOT jelet (\$05) talál, a beolvasást megszakítja. Azt a programot azonban, amelyet EOT jellel együtt mentünk, később mégis be lehet olvasni. Ez azért lehetséges, mert az EOT jelet tartalmazó fejlécet nem a program elejére, hanem a végére teszi, azaz megismétli a fejlécet.

Az operációs rendszer a másodlagos cím értéke alapján helyezi el az EOT jelet, ha arra szükség van. A másodlagos cím értékei:

- 0: a program a tárban eltolható, EOT jel nincs,
- 1: a program a tárban nem tolható el, EOT jel nincs,
- 2: a program a tárban eltolható, EOT jel van,
- 3: a program a tárban nem tolható el, EOT jel van.

Ha adatfile-t nyitunk, EOT jelet szintén alkalmazhatunk. A megfelelő móduzt itt is a másodlagos cím adja meg. Az adatfile írásra és olvasásra nyitható meg, az EOT jelnek íráskor van jelentősége. A másodlagos cím értékei:

OPEN 1,1,0,"NEV"

- 0: az adatfile olvasásra nyitva,
- 1: az adatfile írásra nyitva, EOT jel nélkül,
- 2: az adatfile írásra nyitva, EOT jellel.

3.5.2. A lemezegység

A lemezegység önmaga is egy mikroszámítógép, saját processzorral, önálló operációs rendszerrel. A lemezegység a 6502 processzorra és a 6522 VIA I/O chipre épült mágneslemez háttértároló. A C64-es mellett a leginkább elterjedt lemezegység a VC — 1541-es. A számítógép és a VC — 1541 a soros buszon keresztül kommunikál egymással.

A lemezegység tárolási rendszere

A lemezegység tárolási kapacitása kb. 170 kbyte. Ezt az adatmennyiséget $5\frac{1}{4}$ inches mágneslemezen tárolja. A mágneslemez teljes kapacitása 174 848 byte/oldal; 683 szektor. A szabad szektorok száma 664. A sávok száma 35, a szektorok száma 17-21/sáv.

A meghajtó (drive) egy lemezoldal kezelésére képes. (Léteznek kétoldalas meghajtók is: 4040, 8050, SFD 1001 stb.)

A 35 sáv szektorokra bontható; egy-egy szektorban 256 adatbyte tárolható. A szektorok tartalmazzák az azonosításhoz szükséges jeleket és az adatblokkot. A blokk két részre bontható:

- blokkfej, 16 nyolcbites byte
- adatblokk, 256 nyolcbites byte és 4 byte azonosító

A blokk nem tartalmazza ugyan a szinkronjeleket és a GAP-et, de a továbbiakban mégis együtt említjük őket, mert a tagoltság rontaná az áttekinthetőséget.

A blokkfej szerkezete:

- szinkronjel (\$FF), 5 byte
- a blokkfej azonosítója (\$08), 1 byte
- a blokkfej ellenőrző összege (checksum), 1 byte
- a blokk sorszáma, 1 byte
- a sáv sorszáma, 1 byte
- ID alsó byte (lemezazonosító), 1 byte
- ID felső byte (lemezazonosító), 1 byte
- a blokkfej vége (\$0F), 2 byte
- blokkfej GAP (\$55), 8 byte

Az adatblokk szerkezete:

- szinkronjel (\$FF), 5 byte
- az adatblokk azonosítója (\$07), 1 byte
- adatbyte-ok, 256 byte
- az adatblokk ellenőrző összege (checksum), 1 byte
- az adatblokk vége (\$00), 2 byte
- adatblokk (interszektor) GAP (\$55), határozatlan

A sáv egyetlen folyamatos bitsorozat. A meghajtó a lemezre megszakítás nélkül ír. Az adatblokkok és egyáltalán az adatok megkülönböztetéséhez a byte-ok egy részét kódolt formában kell felírni. A blokkot és a blokkfejet a DOS 2.6 öt nyolcbites \$FF byte-tal (azaz 40 magas bittel) választja el egymástól; ez a jel a szinkronjel. Az 1541-es kétféleképpen ír a lemezre. Az egyik mód, amikor a szinkronjeleket helyezi el direkt formában, a másik pedig amikor kódolt formában írja fel az adatokat. A kódolás lényege az, hogy a nyolcbites byte-okat a rendszer tízbites byte-okra alakítja át, és ezeket írja fel a lemezre. A kódolás elnevezése az angol eredeti alapján GCR-kód (Group Code Recording). A GCR-technika kizárja annak

lehetőségét, hogy a hardver összetéve a szinkronjelet az adatokkal, mert a GCR rendszerben kódolt adatok sohasem alkothatnak 8-nál több magas bitből álló jelsorozatot, amit a rendszer szinkronjelként használ. A másik technikai követelmény, ami szükségessé teszi a GCR alkalmazását, az, hogy 2-nél több alacsony bit nem következhet egymás után, mert a hardver nem tudja őket szétválasztani.

A GCR-kód képzése közben a DOS a byte-ot két félbyte-ra bontja, s minden félbyte-hoz hozzárendel egy ötbites kódot (lásd a 14. táblázatot).

14. táblázat. A GCR-kód képzése

Félbyte		GCR-kód	Félbyte		GCR-kód
\$0	0000	01010	\$8	1000	01001
\$1	0001	01011	\$9	1001	11001
\$2	0010	10010	\$A	1010	11010
\$3	0011	10011	\$B	1011	11011
\$4	0100	01110	\$C	1100	01101
\$5	0101	01111	\$D	1101	11101
\$6	0110	10110	\$E	1110	11110
\$7	0111	10111	\$F	1111	10101

A DOS egyszerre négy nyolcbites adatbyte-ot dolgoz fel. A 40 bit tehát az öt nyolcbites byte-on problémamentesen ábrázolható. Mivel 4 adatbyte-ból 5 GCR-byte keletkezik, így egy szektor fizikailag nem 276 byte-ból áll. A szektorban azonban nem minden byte kódolt. A szinkronjel mellett a GAP sem kódolt. A DOS ezeket (\$FF, \$55) direkt módon, egy az egyben írja fel a lemezre.

Egy szektort ténylegesen a következő elemek alkotnak:

szinkronjel (\$FF)	5 x 8 bit	5 byte
fejblokk	8 x 10 bit	10 byte
fej-GAP (\$55)	8 x 8 bit	8 byte
szinkronjel (\$FF)	5 x 8 bit	5 byte
adatblokk	260 x 10 bit	325 byte
interszektor GAP (\$55)	változó	

A szektor tehát fizikailag 353 byte-ból áll. A szinkronjel és a GAP sohasem vesz részt GCR-konverzióban, a blokk olvasásakor a DOS ezeket nem is tölti be a RAM-ba. Az interszektor GAP hossza a sáv számától, a meghajtótól és a lemez mechanikai jellemzőitől függ. A GAP hosszát a DOS a formázás közben számítja ki a sávra felírt bitek számából.

A tartalomjegyzék

A lemezes tárolás fő előnye az áttekinthetőség. Ezt a tartalomjegyzék (directory) biztosítja. A tartalomjegyzék fizikailag kb. a lemez közepén helyezkedik el, a 18-as sorszámu sávon. A tartalomjegyzék két fő részből áll: a BAM-ból (foglaltsági térkép) és a file-bejegyzésekből.

A BAM

A *foglaltsági térkép* minden esetben a 18-as sáv 0. blokkján helyezkedik el. A BAM (Block Availability Map vagy Block Allocate Map) binárisan kódolt formában tartalmazza az egyes sávok egyes blokkjainak foglaltságát, illetve felhasználhatóságát. Példaként bemutatjuk a lemezegységgel szállított TEST/DEMO lemez 18-as sávjának 0. blokkját.

A 18. sáv 0. blokkja fontos szerepet tölt be; nemcsak a BAM-ot tárolja, hanem a DOS számára fontos változókat is. A blokk 3. byte-ja (blokk alatt a továbbiakban mindig adatblokkot értünk) tárolja a DOS verziószámát. Ha a byte tartalma \$41, a lemezt az 1541-es formázta meg. A verziószám azt mutatja meg a DOS-nak, hogy a lemez írható-e az 1541-essel. A blokkban található a lemez neve és azonosítója is.

A lemez 18-as sávjának 0. blokkját a következő felépítés jellemzi:

\$00 — 01: sáv-blokk láncolás. Mindig a 18-as sáv 1. blokkjára mutat.

\$02: DOS-verziószám. Értéke az 1541-esnél \$41.

\$03: \$00, nem használt.

\$04 — 07: az 1. sáv BAM adatai.

\$08 — 8F: a BAM (2 — 35. sávok).

\$90 — 9F: a lemez neve.

\$A0 — A1: kitöltő karakterek, \$A0.

\$A2 — A3: a lemez ID-je.

\$A4: \$A0.

\$A5 — A6: DOS-verziószám, az 1541-esnél 2A.

\$A7 — AA: \$A0.

\$AB — FF: nem használt.

A régebbi meghajtók a \$B4 — BF területen a BLOCKS FREE szöveget tárolták.

ID : 84

[illegible]

33. ábra. A TEST/DEMO lemez 18. sáv 0. blokkja

Egy sávhoz 4 byte tartozik. Az első byte a sáv szabad blokkjainak számát tartalmazza, a többi három byte minden egyes bitje egy blokk állapotát jelzi. Szabad blokkot magas (1) bit jelez. Mivel három byte-ban 24 bit van, a sávonkénti maximális blokkszám pedig 21, ezért a 3. byte felső bitjei (olykor akár 7 is) nem használtak. Nézzünk egy példát.

A BAM-ban a 7. sáv foglaltságát ez a 4 byte mutatja: \$15 - FF - FF - 1F. Ez azt jelenti, hogy 21 (\$15) blokk szabad a sávban. A térképet az utolsó három byte bitmintája jelképezi:

11111111111111111000

A bitek a szokásos ábrázolás fordítottjával értelmezendők. Balról a 0. bit van. Ha megnéz-

zük a 34. ábrát, látható, hogy az egyeseknek a pont felel meg. Egy másik sáv példája még pontosabban mutatja ezt. A 11. sáv mutatói: \$04 - 00 - 02 - 19, bitmintája:

```
0000000000100000010011(000)
```

Ha a nullákat csillagokra, az egyeseket pedig pontokra cseréljük:

```
*****.******.**..(***)
```

jól látható az azonosság a 33. ábra azonos sávszámú sorával.

**** BAM ****

"TEST/DEMO 12/84 " ID : 84

BloKK

11111111112

Sáv 012345678901234567890

```
-----
1 : *****
2 : *****
3 : *****
4 : *****
5 : *****
6 : *****
7 : .....
8 : .....
9 : .....
10 : .....
11 : *****.******.**..
12 : *****
13 : *****
14 : *****
15 : *****
16 : *****
17 : *****
18 : **..*..*..*.....
19 : *****
20 : *****
21 : *****
22 : *****
23 : *****
24 : *****
25 : **.....*.....
26 : .....
27 : .....
28 : .....
29 : .....
30 : .....
31 : .....
32 : .....
33 : .....
34 : .....
35 : .....
```

34. ábra. A BAM

A file-bejegyzések

A tartalomjegyzék értelmezésénél figyelembe kell venni azt, hogy a lemezen tárolt adatokat is, valamint az ún. directory-file-t (\$) is tartalomjegyzéknek nevezzük. A figyelemre azért van szükség, mert bár tartalmilag azonosak, szerkezetük jelentősen eltér.

A directory-file programfile-ként töltődik a számítógép memóriájába. Ennek szerkezete a következő:

- fejléc: a lemez neve, a lemez azonosítója (ID), a DOS-verzió jele (2A)
- file-bejegyzések: a file által elfoglalt blokkok száma, a file neve, a file típusa
- a lemezen található szabad blokkok száma.

A lemezen tárolt tartalomjegyzék ettől felépítésében nagymértékben, tartalmában kismértékben tér el.

A tartalomjegyzék file-bejegyzései a 18-as sávot teljes mértékben elfoglalhatják. A sávra a DOS 19 blokkot formáz, ebből egy (az első) a BAM számára foglalt, így a file-bejegyzések számára 18 blokk szabad. Egy blokk maximum 8 bejegyzést tartalmazhat. Ebből az következik, hogy maximálisan 144 különböző file-bejegyzés lehet a sávon, és ennek megfelelően 144 file a lemezen. A kapacitást a gyakorlatban nagyon ritkán használják ki, mert egy file átlagos hossza kb. 4—6 blokk lenne, ami nagyon csekély érték.

Egy tartalomjegyzékblokk felépítése:

- \$00—01: a következő tartalomjegyzékblokk sáv-blokk mutatói. Ha ez az utolsó blokk, a két byte rendre: \$00 és \$FF.
- \$02—1F: az 1. bejegyzés.
- \$20—21: nem használt.
- \$22—3F: a 2. bejegyzés.
- \$40—41: nem használt.
- \$42—5F: a 3. bejegyzés.
- \$60—61: nem használt.
- \$62—7F: a 4. bejegyzés.
- \$80—81: nem használt.
- \$82—9F: az 5. bejegyzés.
- \$A0—A1: nem használt.
- \$A2—BF: a 6. bejegyzés.
- \$C0—C1: nem használt.
- \$C2—DF: a 7. bejegyzés.
- \$E0—E1: nem használt.
- \$E2—FF: a 8. bejegyzés.

Egy bejegyzés 30 byte-ból áll. A bejegyzések felépítése:

0: file-típus

1—2: az első file-blokk sáv-blokk mutatói (kezdőcím).

- 3—18: a file neve. Ha rövidebb, mint 16 karakter, a maradék hely \$A0 (SHIFT-SPACE) karakterekkel van feltöltve.
- 19—20: csak relatív file-oknál; az első oldalszektor blokk sávja és szektora.
- 21: Csak relatív file-oknál; a rekord hossza.
- 22—25: nem használt.
- 26—27: az új file első blokkjának sáv-blokk mutatói felülírásnál (@:).
- 28—29: a file mérete blokkokban, LB—HB.

A 0. byte bitjei külön-külön is hordoznak információt.

- 0—2. bit: file-típus.
- 3—5. bit: nem használt.
- 6. bit: a file védettségét mutatja.
- 7. bit: a file szabályos lezárását mutatja.

A DOS 2.6 négy file-típust képes kezelni, de megkülönbözteti a törölt file-okat is. A bitek kombinációi:

- 000: törölt file. Jelzése: DEL.
- 001: szekvenciális file. Jelzése: SEQ.
- 010: programfile. Jelzése: PRG.
- 011: user file. Jelzése: USR.
- 100: relatív file. Jelzése: REL.

Az 101, az 110 és az 111 bitkombináció nem használt.

A 0. byte-ban a 3—5. bitek nem definiáltak. A 6. bit jelzi a DOS számára, hogy a file törölhető-e vagy sem. Ha a bit értéke magas, a file nem törölhető a szokásos módon. Ha újra megnyitjuk írásra (Append), a védettség törlődik. A directory file-ban a védettséget a file-típus után álló < jel mutatja, pl. PRG<. A 7. bit a file szabályos lezárását jelzi. Ha a file-t pl. a lemezegység kikapcsolásával zárjuk le (illetve nem zárjuk le), a tartalomjegyzék következő beolvasásánál a file-típus megjelölését egy * előzi meg, pl. *SEQ.

A tartalomjegyzék blokkjai nem a kézenfekvő logikai sorrendet követik a sávban. A 18 01 blokkot nem a 18 02 blokk követi, hanem a 18 04, azt pedig a 18 07. Erre azért van szükség, mert amíg a DOS feldolgozza az olvasófejtől érkezett adatokat, addig a motor kb. 2 blokkal elforgatja a lemezt. Ez a láncolási módszer érvényes a többi sáv láncolására is, de ott a blokkok közötti távolság nem 2, hanem 9 blokknyi.

Egy tartalomjegyzékblokk fizikai felépítését a 35. ábrán, a directory-file formátumát pedig a 36. ábrán láthatjuk:

ID : 84

35. ábra. A tartalomjegyzékblokk fizikai felépítése

A tartalomjegyzék-file megnyitható, mint a többi file, de csak olvasásra. A 18-as sáv tartalmát csak a blokkutasítások segítségével írhatjuk át.

** Tartalomjegyzék **

0	test demo 12 24 " 24 23	
11	"how to use"	prg
8	"how part two"	prg
8	"how part three"	prg
7	"how part four"	prg
4	"vic-20 wedge"	prg
1	"c-64 wedge"	prg
4	"dos 5.1"	prg
9	"printer test"	prg
4	"disk addr change"	prg
6	"view bam"	prg
14	"display t&s"	prg
4	"check disk"	prg
9	"performance test"	prg
5	"seq.file.demo"	prg
18	"rel.file.demo"	prg
7	"sd.backup.c16"	prg
7	"sd.backup.plus4"	prg
10	"sd.backup.c64"	prg
7	"print.64.util"	prg
7	"print.c16.util"	prg
7	"print.+4.util"	prg
13	"uni-copy"	prg
30	"c64 basic demo"	prg
35	" +4 basic demo"	prg
8	"load address"	prg
7	"unscratch"	prg
5	"header change"	prg
1	"seq test file "	seq
4	"teszt"	rel
278	blocks free.	

36. ábra. A directory-file formátuma

3.5.3. File-szervezés, file-szerkezet

A *programfile* és a *szekvenciális file* szervezése hardver szempontból semmiben sem különbözik egymástól, ezért együtt tárgyaljuk őket. Ezeknek a file-oknak a szervezése rendkívül egyszerű. A program- vagy adatblokk első két byte-ja adja mindig a következő blokk sáv-blokk mutatóját. Ha az utolsó blokkról van szó, a láncolási cím első byte-ja \$00, a második pedig azt az utolsó byte-ot mutatja a blokkban, amelyik még a file-hoz tartozik. Mindkét file tartalmát, adatait folyamatosan végig kell ahhoz olvasni, hogy az utolsó byte-hoz vagy rekordhoz jussunk. (Adatokat programfile-ban is tárolhatunk, de ez zavaró lehet.)

A *user-file* kezelése és tárolási rendszere lényegében azonos a soros file hasonló jellemzőivel. Írásra és olvasásra ugyanúgy nyitható meg, mint a szekvenciális vagy a programfile. A user-file egy blokkjában szintén 254 adatbyte helyezhető el, mert a blokk első két byte-ja a láncolás számára van fenntartva. Az utolsó blokkban található láncolási cím első byte-ja \$00, a második pedig a blokkban található adatbyte-ok közül az utolsóra mutat.

A *relatív file* főként abban különbözik a szekvenciális file-tól, hogy rekordjait külön-külön is el lehet érni, a file-t nem kell végigolvasni a kérdéses rekordig. Az, hogy a file rekordjaihoz

egyenként is hozzáférhetünk, speciális feladatot ró a lemezegység szoftverére: „emlékeznie” kell arra, hogy a kívánt sorszámú rekordot hol találja a lemezen. A DOS ezt az ún. oldalszektorokkal (Side-Sector) oldja meg. Az oldalszektorok lényegében mutatótáblázatok, amelyek a file rekordjainak sáv-blokk mutatóit tartalmazzák. E mutatók felhasználásával számítja ki a DOS a rekord helyét a lemezen. És ami nem elhanyagolható szempont: ezt kielégítő sebességgel teszi. A DOS az oldalszektorokat és azok tartalmát automatikusan hozza létre, így azzal a programozónak nem kell foglalkoznia.

Amint már a file-bejegyzéseknél említettük, a bejegyzésben a relatív file rekordhosszát és az első oldalszektor sáv-blokk mutatóit is tárolja a DOS. Erre azért van szükség, mert az oldalszektorok helye sem rögzített, hasonlóan a többi file-blokkhoz. File-nyitáskor viszont a DOS-nak azonnal meg kell keresnie az első (0.) oldalszektorot. Egy relatív file maximum 6 oldalszektorral rendelkezhet, ezek számozása 0-tól 5-ig tart.

Az oldalszektorban a mutatókon kívül szerepel még néhány adat. Az első két byte a szokásos módon tartalmazza a következő oldalszektor sáv-blokk mutatóit; így az oldalszektorok akár sorosan is végigolvashatók. A harmadik (\$02) byte tartalmazza az oldalszektor sorszámát (0 — 5). A negyedik (\$03) byte a rekord hosszát tárolja.

A további 12 byte tartalmazza mind a hat oldalszektor sáv-blokk mutatóit. Ha történetesen csak egy oldalszektor van, az első két byte kivételével a többi tartalma mind \$00. Tehát az összeláncolás mellett minden oldalszektor tartalmazza a saját és az összes többi címét is. Az oldalszektor további byte-jai egy-egy mutatópárt alkotnak, szám szerint 120-at. Ha az első (0.) oldalszektorról van szó, az első 120 rekord mutatója itt található. A hat oldalszektorban összesen 720 ilyen mutató létesíthető. Ez adja egy relatív file blokkszámának felső határát. Elvileg, 254 byte-os rekordokat feltételezve, egy relatív file 182880 byte-ból állhat, azonban ez meghaladja a lemez 167132 byte (658 blokk) kapacitását. A létrehozható rekordok száma sem korlátlan, de meglehetősen nagy számú (65535) rekord képzelhető el. A relatív file adatblokkjai ugyanúgy össze vannak láncolva, mint a többi file blokkjai, emiatt a relatív file sorosan is végigolvasható, mint egy szekvenciális file.

A relatív file-t első megnyitásakor célszerű megformázni. Ez azt jelenti, hogy a tervezett legmagasabb kulcsú (sorszámú) rekordba írunk egy CHR\$(255) karaktert. Ekkor a DOS létrehozza az oldalszektorokat, feltölti az egész file-t \$00 karakterekkel. Egy nagyobb file esetén a lemezegység akár percekig is dolgozhat, azonban célszerű a formázást az elején megoldani, mert az új rekordok létesítése a programot érezhetően lassítja. Emellett még az is előfordulhat, hogy a hibátlanul gondolt program ellenére is villog a LED a lemezegységen, ami hibát jelez. Nos, hiba valójában nincs, mert az új rekordot a DOS hibátlanul felírta a lemezre. A DOS szerzői azonban elfelejtették kivédeni azt a lehetőséget, hogy amikor olyan rekordba írunk, amely eddig még nem létezett a lemezen, ne küldjön RECORD NOT PRESENT hibaüzenetet a lemezegység. A hibaüzenetet ilyenkor figyelmen kívül kell hagyni. Egészen más a helyzet akkor, amikor olvasni próbáltuk a még nem létező rekordot. A hibaüzenetet ilyenkor komolyan kell venni.

Az 1541-es lemezegység memóriájában 5 puffer van, melyek közül egyet a BAM állandóan lefoglalva tart. Ha file-okkal dolgozunk, egy puffer a tartalomjegyzéknek van fenntartva, a

maradék három puffer alkalmazható file-kezelésre. Mivel a szekvenciális file-ok egyszerre csak egy puffert használnak, egyszerre három szekvenciális file-t tarthatunk nyitva. A relatív file viszont két puffert igényel, egyet az adatblokknak, egyet az oldalszektornak, így csak egy relatív file-t nyithatunk meg egyszerre. A relatív file mellett esetleg lehet egy szekvenciális file is.

3.5.4. A DOS kezelése

A lemezegységen egyszerre három csatorna nyitható meg, ezek a csatornák biztosítják a számítógép és a lemezegység közötti kommunikációt. Programozási oldalról nézve azonban 16 csatorna különböztethető meg. A csatornát az OPEN utasításban megadott másodlagos címmel határozzuk meg. A másodlagos cím a 0 — 15 közötti értékeket veheti fel. A 0 — 14 csatornák adatok továbbítására szolgálnak, a 15-ös csatornán keresztül pedig a DOS-t utasíthatjuk rendszerutasításainak végrehajtására.

A csatornák BASIC-ből a következőképpen nyithatók meg:

OPEN logikai file-szám, egység szám, másodlagos cím

A logikai file-szám a C64-es számára különbözteti meg a nyitott file-okat, a másodlagos cím pedig a lemezegység számára. A logikai file-szám 1 — 127 közötti érték lehet. Az egység szám a legtöbb esetben 8, de a 8 — 11 közötti értékek bármelyikét felveheti. Esetünkben a parancscsatorna megnyitását az alábbi BASIC utasítással is elvégezhetjük:

OPEN 15,8,15

A file megnyitását követi az adatátvitel, amely kétirányú lehet. Ha adatokat (vagy parancsot) továbbítunk a lemezegység felé a CMD, PRINT és PRINT# utasításokat használhatjuk. Ha adatokat kérünk onnan, a GET# és az INPUT# utasításokat alkalmazhatjuk.

Az output utasítások megkerülésére is van lehetőség. Ha parancsot továbbítunk, az OPEN után közvetlenül is állhat a parancs, például:

OPEN 15,8,15,"0:1"

vagy

OPEN 15,8,15,"NO: URES,00"

A parancscsatorna egyszerre input és output csatorna. Sem az egyik, sem a másik adatátviteli irány kijelölésére nincs szükség. Az adatcsatornák esetében azonban (a relatív file kivételével) meg kell adni a csatorna adatátviteli irányát. Az irányt mindig a számítógép szemszögéből nézzük, tehát input az, amikor a számítógép adatokat vár. Az irány kijelölése az OPEN utasításban lehetséges:

```
OPEN 1,8,2, "SOROS,S,R"  inputra,  
OPEN 2,8,2, "SOROS,S,W"  outputra,
```

vagy

```
OPEN 3,8,3, "USER,U,R"  
OPEN 1,8,1, "PROGRAM,P,W"
```

Az outputot a W (Write), az inputot az R (Read) jelzi.

I — Initialize: betölti a lemez BAM-ját a lemezegység RAM-jába. Törli a hibaüzenetet, és alapállapotba hoz néhány rendszerváltozót. A nyitott file-okat lezárja. Az író-olvasó fejet a 18-as sávra állítja. Ha a VC — 1541 számára a lemezformátum nem olvasható, vagy a 18-as sáv 0-ás szektora hibás, az utasítás végrehajtása hibaüzenetet generál. Használata:

```
OPEN 15,8,15  
PRINT#15,"I"
```

vagy

```
PRINT#15,"0:I"  
CLOSE 15
```

vagy

```
OPEN 15,8,15,"I"  
CLOSE 15
```

V — Validate: újraszervezi a lemez BAM-ját. Végigolvassa az összes file-t, és a foglalt blokkokat rögzíti. Lényegében egy üresre formázott lemez BAM-jából indul ki, és ahogyan érzékeli a file-ok elhelyezkedését, úgy foglalja a blokkokat az új RAM-ban. A szabályosan le nem zárt file-okat törli a tartalomjegyzékből.

```
OPEN 15,8,15,"V"  
CLOSE 15
```

R — Rename: egy, a lemezen megtalálható file nevének módosítására szolgál. A bejegyzésben szereplő file-nevet írja át, más adatállományra nincs befolyással. Csak szabályosan lezárt file-ok nevét változtathatjuk meg. Alkalmazása:

```
OPEN 15,8,15  
PRINT#15,"R:ADATOK=ADTOK"  
CLOSE 15
```

Az egyenlőségjel bal oldalán kell állnia az új névnek, egyébként műveletünk eredménye hibaüzenet.

C — Copy: lemezen belüli file-másolásra alkalmas. Segítségével összemásolhatunk egy file-ba több kisebbet. Az új file tartalmában azonos az eredeti file(ok) tartalmával, de neve különböző kell legyen. Az összemásolás programokra nem alkalmazható. Használatának

szükségessége ritkán merül fel. Akkor van értelme egy lemezen egy file-ról másolatot képezni, ha tudjuk, hogy pl. tesztelés közben megsérthetjük, esetleg felboríthatjuk az eredeti adatszerkezetet. Ilyenkor nyugodtan lehet kísérletezni, mert tudjuk, van eredeti kópiánk. Alkalmazása:

```
OPEN 15,8,15
PRINT#15,"C:KOPIA=ADATOK"
CLOSE 15
```

vagy

```
OPEN 15,8,15
PRINT#15,"C:GYUJTO=DATA1,DATA2,DATA3"
CLOSE 15
```

S — Scratch: file-ok törlése; a file-t törli a tartalomjegyzékből. A lemezen fizikailag csak néhány mutatót változtat meg, a file tartalmát ténylegesen nem törli. A törölt file ezután még visszaállítható, amíg a lemezre nem írtunk, ugyanis a törölt file blokkjai törlés után szabad blokként szerepelnek a BAM-ban. Emiatt a következő írásműveleteknél nagy annak a valószínűsége, hogy az előzőleg törlés útján felszabadult blokk(ka)t a DOS felhasználja az új file tárolására. Ezután a törölt file már nem menthető meg. Alkalmazása:

```
OPEN 15,8,15
PRINT#15,"S:FILE"
CLOSE 15
```

A DOS törlés után mindig a következő rendszerüzenetet adja ki:
01,FILES SCRATCHED,xx,00 — itt az xx a törölt file-ok száma.

N — New: az egész lemez törlése. Új, még nem használt lemezt DOS 2.6 formátumúra formáz. A művelet az egész lemez minden bitjét újraírja függetlenül attól, hogy korábban voltak-e rajta adatok, vagy sem. Emiatt körültekintően alkalmazzuk.

A törlés folyamata meglehetősen hosszadalmas a kétszeri írás és az ellenőrzés miatt, valamint amiatt, mert a meghajtónak legkevesebb 1950648 bitet kell felírnia a lemezre. A tényleges szám ennél valamivel nagyobb, mivel a szektorok közötti GAP-ek változó hosszúságúak lehetnek, de legkevesebb 32 bitesek.

A NEW parancsnak két változata létezik:

```
(1) OPEN 15,8,15
    PRINT#15,"N:DISK,01"
    CLOSE 15
```

```
(2) OPEN 15,8,15
    PRINT#15,"N:DISK"
    CLOSE 15
```


A két változat hatásában jelentősen eltér. Az első az előbb említett módon végigtörli és újraírja a lemezt, a második csak a 18-as sáv 0. és 1. blokkját törli. Emiatt az adatok túlnyomórésze megmenthető, ha a második változatot használjuk. A két változat közötti különbséget az teszi, hogy az elsőben megadtuk a lemez új ID-jét, a másodikban pedig nem. Az ID (identifier) kétbyte-os azonosító, amely minden szektor fejlécében szerepel, és csak a formázás során változtatható meg. A DOS az ID alapján ismeri fel a lemezek közötti különbségeket, pl. hogy a meghajtóban kicseréltük a lemezt.

*A DOS különlegességei: a @, a * és a ? operátor*

A @ operátor akkor használatos, amikor a lemezre file-t akarunk felírni, de feltételezhetjük, hogy ilyen nevű file már van a lemezen. Ha az operátort nem alkalmazzuk, a DOS ilyen esetben 63, FILE EXIST, 00, 00 hibaüzenetet küld, és nem hajlandó a lemezre írni. Ahhoz, hogy ilyenkor a már meglévő file-t ne kelljen törölni, az utasításban szereplő file-név elé be kell iktatni a @: karaktereket:

```
SAVE "@:PROGRAM",8
OPEN 2,8,2,"@:SEQ.FILE,S,W"
```

A DOS ebben az esetben felülírja az előzőleg már felírt file-t. A felülírás azonban csak jelképes; a valóságban a DOS először tárolja az új programot vagy file-t a lemez még szabad blokkjain, és csak azután szabadítja fel a régi file blokkjait, ha az új file felírása hiba nélkül megtörtént. Ez a file-bejegyzésben szereplő mutató átírását jelenti.

A * és a ? operátor segítséget nyújt a file-ok kereséséhez. A * operátor a file-nevek rövidítését teszi lehetővé. Használata egyszerű. Ha van egy READER nevű programunk a lemezen, és azt is tudjuk, hogy R-rel kezdődő file-név nincs több a lemezen, elég megadni a rövidített file-nevet, így:

```
LOAD "R*",8
```

A csillag operátor másképpen is felhasználható. A SCRATCH (file-törlés) utasításban pl. minden olyan file-t töröl, amelyre igaz, hogy a név első néhány karaktere megegyezik a megadottal:

```
OPEN 15,8,15,"S:AD*:CLOSE 15
```

Az utasítás letöröl minden AD-vel kezdődő nevű file-t.

Ha a tartalomjegyzékben keresünk, az alábbi utasítás betölti az összes olyan file-bejegyzést, amelynek neve AD-vel kezdődik, álljon az bármennyi karakterből:

```
LOAD "AD*",8
```

A tartalomjegyzékben az állomány típusa szerint is kerestethetünk. A "\$*=S" kiválasztja a szekvenciális file-ok bejegyzéseit, és csak azokat tölti be. Az S kicserélhető a többi file-típus kezdőbetűjére is (P, U, R). Kombinálható az előbbi trükkel is:

LOAD "\$AD*=S",8

amely csak az AD-vel kezdődő szekvenciális file-ok bejegyzéseit tölti be. Az említett verzió alkalmas a SCRATCH megkönnyítésére is. Az

OPEN 15,8,15,"S:*=U" : CLOSE 15

utasítás csak a USER file-okat törli le a lemezről, de nevüktől függetlenül mindet.

A ? operátor a „mindegy” operátor. Használata azt jelenti, hogy azon a helyen, ahol áll, bármilyen karakter lehet a névben. Pl. a LOAD "???.TXT",8 utasítás betölti azt a programot, amelynek nevében az utolsó négy helyen .TXT áll, függetlenül az első három karaktertől.

A két operátor kombinálható, de figyelembe kell venni, hogy a csillag után nincs értelme kérdőjelet vagy más karaktert kitenni.

A DOS üzenetei

A DOS üzenetei túlnyomó többségükben hibaüzenetek. Az üzenet a parancscsatornán olvasható be a számítógépbe.

A beolvasó program BASIC-ben:

```
10 OPEN 15,8,15
20 INPUT#15,A$,B$,C$,D$
30 CLOSE 15
40 PRINT A$," ",B$," ",C$," ",D$
50 END
```

Gépi nyelven:

C000	100	*=\$C000	
FFB4	110	TALK	=\$FFB4
FF96	120	TKSA	=\$FF96
FFA5	130	IEC IN	=\$FFA5
FFD2	140	PRINT	=\$FFD2
AA07	150	CROUT	=\$AA07
FFAB	160	UNTALK	=\$FFAB
C000 20 D7 AA	170	DISK	JSR CROUT ; Return kiírása
C003 A9 6F	180		LDA #\$6F ; 15+\$60, parancscsatorna
C005 A2 08	190		LDX #\$08 ; egységszám
C007 85 B8	200		STA \$B8 ; logikai file-szám
C009 85 B9	210		STA \$B9 ; másodlagos cím
C00B 86 BA	220		STX \$BA ; egységszám
C00D 8A	230		TXA
C00E 20 B4 FF	240		JSR TALK ; az egység adatküldő
C011 A5 B9	250		LDA \$B9 ; másodlagos cím
C013 20 96 FF	260		JSR TKSA ; küldése TALK után
C016 20 A5 FF	270	AA0	JSR IEC IN ; egy byte a soros buszról
C019 C9 0D	280		CMP #\$0D ; Return ?
C01B F0 06	290		BEQ AA1 ; igen, befejezni
C01D 20 D2 FF	300		JSR PRINT ; nem, kiírni
C020 4C 16 C0	310		JMP AA0 ; folytatni
C023 20 AB FF	320	AA1	JSR UNTALK ; a TALK állapot törlése
C026 4C D7 AA	330		JMP CROUT ; Return kiírása
C029	340		.END

ZEILEN:25 SYMBOL:9 FEHLER:0

AA0 =C016 AA1 =C023 CROUT =AAD7 DISK =C000 IECIN =FFA5 PRINT =FFD2
TALK =FFB4 TKSA =FF96 UNTALK=FFAB

```

100 FOR I= 49152 TO 49192 :READ X:POKE I,X:S=S+X:NEXT
110 DATA 32,215,170,169,111,162, 8,133,184,133,185,134
120 DATA 186,138, 32,180,255,165,185, 32,150,255, 32,165
130 DATA 255,201, 13,240, 6, 32,210,255, 76, 22,192, 32
140 DATA 171,255, 76,215,170
150 IF S <> 5832 THEN PRINT "HIBA A DATASORBAN !":END
160 PRINT "OK !!":END

```

00, OK,00,00

Az utolsó I/O művelet során nem keletkezett hiba, alapállapot.

01, FILES SCRATCHED, xx,00

A SCRATCH parancs után érkezik; nem hibaüzenet. Az *xx* paraméter a törölt file-ok számát adja meg.

20, READ ERROR, TT, SS

Az FDC (Floppy Disc Controller) nem találja a fejblokkot. A fejblokk GCR-kódját (\$52) nem sikerült beolvasni 90 kísérlet után sem. A TT, SS a sáv, szektor számát mutatja.

21, READ ERROR, TT, SS

Az FDC nem találta meg a szinkronjelet (10 — 40 magas bit egymás után) 20 ms-on belül.

22, READ ERROR, TT, SS

Az adatblokkot megelőző byte tartalma nem egyezik meg a RAM \$0047-es címén tárolt \$07 byte-tal. Az adatblokkban egyébként nincs hiba, a GCR-konverzió hibás, mert az azonosító nem \$07.

23, READ ERROR, TT, SS

Az adatblokkban egy vagy több byte hibás. A lemezen tárolt ellenőrző összeg nem egyezik meg a kiszámított értékkel.

24, READ ERROR, TT, SS

Ha formázás közben a felírt adatok nem egyeznek meg a pufferrel, ez a hibaüzenet keletkezik. A lemez valószínűleg fizikailag hibás.

25, WRITE ERROR, TT, SS

A vezérlő az adatokat nem tudta hibátlanul felírni, a visszaolvasott adatok nem egyeznek az eredetivel. Valószínűleg fizikai hiba.

26, WRITE PROTECT ON, TT, SS

A lemezen írásvédelmi tapasz van, így nem írhatunk rá.

27, READ ERROR, TT, SS

Hibás fejblokk a sávon. A fejblokk ellenőrző összege nem egyezik a lemezen tárolttal. Az adatblokkot már nem olvassa be.

28, WRITE ERROR, TT, SS

A blokk felírása után a vezérlő nem találja megadott időn belül a következő szektor szinkronjelét. Rossz a lemez formázása, vagy a hajtómotor nem tartja a fordulatszámot, és a fej beleírt a következő szektorba.

29, DISK ID MISMATCH, TT, SS

ID keveredés. A meghajtóban levő lemez ID-je az olvasott szektorban nem egyezik a RAM-ban tárolt ID-vel. Vagy a lemez hibás, vagy a DOS nem végzett inicializálást lemezcseré után.

30, SYNTAX ERROR, 00, 00

A kiadott parancs a DOS számára értelmetlen.

31, SYNTAX ERROR, 00, 00

A parancs a DOS utasításkészletében nem szerepel.

32, SYNTAX ERROR, 00, 00

A küldött parancs hosszabb, mint 40 karakter.

33, SYNTAX ERROR, 00, 00

Szintaktikai hiba a file-névben.

34, SYNTAX ERROR, 00, 00

Nincs file-név, vagy az nem értelmezhető a kettőspont hiánya miatt.

39, FILE NOT FOUND, 00, 00

Érvénytelen parancs (&) hatására adott hibaüzenet.

50, RECORD NOT PRESENT, 00, 00

A rekordban még nincsenek adatok. Relatív file-nál olyan rekordot írtunk vagy olvastunk, amelyet a DOS még nem hozott létre. Írásnál ez nem hiba.

51, OVERFLOW IN RECORD, 00, 00

A relatív file-ba írt adat hossza túllépte a rekord hosszát. A túlsordult karakterek elvesznek.

52, FILE TOO LARGE, 00, 00

A file túl nagy. A relatív file rekordjainak mérete meghaladja a lemez kapacitását.

60, WRITE FILE OPEN, 00, 00

A file-t íráskor nem zártuk le. A hibaüzenet akkor keletkezik, ha ismét meg kívánjuk nyitni. A file az M módusz segítségével olvasható.

61, FILE NOT OPEN, 00, 00

A file nincs megnyitva, a hozzáférés nem lehetséges.

62, FILE NOT FOUND, 00, 00

Olyan file-ra hivatkoztunk az OPEN vagy LOAD utasításban, amely nincs a lemezen.

63, FILE EXIST, 00, 00

A megadott névvel már létezik file a lemezen. Vagy másik nevet, vagy a @ operátort kell alkalmazni.

64, FILE TYPE MISMATCH, 00, 00

File-típus keveredése. A megnyitott file neve igen, típusa nem egyezik a lemezen levő file típusával.

65, NO BLOCK, TT, SS

A blokk foglalt. A B-A (Block-Allocate) utasítás váltja ki, ha a lefoglalni kívánt blokk már foglalt. A TT, SS paraméterek a következő, legközelebbi szabad blokk sávját és szektorát adják. Ha 00,00 érkezik, a lemezen nincs szabad blokk.

66, ILLEGAL TRACK OR SECTOR, TT, SS

Nem elérhető sáv vagy szektor. Közvetlen lemezkezelés során olyan blokkot akartunk elérni, amely fizikailag nem létezik.

67, ILLEGAL TRACK OR SECTOR, TT, SS

Nem elérhető sáv vagy szektor. A file-láncolás hibás. A láncolási byte-ok fizikailag nem létező sávra és/vagy szektorra mutatnak.

70, NO CHANNEL, 00, 00

Nincs szabad csatorna. Minden puffer foglalt. A kijelölt puffer már foglalt. 3-nál több szekvenciális vagy programfile-t próbáltunk megnyitni. A közvetlen file-ok száma nagyobb négy-nél. Két relatív file-t próbáltunk megnyitni stb.

71, DIR ERROR, TT, SS

Hibás a tartalomjegyzék. A lemezen levő BAM nem egyezik a DOS nyilvántartásával.

72, DISK FULL, 00, 00

A lemez megtelt. A directoryban már 144 bejegyzés van. File-kezelés esetén a szabad blokkok száma 3-nál kevesebb.

73, CBM DOS V2.6 1541,00,00

Bekapcsolási üzenet. A lemezegység inicializált állapotát jelzi. Ha a LED is villog, a lemezt, amelyre a DOS írni próbált, nem az 1541-es formátumával formázták meg. A lemez nem íráskompatibilis.

74, DRIVE NOT READY, 00, 00

Az egység nem üzemkész. A meghajtóban nincs lemez, vagy a lemezzörgzítő nincs bekapcsolva. A lemeztok annyira szoros, hogy a motor a diszket nem képes 300 1/min fordulatra gyorsítani. A lemez nincs megformázva. A 18-as sáv 0-ás szektora beolvashatatlan. A lemezt nem Commodore drive formázta, stb.

3.5.5. Közvetlen lemezkezelés

Előfordulhat az az eset, hogy a lemez egy adott blokkjával kell foglalkoznunk. Ilyenkor a közvetlen blokk-kezelő utasításokat célszerű használnunk.

A közvetlen lemezkezeléshez egy puffert és egy csatorna szükséges. Az adatpuffer a blokk beolvasása és ismételt felírása közben átmeneti tárolóként működik. A DOS a file-névből fogja megtudni, hogy közvetlen lemezkezelés a szándékunk: a file-név a "#".

OPEN 2,8,2,"#"

A file-nyitás után a DOS kiválaszt egy szabad puffert, amely a beolvasott blokkokat tárolhatja. A DOS öt puffert különböztet meg, de az ötödiket mindig elfoglalja a BAM. Ha valamilyen más file-t is használunk, a negyedik puffer sem használható, mert ezt a tartalomjegyzék számára foglalja a DOS.

Ha konkrét puffert akarunk kijelölni, a következő módon tehetjük:

OPEN 2,8,2,"#2"

Esetünkben a 2-es puffert (ez a harmadik) akarjuk használni. Ha a puffert már lefoglalta a DOS, a lemezegység a 70, NO CHANNEL, 00,00 hibaüzenetet küldi. Általában jobb a DOS-ra bízni, hogy melyik puffert jelöli ki. A pufferszámot úgy tudhatjuk meg, hogy a file-nyitás után azonnal beolvasunk egy karaktert a csatornáról. Ez a karakter a csatorna száma lesz.

A pufferek elhelyezkedése az 1541-es RAM-jában:

0	\$0300 — 03FF	768 — 1023
1	\$0400 — 04FF	1024 — 1279
2	\$0500 — 05FF	1280 — 1535
3	\$0600 — 06FF	1536 — 1791
4	\$0700 — 07FF	1792 — 2047

A közvetlen blokk-kezeléssel hat utasítás foglalkozik: B-R, B-W, B-E, B-A, B-F és B-P.

A Block-Read, B-R utasítás beolvas egy blokkot a megnyitott pufferbe. Formátuma:

B-R *csatornaszám* *meghajtóegység* *sáv* *blokk*

Példa: **PRINT#15,"B-R 2 0 18 1"**

A PRINT# utasítás szintaxisa más is lehet:

PRINT#15,"B-R";2;0;T;S

ahol T a sáv, S a blokk száma.

Ha VC — 1541-sel dolgozunk, a meghajtó száma mindig 0. A csatorna számának az OPEN másodlagos címével kell egyenlőnek lennie.

A Block-Read kellemetlen tulajdonsága, hogy a blokk beolvasása után a puffermutatót nem a 0., hanem az 1. byte-ra állítja. Ezt kétféle módon védhetjük ki:

1. A puffermutatót a B-P utasítással a 0. pozícióra állítjuk.

2. A B-R utasítás helyett az U1 utasítást használjuk, amelynek nincs ilyen tulajdonsága, a puffermutatót a 0. byte-on hagyja.

Egy alkalmazási példa:

```
10 OPEN 15,8,15
20 OPEN 2,8,2,"#"
30 PRINT#15,"B-R 2 0 18 0"
40 GET#2,A$
50 PRINT ASC(A$+CHR$(0))
60 CLOSE 2:CLOSE 15
70 END
```

A program beolvassa a 18-as sáv 0. blokkját egy üres pufferbe, és onnan a GET# utasítás a blokk 1. byte-ját beolvassa az A\$ változóba. Ha a 0. byte-ot akarjuk elsőnek beolvasni, cseréljük ki a 30-as sort:

30 PRINT#15,"U1 2 0 18 0"

A Block-Write, B-W utasítás a puffer tartalmát a lemez megadott blokkjába írja. Ez a blokk bármelyik lehet, bár nem mindegyik blokk felülírása mentes a következményektől. A B-W utasítás formátuma:

B-W *csatornaszám* *meghajtóegység* *sáv* *blokk*

Példa: **PRINT#15,"B-W 2 0 10 3"**

A B-W utasításnak is van hátrányos tulajdonsága: a puffer 0. byte-ját felülírja az írás előtti utolsó puffermutatóval. A puffer tartalma ezután kerül a megadott blokkba. Ez a kellemetlenség is kivédhető: az U2 utasítás ugyanazt végzi, amit a B-W, de nem változtat sem a puffer tartalmán, sem a puffermutatón.

Egy alkalmazási példa: a lemez nevét változtatjuk meg.

```
10 OPEN 15,8,15
20 OPEN 2,8,2,"#"
30 PRINT#15,"U1 2 0 18 0"
40 PRINT#15,"B-P 2 144"
50 INPUT "LEMEZNEV: ";A$
60 IF LEN(A$)>16 THEN A$=LEFT$(A$,16)
70 IF LEN(A$)<16 THEN A$=A$+CHR$(160):GOTO 70
80 PRINT#2,A$;
90 PRINT#15,"U2 2 0 18 0"
100 PRINT#15,"I"
110 CLOSE 2:CLOSE 15
120 END
```

A program először beolvassa a 18-as sáv 0. blokkját. A puffermutatót beállítja a 144. byte-ra. Ezután kéri a lemez új nevét, amelyet megfelelő formátumra hoz. Ezt követően az új nevet beírja a pufferbe. A puffer tartalmát az U2 utasítás felírja a 18-as sáv 0. blokkjába. A lemez neve ezután az új név lesz, erről a tartalomjegyzék betöltése után magunk is meggyőződhetünk.

A Block-Execute, B-E utasítás beolvas egy blokkot a lemezeről a pufferbe. Ebben a blokkban olyan gépi kódú programnak kell lennie, amelyet a 6502 megért és végre tud hajtani, mert

beolvasás után a DOS egy indirekt ugrással a puffer első (0.) byte-jára ugrik, és a programvégrehajtást onnan folytatja. A puffert célszerű előre kijelölni, ha a gépi programban belső, abszolút ugrások is vannak. Az utasítás formátuma:

B-E *csatornaszám* *meghajtóegység* *sáv* *blokk*

Példa: **PRINT #15,"B-E 2 0 15 0"**

Alkalmazási példa:

```
10 OPEN 15,8,15
20 OPEN 2,8,2,"#2"
30 PRINT#15,"B-E 2 0 15 0"
40 CLOSE 2:CLOSE 15
50 END
```

A program betölti a 2-es pufferbe a 15-ös sáv 0. blokkját, és a programvégrehajtás a \$0500 címen folytatódik. Ha blokkban nincs értelmezhető gépi program, akkor a példaprogram minden bizonnyal lefagy. Ezt egyszerűen kivédhetjük. Az U1 utasítással a blokk 0. byte-jába #96-ot írunk, ez az RTS kódja. Így a program már lefut.

A Block-Allocate, B-A utasítás blokkot foglal le a BAM-ban. Erre azért van szükség, mert a közvetlen lemezkezelés U2 utasítása nem teszi automatikusan foglalttá a blokkot a BAM-ban. Emiatt normál file-műveletek esetén a DOS a blokkot saját céljaira felhasználhatja. Ezt nem teheti meg, ha a blokkot előzetesen lefoglaltuk. A B-A utasítás formátuma:

B-A *meghajtóegység* *sáv* *blokk*

Példa: **PRINT #15,"B-A 0 10 2"**

Alkalmazási példa:

```
10 OPEN 15,8,15
20 INPUT "SAV, SZEKTOR:";T,S
30 PRINT#15,"B-A 0 ";T;S
40 INPUT#15,A$,B$,C$,D$
50 PRINT A$," ",B$," ",C$," ",D$
60 CLOSE 15
70 END
```

A program megkísérli lefoglalni a megadott blokkot. Ha ez sikeres volt, a 00, OK, 00,00 üzenetet kapjuk. Ha a blokk már foglalt volt, a 65,NO BLOCK, *sáv*, *blokk* üzenetet kapjuk. A *sáv* és a *blokk* adja a legközelebbi még nem foglalt blokk sáv-blokk mutatóit. Ha a hibaüzenet végén 00,00 jelenik meg, minden blokk foglalt a lemezen. Ha olyan blokkot akarunk lefoglalni (pl. 19. sáv 21. blokk), amely fizikailag nem létezik a lemezen, a 66, ILLEGAL TRACK OR SECTOR, 19,21 hibaüzenetet kapjuk. Az így lefoglalt blokkokat a VALIDATE utasítás felszabadítja, ezért lehetőleg ne használjunk validálást olyan lemezen, ahol közvetlen elérésű állományunk van.

A Block-Free, B-F a Block-Allocate utasítás ellentéte: felszabadítja a megadott blokkot a BAM-ban. Alkalmazása nem járhat hibaüzenettel, hacsak nincs valami komoly baj a lemezegységgel vagy a lemezzel. Az utasítás formátuma:

B-F *meghajtó* *sáv* *blokk*

Példa: **PRINT #15,"B-F 0 10 2"**

Alkalmazási példa:

```
10 OPEN 15,8,15
20 INPUT "SAV, SEKTOR: ";T,S
30 PRINT#15,"B-F 0. ";T;S
40 CLOSE 15
50 END
```

A Buffer-Pointer, B-P utasítás a puffermutató beállítására szolgál. A puffer írásakor vagy olvasásakor a DOS megjegyzi, hogy melyik volt az utolsó byte, amelyet a pufferbe írtunk, illetve amelyet kiolvastunk belőle. Egészen pontosan a helyet tartja számon, ahol a kérdéses byte elhelyezkedik. Ha például egy blokkban csak a 144. byte-tól kívánunk írni, elég kényelmetlen lenne előtte 143 byte-ot beolvasni csak azért, hogy a puffermutató a megfelelő helyre mutasson. A puffermutatót persze beállíthatjuk másképpen is, ha tudjuk a puffer sorszámát. Ha a DOS rendelte ki a puffert, a számát az alábbi program megadja.

```
10 OPEN 2,8,2,"#"
20 GET#2,A$
30 PRINT ASC(A$+CHR$(0))
40 CLOSE 2
50 END
```

Ha megvan a puffer száma, tudjuk a címét is, valamint ki tudjuk számítani a mutató helyét a DOS RAM-jában. A mutatók a RAM \$99-es címétől helyezkednek el pufferszám szerinti növekvő sorrendben. A 0. puffer mutatója a \$99-9A byte-okon, az 1. puffer mutatója a \$9B-9C byte-okon található és így tovább. Ha ezekbe a byte-okba Memory-Write utasítással beírjuk a megfelelő értéket, nem kell használni a B-P utasítást. Mellesleg megjegyezve, ez a megoldás jóval bonyolultabb, mint a B-P utasítást használni. Nem is nagyon valószínű, hogy ezt így kelljen valakinek alkalmaznia. Viszont a puffermutató aktuális értékét megtudhatjuk, ha az előbb említett területről Memory-Read utasítással kiolvassuk. Ez a változat már valószínűleg előfordulhat a saját gyakorlatunkban is.

A B-P utasítás formátuma:

B-P *csatornaszám* *pozíció*

Példa: **PRINT#15,"B-P 2 16"**

A puffermutató 0 és 255 közötti értékeket vehet fel. Ezeken a határokon belül teljesen szabadon változhat, a 176. byte után nyugodtan olvashatjuk akár a 12.-et is.

3.5.6. Közvetlen memóriakezelés

Az eddigi utasítások nem adnak lehetőséget a DOS operációs rendszer ROM-jának és RAM-jának elérésére. Az itt tárgyalt utasítások segítséget nyújtanak a memória kezeléséhez. A következő utasítások állnak kapcsolatban a memóriával: M-R, M-W, M-E és User utasítások.

A Memory-Read, M-R utasítással a teljes címterület (\$0000 — FFFF) olvasható, bár nagy területek nem használtak, pl. a \$1C10 — BFFF. Az olvasás azonban ezeken a területeken is hibaüzenet nélkül zajlik le. Az M-R utasítás formátuma:

M-R cím alsó cím felső hossz

Példa: **PRINT#15,"M-R";CHR\$(12);CHR\$(0)**

vagy

PRINT#15,"M-R";CHR\$(0);CHR\$(2);CHR\$(20)

A hossz nem kötelező paraméter, elhagyása esetén a beolvasható adat egybyte-os.

Alkalmazási példa:

```
10 OPEN 15,8,15,"I"
20 PRINT#15,"M-R"CHR$(250)CHR$(2)
30 GET#15,L$:L$=L$+CHR$(0)
40 PRINT#15,"M-R"CHR$(252)CHR$(2)
50 GET#15,H$:H$=H$+CHR$(0)
60 PRINT ASC(L$)+256*ASC(H$)
70 CLOSE 15
80 END
```

Egy másik példa:

```
10 OPEN 15,8,15,"I"
20 PRINT#15,"M-R"CHR$(250)CHR$(2)CHR$(3)
30 GET#15,L$:L$=L$+CHR$(0)
40 GET#15,X$
50 GET#15,H$:H$=H$+CHR$(0)
60 PRINT ASC(L$)+256*ASC(H$)
70 CLOSE 15
80 END
```

A program a memória 762 — 764 tárcímeiről a szabad blokkok számát olvassa be anélkül, hogy a tartalomjegyzéket be kellene olvasni. Ahhoz, hogy eligazodjunk a lemezegység memóriájában, ismerni kell annak felépítését. Terjedelmi korlátok miatt azonban csak vázlatos képet tudunk adni a DOS memóriájáról.

\$0000 — 07FF:	2 kbyte RAM
\$0800 — 17FF:	nem használt
\$1800 — 180F:	VIA1, I/O regiszterek
\$1810 — 1BFF:	nem használt

\$1C00 — 1C0F:	VIA1, I/O regiszterek
\$1C10 — BFFF:	nem használt
\$C000 — FFFF:	DOS ROM, 16 kbyte

A Memory-Write, M-W feladata, hogy adatokat írjon a lemezegység RAM területére, mert a DOS írásra nem érhető el. Az M-W utasítással megváltoztathatjuk a VIA I/O chipek regisztereit is, de azzal is számolnunk kell, hogy ezek némelyikét a DOS azonnal felülírja. Az M-W utasítás formátuma:

M-W cím alsó cím felső hossz adat1 adat2...

Példa: **PRINT#15,"M-W"CHR\$(2) CHR\$(0) CHR\$(2) CHR\$(10) CHR\$(5)**

A hosszparaméter adja meg az átvihető adatok számát, amelyeket a megadott címtől kezdődően a DOS elhelyez a RAM-ban. A hosszt az input-puffer mérete korlátozza, amely 40 byte. Az egyszerre átvihető adathossz 34 byte, mert a parancs eleje 6 byte-ot lefoglal az input-pufferben.

Alkalmazási példa:

```
10 OPEN 15,8,15
20 PRINT#15,"M-W"CHR$(119)CHR$(0)CHR$(2)CHR$(9+32)CHR$(9+64)
30 CLOSE 15
40 END
```

A program 9-re változtatja a 8-as egység számát. Az egységszámot a DOS a nulláslap 119 és 120 címén tárolja. A 119-es címen a LISTEN egység száma található, a 120-as címen pedig a TALK egység száma. A LISTEN byte 5., a TALK byte 6. bitjét magasra kell állítani, a paraméterek ezért növekedtek 32-vel, illetve 64-gyel.

Az M-W utasítás segítségével a DOS egyes jellemzői megváltoztathatók, akár a rendszerváltozók módosításával, akár úgy, hogy gépi programot építünk a szabad pufferek valamelyikébe.

A Memory-Execute, M-E utasítás segítségével futtathatjuk a DOS rutinjait, de a Memory-Execute indíthatja el saját gépi programjainkat is, amelyeket a szabad pufferek valamelyikébe helyeztünk. A rutinok RTS-sel kell végződjenek. Az utasítás formátuma:

M-E cím alsó cím felső

Példa: **PRINT#15,"M-E"CHR\$(0) CHR\$(5)**

A címet a szokásos alsó-felső byte formában kell megadni.

Alkalmazási példa:

```
10 OPEN 15,8,15,"1"
20 PRINT#15,"M-W"CHR$(74)CHR$(0)CHR$(1)CHR$(55)
30 FOR I=1 TO 56
40 PRINT#15,"M-E"CHR$(59)CHR$(250)
50 NEXT I
60 CLOSE 15
70 END
```


A program „kiakasztja” az olvasófejet, azaz a 40. sáv fölötti sávra mozgatja, egészen a mechanikus ütközőig. A program ehhez a \$FA3B címen kezdődő DOS-rutint használja fel, amely megvizsgálja, hogy a \$4A (74) címen a léptetésszámláló nulla-e már. Ha igen, a rutin véget ér, ha még nem, a léptetőmotor az olvasófejet egy félsávval feljebb mozgatja, és egyben csökkenti a számlálót. Ahhoz, hogy a rutint használhassuk, a \$4A című byte-ot be kell állítani, esetünkben 55-re.

A *user utasítások* a lemezegység címtartományában elhelyezkedő előre meghatározott kezdőcímű programok futtatására alkalmasak. Az utasítások egy része ROM-rutint indít, más része a 2. pufferben elhelyezett ugrótábla alapján a RAM-ban található gépi rutinokat hívja meg. Az utasítások:

UA vagy U1	egyenértékű JMP \$CD5F, mint Block-Read,
UB vagy U2	egyenértékű JMP \$CD97, mint Block-Write,
UC vagy U3	egyenértékű JMP \$0500,
UD vagy U4	egyenértékű JMP \$0503,
UE vagy U5	egyenértékű JMP \$0506,
UF vagy U6	egyenértékű JMP \$0509,
UG vagy U7	egyenértékű JMP \$050C,
UH vagy U8	egyenértékű JMP \$050F,
UI vagy U9	egyenértékű JMP \$FF01,
UJ vagy U:	egyenértékű JMP \$EAA0, Reset.

A felsorolásból látható, hogy a lemezegység RESET-jét programból is létrehozhatjuk. A VC—1541 RESET rutinja futása idejére nem engedi meg a soros busz használatát. Az újabb lemezegységek (1570 és 1571) ezt már megengedik. Ha a lemezegységnek kiadjuk az UJ parancsot, és megkezd a RESET-et, az viszonylag sokáig tart. Ha ezalatt a számítógép megcímzi a lemezegységet, a DOS lemerevedik. Ha ezt a patthelyzetet el akarjuk kerülni, meg kell oldani, hogy a C64-es addig ne forduljon a DRIVE-hoz, amíg annak RESET-rutinja fut. Erre van lehetőség.

A soros busz vonalait közvetlenül is ellenőrizhetjük, mert a vonalak a CIA2 A portjára (\$DD00 = #56576) vannak kötve. A lemezegység RESET közben alacsonyan tartja a DATA IN vonalat, amely a CIA2 A portjának 7. bitjén figyelhető meg. A RESET rutin lefutása után a vonal ismét magasra vált. Így egyetlen utasítással várhatunk addig, amíg a vonal ismét magas nem lesz:

WAIT 56576, 128.

A várakozás megoldható kitartó ciklussal is, de gépi kódban problematikus pontos kitartó ciklust írni.

Gépi kódban a várakozás szintén egyszerű:

```
AA0 BIT $DD00
    BPL AA0
```


Az U1 és U2 utasításokat már ismertettük a B-R és B-W utasításoknál. Az U3-U8 utasítások a 2. pufferben elhelyezett ugrótábla alapján hívható rutinokat hívják. Az UI vagy U9 utasítás lényegében egy szűkített RESET-rutin, feladata a rendszerváltozók inicializálása.

3.5.7. File-kezelés

A programok durva egyszerűsítéssel két csoportra oszthatók: olyanokra, amelyek használnak file-okat, és olyanokra, amelyek nem. Azok a programok, amelyek használnak file-okat, nem feltétlen bonyolultabb, fontosabb programok, mint azok, amelyek nem, de a számítástechnikában jelentős szerepet töltenek be.

Adatfeldolgozás a technika mai szintjén nem képzelhető el háttértárolók nélkül. Ezeken a háttértárolókon nemcsak az adattárolás fizikai elve különbözik a számítógép memóriájában megvalósítható tárolás módjától, hanem (általában) a szerkezeti felépítés is. Ahhoz, hogy a háttértárolón tárolt adatainkat a számítógépen futó program fel tudja használni, szükség van valamilyen átalakítási technikára, hogy a mágneses úton tárolt információ digitális elektromos jellé alakuljon át. Ezt általában az illető háttértároló hardvere valósítja meg. Az adattárolás azonban nem csupán digitális jelek rögzítése. Fogalma nem merül ki abban, hogy a biteket felírjuk a szalagra vagy a lemezre. A visszaolvasáshoz és a visszkapott adatok értelmezéséhez az adattömeget valamilyen módon (ez lehet sematikus vagy teljesen egyedi is) szervezni, rendezni, egységesíteni kell. Ez a file-kezelés lényege, feladata.

Az *adatok programfile-ban* való tárolása a gyakorlatban nem terjedt el. Ennek oka főleg a tárolás körülményessége és rugalmatlansága. A programfile-ban tárolt adathalmazt BASIC programban kell elhelyezni, pl. DATA utasításokban. Ez a megoldás azonban tárolóhelyet pocsékol, munka- és időigényes. Ha a programfile-ba írjuk adatainkat, ezt megtehetjük soros file-ba is, mert kezelésük teljes mértékben megegyezik. A programfile-ban elhelyezett adathalmaz semmivel sem kezelhető rugalmasabban, mint az adatfile-ok tartalma.

A *soros (szekvenciális) file* rekordjai és azok mezői sorban egymás után írhatók és olvashatók. Nincs mód arra, hogy az adathalmaz részeit külön-külön írjuk fel és olvassuk be. A soros file-t az elejétől a végéig olvashatjuk, illetve írhatjuk. A soros file megnyitása:

OPEN 2,8,2,"SEQ.FILE,S" — olvasásra,
OPEN 2,8,2,"SEQ.FILE,S,R" — olvasásra,
OPEN 2,8,2,"SEQ.FILE,S,W" — írásra,
OPEN 2,8,2,"SEQ.FILE,S,A" — folytatólagos írásra,
OPEN 2,8,2,"SEQ.FILE,S,M" — lezáratlan file olvasására.

Gyakorlati példa szekvenciális file írására:

```
10 OPEN 2,8,2,"SEQ.FILE,S,W"
20 FOR I=0 TO 99
30 PRINT#2,I
40 NEXT I
50 CLOSE 2
```

A minta 0-tól 99-ig felírja az egész számokat egy soros file-ba. A számok a file rekordjai, amelyek CHR\$(13) karakterekkel vannak elválasztva. A rekordok nem oszthatók tovább mezőkre, de ha kicseréljük a 30-as sort:

30 PRINT#2,I; CHR\$(I), "AAA"

a rekord három mezőre oszlik. A mezők között itt nincsenek elválasztójelek, de a típus és a hossz alapján többé-kevésbé elkülöníthetők. Ha a mezők tartalma olyan, illetve hosszuk változó, feltétlenül tegyünk elválasztójelet a mezők közé is. Erre a célra a legalkalmasabb a CHR\$(13), mert adatként nagyon ritkán fordul elő, illetve az operációs rendszer külön utasítás nélkül is kiteszi.

A szekvenciális file tartalmának visszaolvasásához nem kellenek különleges ismeretek. A file szerkezetének ismeretében a visszaolvasás történhet INPUT# utasításokkal is. Ilyenkor arra kell csupán figyelmet fordítani, hogy az olvasó program felismerje a file végét. A file végét a BASIC ST rendszerváltozó 6. bitjének állapota jelzi. Ha a bit magas, a file utolsó karakterét is beolvastuk. Ha nem ismerjük a file szerkezetét (ez a ritkább eset), a GET# utasítást kell alkalmazni. Ilyenkor a programnak figyelnie kell arra is, hogy a GET# a CHR\$(0) karaktereket üres stringként olvassa be.

Gyakorlati példák olvasásra:

```
10 OPEN 2,8,2,"SEQ.FILE,S,R"
20 INPUT#2,A$
30 PRINT A$;
40 IF ST=64 THEN CLOSE 2:END
50 GOTO 20
```

A másik példa:

```
110 OPEN 2,8,2,"SEQ.FILE,S,R"
120 GET#2,A$
130 IF A$="" THEN A$=CHR$(0)
140 PRINT A$;
150 IF ST=64 THEN CLOSE 2:END
160 GOTO 120
```

Mindkét program azonos végeredményt produkál, ha az egyes rekordok hossza kisebb 88-nál. Ha ez nem áll fenn, az első program nem adja vissza hűen a file tartalmát, emiatt azt mondhatjuk, hogy érdemes az óvatosabb második, lassúbb változatot választani.

A szekvenciális file lezárása, mint a többi esetben, a CLOSE utasítással lehetséges. A CLOSE után álló számparaméterből tudja meg az operációs rendszer, hogy melyik file-t kell lezárnia. A számnak meg kell egyeznie az OPEN utasítás első paraméterével. Pl. ha egy file-t az OPEN 1,8,2,"ADAT,S,W" utasítással nyitottunk meg, az adatok felírása után a CLOSE 1

utasítással közölhetjük az operációs rendszerrel, hogy a file-ba a továbbiakban nem kívánunk írni. A CLOSE azért is fontos, mert enélkül a lemezen levő file lezáratlan marad. Ez azzal a kellemetlen következménnyel jár, hogy a felírt, de le nem zárt file tartalma a továbbiakban nem érhető el a szokásos olvasási módszerekkel. Akár írással, akár olvasással akarunk hozzáférni a file-hoz, a DOS a 60, WRITE FILE OPEN, 00,00 hibaüzenetet küldi. A DOS az ilyen esetekre is biztosítja az olvasás lehetőségét. Ha az ilyen file-t M módusszal nyitjuk meg, tartalma beolvasható és átírható egy másik file-ba.

Tételezzük fel, hogy az ADAT nevű szekvenciális állományunk felírása áramszünet miatt megszakadt. A DOS puffereiben levő adatoknak ilyenkor örökre búcsút mondhatunk, de ha a DOS előzőleg már több blokkot felírt a lemezre, azok még megmenthetők. Ezeket az adatokat átmásolhatjuk egy szabályosan lezárt file-ba. A megoldás azonban nem olyan egyszerű. A lezáratlan file-nak a DOS általában nem szokta a végét érzékelni, ezért a program vagy az idők végezetéig dolgozik, vagy a lemezegység küld valamilyen hibaüzenetet (pl. 67, ILLEGAL TRACK OR SECTOR, 75, 01). Ezért figyelni kell, hogy a beolvasott adatok meddig érvényesek, és hol kezdődik az érvénytelen, hozzácsatolt rész.

```
10 OPEN 2,8,2,"ADAT,S,M"
20 OPEN 3,8,3,".ADAT,S,W"
30 A=1:GET#2,A$
40 PRINT CHR$(18);A$;
50 WAIT 198,1:GET X$
60 IF X$=CHR$(13) THEN CLOSE 3:CLOSE 2:END
70 PRINT#3,A$;
80 GOTO 30
```

A program az ADAT nevű lezáratlan file-t másolja át a .ADAT nevű file-ba byte-onként. Minden byte után egy billentyű lenyomására vár. Ha a lenyomott billentyű a RETURN, a beolvasott byte-ot már nem írja át. A beolvasott byte-ok inverzben jelennek meg a képernyőn, így megakadályozhatjuk azt, hogy az érvényes adatállományhoz „szemét” csatlakozzon.

A relatív adattárolás elve a következőkben foglalható össze:

- a rekordok azonos, de korlátozott hosszúságúak,
- a rekordok sorszám szerint címezhetők,
- a rekordok sorban egymás után is beolvashatók,
- a rekordok kereséséhez nem szükséges külön szoftver.

A relatív elnevezés abból származik, hogy a DOS a rekordot relatíve az első rekordhoz képest keresi. A keresés kulcsa a rekord sorszáma.

A relatív file kezelésében az írásművelet két részre bontható. Az első megnyitás alkalmával hozza létre a DOS a file-t. De írni már előzőleg létrehozott file-ba is lehet. Ez a legjelentősebb különbség a relatív és a szekvenciális file kezelése között. Ezért a relatív file megnyitása után háromféleképpen kezelhető:

- (1) pozicionálás — az utolsó rekord felírása — a file lezárása
- (2) pozicionálás — a rekord felírása — a file lezárása
- (3) pozicionálás — a rekord beolvasása — a file lezárása

A relatív file-t nyitó OPEN utasítás szintaktikailag alig különbözik a szekvenciális file-t nyitó utasítástól. Az egyetlen különbség, hogy az adatáramlás irányát meghatározó módusz (W, R, A) hiányzik. A relatív file — megnyitása után — egyaránt írható és olvasható. A file-nyitó utasítás formátuma:

OPEN *logikai file-szám, egységyszám, csatorna, "file-név,L," + CHR\$(rekordhossz)*

Példa: OPEN 3,8,3,"DATA.A,L,"+CHR\$(189)

Az utasításban az L paraméter a relatív file azonosítója a DOS számára. A hosszparaméter értéke 2 — 254 között változhat. A tényleges hossz viszont 1 — 253 közé eshet, mert a rekordot egy CR zárja le.

A DOS a paramétereket vesszővel kéri elválasztani. Ezért az L után is vesszőt kell tenni, mert egyébként a DOS hibaüzenetet küldhet. Ez akkor fordul elő, amikor a file még nem szerepel a lemezen. Ilyenkor 62, FILE NOT FOUND, 00,00 hibaüzenetet kapunk.

Már létező relatív file megnyitása esetén a DOS megengedi a hosszparaméter elhagyását. Ilyenkor a tartalomjegyzékben szereplő hossz lesz érvényes.

Amennyiben a rekord hossza nem haladja meg a 88 karaktert, a rekord INPUT# utasítással egyben is beolvasható; ha viszont meghaladja, csak a GET# utasítás használható. Ha nem akarunk magunknak kellemetlenséget, a rekordhosszba mindig számítsuk bele a rekordot lezáró RETURN-t (CHR\$(13)) is. Így pl. egy 76 byte-os rekordhosszúságú relatív file nyitási parancsában hosszként 77-nek kell szerepelnie.

A relatív file-t a DOS automatikusan létrehozza az írásműveletek közben. A relatív file, éppen szervezési elve miatt, nem feltétlenül folytonos adathalmazt tárol. Két adatrekord között előfordulhat jókora „üres” terület is, azaz olyan előkészített rekordok, amelyekben még nincsenek adatok.

A file-t mindjárt a megnyitás után megformázhatjuk a kívánt hosszra: az általunk utolsónak tartott rekordba egy \$FF karaktert kell írunk. A DOS ekkor létrehozza az előtte álló összes rekordot, feltöltve azokat \$00 byte-okkal. Minden rekord első karaktere \$FF byte. Ily módon csak egyszer kell várnunk a DOS-ra. A továbbiakban a rekord elérése mindaddig néhány tizedmásodpercig tart, amíg a formálás során felírt rekordot nem lépjük túl. A folyamat a gyakorlatban nagyjából így néz ki:

```

10 OPEN 15,8,15
20 OPEN 2,8,2,"DATA.A,L,"+CHR$(189)
30 R=300
40 HB=INT(R/256):LB=R-HB*256
50 PRINT#15,"P"+CHR$(2)+CHR$(LB)+CHR$(HB)+CHR$(1)
60 PRINT#2,CHR$(255)
70 CLOSE 2
80 CLOSE 15
90 END

```

A példaprogram a DATA.A file-ban 300 db 189 byte hosszúságú rekordot foglalt. Ez több mint 200 blokkon fér el, a DOS ezért mintegy 3 percig egyfolytában dolgozik, s ezalatt kb. 60 000 byte-ot ír fel a lemezre.

A relatív file-kezelés során okvetlenül meg kell nyitni a parancscsatornát is, mert a pozicionálás parancsát a DOS csak ott tudja fogadni. A *pozicionálás* formátuma:

"P" csatorna alsó byte felső byte pozíció

Példa: **PRINT#15, "P"CHR\$(5)CHR\$(7)CHR\$(0)CHR\$(1)**

A példa az 5-ös csatornán megnyitott relatív file 7. rekordjának első byte-jára pozicionál.

Az alsó és felső byte-ok adják a rekordsorszámot (kulcsot). A paraméterek az alábbi módon számíthatók a rekord sorszámából:

$$HB = \text{INT} (R/256)$$

$$LB = R - HB * 256$$

A csatorna számának meg kell egyeznie az OPEN másodlagos címével. Az utolsó paraméter a rekordon belüli pozicionálásra szolgál. Ha értéke 1, a mutató a rekord elejére áll. Ha a paraméter elmaradt, a mutató a rekordonban tárolt adatokat lezáró RETURN-ra mutat. Ezzel vigyázni kell, mert a rekordon belül a szekvenciális file-ra vonatkozó szabályok érvényesek. Emiatt előfordulhat, hogy a beírt adat nem felülírja a régi rekordot, hanem mögéje kerül, de a rekord véges hossza miatt valószínűleg 51, OVERFLOW IN RECORD, 00,00 hibaüzenetet kapunk.

Ha a pozíciót meghatározó paraméter értéke eltér 1-től, ennek külön jelentősége van. Egy rekordon belül több, különálló adatot is elhelyezhetünk. Ilyenkor azt mondjuk, hogy a rekord mezőkre van osztva. A mezőket szintén RETURN választja el egymástól. Amikor a mezők hossza rögzített, és ez szinte mindig így van, az egyes mezők rekordon belüli kezdőpozícióját egyszerűen kiszámíthatjuk. A belső pozicionálással elérhetjük a rekord bármelyik mezőjét anélkül, hogy az egész rekordot be kellene olvasni. Ennek akkor van jelentősége, amikor a rekord hosszú és több mezőből áll. Egy 206 byte-os rekord utolsó előtti mezőjét sorosan jóval hosszabb idő alatt érhetjük csak el, mint a belső pozicionálás segítségével.

A rekordok felírása és beolvasása közvetlenül egymás után is következhet, mivel a relatív file-nak nincs előre kijelölt adatáramlási iránya. Az írás és olvasás a szokásos PRINT# és INPUT# (esetleg GET#) utasításokkal történhet. A rekordokat tárolás előtt elő kell készíteni, mert a rekord hossza rögzített. Bár a DOS megengedi, hogy a rekordhossznál rövidebb adatot írjunk a rekordba, a meglepetések elkerülése végett a kiírt adatot egészítsük ki üres karakterekkel a rekord teljes hosszáig. Két lehetőségünk van ennek megvalósítására. Vagy a mezőket töltjük fel megengedett hosszukig, vagy a fennmaradó helyet töltjük ki üres karakterekkel. A gondos előkészítés eredménye az adatok problémamentes visszaolvasása.

A relatív file egyedi sajátossága, hogy a rekord utolsó karakterének beolvasása után a lemezegységtől file-vége (EOF) jel érkezik. A BASIC program ezt az ST állapotváltozó 6. bit-jének állapota alapján érzékelheti. Ha magas (ST = 64), a DOS file-vége jelet küldött. Ennek alapján olyan relatív rekordot is be tudunk olvasni, amelynek nem ismerjük a hosszát.

A relatív file lezárása a megszokott CLOSE utasítással, a megszokott módon történik. Viszont ne felejtsük el, hogy a file megnyitásakor a parancscsatornát is megnyitottuk, ezért azt is le kell zárni.

Adatot általában szekvenciális és relatív file-ban tárolunk. A DOS mégis lehetőséget biztosít arra, hogy programfile-ban vagy user file-ban is helyezhessünk el adatokat. A kezelésükkel kapcsolatos tudnivalókat a szekvenciális file-nál már ismertettük, mert írás-olvasás tekintetében megegyeznek a szekvenciális file kezelésével. Tárolási rendszerük is hasonló. A programfile tér el némileg ettől, mert a file első két byte-ja a kezdőcím kell legyen, különben a DOS nem tudhatná, hogy a file-t adattárolásra használjuk.

A közvetlen lemezkezelés lehetőséget biztosít számunkra, hogy a feladathoz a legjobban illeszkedő file-szerkezetet létrehozassuk. Viszont ilyenkor a file-kezeléssel kapcsolatos összes feladatot a számítógép operatív tárában elhelyezkedő programnak kell végeznie. A programot nekünk kell megírnunk. Ezért, ha csak lehet, válasszuk a relatív file-t, mert kezeléséhez csak az utasítások ismerete szükséges.

4. A BASIC és a gépi nyelv ötvözése

BASIC-ben programozni viszonylag könnyű megtanulni. A gépi nyelv már keményebb dió, ám egy kis szívóssággal és türelemmel azt is el lehet sajátítani. Kezdetben nem sok sikerélménye van az embernek, de ha kitart, előbb-utóbb bekövetkezik: ötleteit, feladatait gyorsan meg tudja valósítani.

Ha nemcsak hobbiból ír programokat az ember, előbb-utóbb adódik olyan programozási feladat, amelyet terjedelme és bonyolultsága miatt csak BASIC-ben érdemes programozni. Úgy viszont lassú. Ami BASIC-ben programozható, az programozható gépi kódban is. A program ilyenkor gyors. De a sebességért általában nagy árat kell fizetni a fejlesztés során. A gépi program kifejlesztése jóval tovább tart, arról nem is beszélve, hogy óriási fejmunkát igényel.

Azt mondtuk, hogy ami BASIC-ben programozható, az gépi kódban is. Fordítva sajnos ez nem igaz. A gépi programok túlnyomó többsége megvalósítható BASIC-ben is, de pl. a megszakításkezelés, a buszműveletek nem. Ott, ahol pontos időzítésre és nagy sebességre van szükség, a BASIC egyáltalán nem alkalmas. Az ilyen programok mind gépi nyelvűek.

A gépi program nagy hátránya, hogy listája sokkal nehezebben követhető, és megértése lényegesen nehezebb, mint a BASIC-programé. A program belövése is sokkal tovább tart. Azonkívül módosítása is sokkal körülményesebb. Betölteni az assembly forrásnyelvi programot, módosítani, visszamenteni, lefordíttatni — mindez sokkal tovább tart, mint kilistázni a programot és módosítani a megfelelő sort.

Az optimum valahol a kettő között van. A programunk legyen csak BASIC — áttekinthetőbb, alakíthatóbb. Viszont amit csak lehet, írjunk meg gépi nyelven, és szubrutinként alkalmazzuk. BASIC-program több-kevesebb gépi kóddal. Ezzel a megoldással elérhetjük a viszonylagosan nagyobb sebességet, és a program továbbra is rugalmasan alakítható marad. A BASIC és a gépi kód ilyen együttese, szimbiózis tágabb lehetőségeket biztosít a programozónak, és a C64-es előnyös oldalait használja fel. Egyébként létezik olyan programnyelv, amely egyesíteni igyekszik a gépi nyelv hajlékonyságát, sokoldalúságát a BASIC egyszerűségével. Ez a nyelv a C.

Ez a fejezet tartalmaz egy sor olyan gépi nyelvű szubrutint, amelyeket programjainkba építve, megkönnyíthetjük programozási munkánkat. A gépi rutinok alkalmazásával meggyorsíthatjuk BASIC-programjaink futását, és tárhelyet takaríthatunk meg.

4.1. Hasznos rutinok

4.1.1. INPUT-rutin

A BASIC interpreter INPUT utasításának van néhány kellemetlen tulajdonsága. A rutin nem ellenőrzi a kurzor mozgását, ezért a képernyőmaszk (ha van) könnyen tönkretelhető. Az INPUT üzenetei (REDO FROM START, EXTRA IGNORED) szintén tönkretelhetik a képernyőmaszkot. Nem követelhető meg a típusok karakterenkénti szétválasztása (numerikus, illetve füzér), az adatbeviteli mező kezdőpozíciójának beállítása körülményes, stb.

Ahhoz, hogy az előbb említett feltételeknek megfelelő INPUT-utasítást alkalmazhassunk, saját rutint kell írunk. Az alábbi program egy ilyen INPUT-ot valósít meg. Jellemzői:

- paraméterek útján kommunikál a program többi részével;
- hat paramétere van:
 - (1) szöveges változó, pl. A\$,
 - (2) a mező sora,
 - (3) a mező oszlopa,
 - (4) a mező hossza,
 - (5) a mező típusa,
 - (6) módjelző;
- a mezőben szabadon mozoghatunk a kurzorral;
- a mód- és típusjelző által meghatározott feltételeknek megfelelő karaktereket tetszőleges sorrendben helyezhetjük el a mezőben;
- a törlés és javítás tekintetében a rutin ugyanazt a szabadságot biztosítja, mint az INPUT utasítás, egy kitétel: a mező határait sem törlés, sem beszúrás esetén nem lépheti át a kurzor, illetve a szöveg;
- a rutinból kétféleképpen léphetünk ki: RETURN-nel és a balra mutató nyíllal. A RETURN hatására a megadott változóba töltődik a mező tartalma, az IR változó nulla értéket vesz fel. A nyíl hatására a mező szintén a megadott változóba töltődik, de az IR változó 1 értéket kap;
- ha a megadott füzérváltozó (amibe a beolvasott adatot teszi a rutin) tartalma szerint nem üres, a változó tartalma a mezőbe íródik. Ha üres füzér, a mező is üres marad.

A rutin egyik legfőbb erénye, hogy nem léphetünk ki az input mezőből. Az IR változó értékétől függően dönthet a főprogram, hogy az adatbevitel során visszalépünk-e vagy előre. A kettős kiléptetési mód ott előnyös, ahol egy képernyőn több adatot is be kell olvasnunk. Programunkat úgy is szervezhetjük, hogy a nyilacska használatával akár az utolsó mezőből is visszajuthassunk az elsőbe, ha ott valamit ki akarunk javítani. Az IR változót figyelve vissza-

léphetünk az előző mezőt beolvasó utasításra. Az előzőleg megadott szöveg nem vész el, mert a rutin ismét betölti a mezőbe. Ezt egyébként nem is vesszük észre.

A numerikus és szöveges karaktereket eleve elkülöníthetjük egymástól, mert két beolvasási mód létezik: numerikus és alfanumerikus. Numerikus módban csak a 0—9 karaktereket és a tizedespontot fogadja el. Alfabetikus módban a 32—94 ASCII-kódok közé eső karaktereket fogadja el.

Ez a rutin segítséget nyújt az adatvédelemhez is. Ha a beolvasott adatokat lemezen tároljuk, visszaolvasásnál egy vessző, kettőspont, pontosvessző problémákat okozhat. A hatodik paraméter beállítása után ezeket a karaktereket a rutin nem fogadja el.

Rutinunknak vannak hátrányai is. A sok paraméter között nehezen lehet eligazodni. A példában szereplő rutin SYS utasítással hívható, de megoldható más hívási eljárás is.

Ha a szöveg hossza eleve nagyobb, mint a paraméterezésben megadott mezőhossz, a rutin automatikusan levágja a felesleget. Ha a numerikus adatbeolvasás során a teljes számaábrázolási tartományt használni akarjuk, az E, +, – jelek használatát engedélyezni kell. Ez csak az assembly lista módosításával érhető el.

A rutin hívásának szintaxisa:

SYS cím, változó, X-pozíció, Y-pozíció, hossz, típus, mód

Például: SYS 49152,AS,2,5,10,1,1

(1) *változó*: itt csak füzér típusú változó állhat.

(2) *X-pozíció*: annak a sornak a száma, amelyben a kurzornak meg kell jelennie; 0-tól 24-ig választható.

(3) *Y-pozíció*: annak az oszlopnak a száma, ahonnan az adatbeolvasás megkezdődik. 0-tól 38-ig választhatunk, de a hossz beolvasása után a rutin ellenőrzi, hogy a hossz és a pozíció összege kisebb-e 39-nél.

(4) *hossz*: az itt megadott hossz érvényes akkor is, ha a szöveges változó előzetes tartalma nagyobb mezőhosszt kívánna meg. A rutin ekkor levágja a hossz feletti részt.

(5) *típus*: ha 0: numerikus, ha 1: alfanumerikus.

(6) *mód*: nem kötelező paraméter, elhagyása automatikusan 0 értéket tételez fel. Ha 0: a vessző, pontosvessző, kettőspont engedélyezett, 1 megadása esetén tiltott.

A rutin assembly listája:

FFD2	100	PRINT	=\$FFD2	
E50C	110	CRSR	=\$E50C	
F13E	120	GETIN	=\$F13E	
0079	130	CHRGOT	=\$0079	
AEFD	140	CHKCOM	=\$AEFD	
B7F1	150	CHKGET	=\$B7F1	
009E	160	XPOS	=\$9E	
009F	170	YPOS	=\$9F	
00A6	180	HOSSZ	=\$A6	
00B0	190	TYPE	=\$B0	
00B1	200	FLAG	=\$B1	
00B2	210	LEN	=\$B2	
00B3	220	POS	=\$B3	
C000	230		*=\$C000	
C000 20 B7 C1	240	INPUT	JSR ERASE	; Az input-puffer törlése
C000 20 B7 C1	250		JSR READ	; A beolvasás megkezdése

C006	20	C2	C1	260		JSR OUT	; A puffer kiírása az input-mezőbe
C009	A5	B2		270		LDA LEN	; A szöveg hossza
C00B	85	B3		280		STA POS	; Kezdőpozíció az input-mezőben
C00D	20	AC	C1	290		JSR CRP	; A kurzor beállítása
C010	20	82	C1	300		JSR IRQ0	; Az IR változó törlése
							; Várakozási ciklus, belépési pont
C013	A9	00		310	ENTRY	LDA #\$00	
C015	85	CC		320		STA \$CC	; A kurzor engedélyezve
C017	85	C6		330		STA \$C6	; A billentyűzetpuffer törlése
C019	20	3E	F1	340	CC0	JSR GETIN	; Egy karakter beolvasása
C01C	F0	FB		350		BEQ CC0	; Nincs karakter, várakozni
C01E	C9	22		360		CMP #34	; Idézőjel ?
C020	F0	F7		370		BEQ CC0	; Igen, vissza
C022	C9	0D		380		CMP #\$0D	; Return ?
C024	D0	03		390		BNE CC1	; Nem, tovább vizsgálni
C026	4C	42	C1	400		JMP RETURN	; Befejezni a rutint
C029	C9	5F		410	CC1	CMP #"+"	; Megszakítás operátor ?
C02B	D0	03		420		BNE CC2	; Nem, tovább
C02D	4C	3F	C1	430		JMP BACK	; Befejezni a rutint
							; A karakter ellenőrzése
C030	C9	2C		440	CC2	CMP #", "	; Vessző ?
C032	F0	08		450		BEQ CC3	; Igen
C034	C9	3B		460		CMP #"; "	; Pontosvessző ?
C036	F0	04		470		BEQ CC3	; Igen
C038	C9	3A		480		CMP #": "	; Kettőspont ?
C03A	D0	04		490		BNE CC4	; Nem, tovább
C03C	24	B1		500	CC3	BIT FLAG	; A módjelző vizsgálata
C03E	50	D9		510		BVC CC0	; Nem engedélyezett, vissza
C040	C9	14		520	CC4	CMP #20	; Delete ?
C042	D0	03		530		BNE CC5	; Nem, tovább
C044	4C	79	C0	540		JMP DEL	; Egy karakter törlése
C047	C9	1D		550	CC5	CMP #29	; Kurzor jobbra ?
C049	D0	03		560		BNE CC6	; Nem, tovább
C04B	4C	C5	C0	570		JMP CRR	; Kurzor jobbra végrehajtása
C04E	C9	94		580	CC6	CMP #148	; Insert ?
C050	D0	03		590		BNE CC7	; Nem, tovább
C052	4C	E2	C0	600		JMP INST	; Egy karakter beszúrása
C055	C9	9D		610	CC7	CMP #157	; Kurzor balra ?
C057	D0	03		620		BNE CC8	; Nem, tovább
C059	4C	D6	C0	630		JMP CRL	; Kurzor balra végrehajtása
C05C	A6	B0		640	CC8	LDX TYPE	; A típusjelző
C05E	F0	0B		650		BEQ CC9	; Csak numerikus adat fogadható el
C060	C9	20		660		CMP #" "	; Szóköz, alsó határ
C062	90	B5		670		BCC CC0	; Kisebb, vissza
C064	C9	5F		680		CMP #"+"	; CHR\$(95), felső határ
C066	B0	B1		690		BCS CC0	; Nagyobb, vissza
C068	4C	14	C1	700	CD0	JMP PUT	; A karakter beépítése
C06B	C9	2E		710	CC9	CMP #", "	; Tizedespont ?
C06D	F0	F3		720		BEQ CD0	; Igen
C06F	C9	30		730		CMP #"0"	; CHR\$(48), alsó határ
C071	90	A6		740		BCC CC0	; Kisebb, vissza
C073	C9	3A		750		CMP #": "	; CHR\$(58), felső határ
C075	B0	A2		760		BCS CC0	; Nagyobb, vissza
C077	90	EF		770		BCC CD0	; Kisebb, beépíteni
							; Egy karakter törlése
C079	A5	B3		780	DEL	LDA POS	; A kurzorpozíció nulla ?
C07B	F0	1C		790		BEQ CD1	; Igen, a törlés nem lehetséges
C07D	C5	B2		800		CMP LEN	; A szöveg hossza nagyobb ?
C07F	90	1B		810		BCC CD2	; Igen, köztes törlés
C081	D0	57		820		BNE CRL1	; Nem, kisebb, kurzor balra
C083	20	75	C1	830		JSR CROUT	; A kurzor kikapcsolása
C086	A9	9D		840		LDA #157	; Kurzor balra karakter
C088	20	D2	FF	850		JSR PRINT	; Kiírása

C08B	A9	20		860		LDA #32	; Szóköz Karakter
C08D	20	D2	FF	870		JSR PRINT	; Kiírása
C090	A9	9D		880		LDA #157	; Kurzor balra Karakter
C092	20	D2	FF	890		JSR PRINT	; Kiírása
C095	C6	B2		900		DEC LEN	; A hossz csökkentése
C097	C6	B3		910		DEC POS	; A kurzorpozíció csökkentése
C099	4C	13	C0	920	CD1	JMP ENTRY	; Vissza a várakozási ciklusra
C09C	20	75	C1	930	CD2	JSR CROUT	; A kurzor kikapcsolása
C09F	A9	9D		940		LDA #157	; Kurzor balra Karakter
C0A1	20	D2	FF	950		JSR PRINT	; Kiírása
C0A4	A4	B3		960		LDY POS	; A kurzorpozíció beolvasása
C0A6	20	CF	C1	970		JSR CB1	; A hátralevő Karakterek Kiírása
C0A9	A9	20		980		LDA #32	; Szóköz Karakter
C0AB	20	D2	FF	990		JSR PRINT	; Kiírása
C0AE	A4	B3		1000		LDY POS	; A kurzorpozíció beolvasása
C0B0	B9	54	C2	1010	CD3	LDA BUF,Y	; A puffer tartalmának
C0B3	99	53	C2	1020		STA BUF-1,Y	; hozzáigazítása a képernyőhöz
C0B6	C8			1030		INY	
C0B7	C4	B2		1040		CPY LEN	; A szöveg hosszát elérte ?
C0B9	D0	F5		1050		BNE CD3	; Még nem, vissza
C0BB	A9	20		1060		LDA #32	; Szóköz Karakter
C0BD	99	53	C2	1070		STA BUF-1,Y	; a pufferbe a szöveg végére
C0C0	C6	B2		1080		DEC LEN	; A hossz csökkentése
C0C2	4C	DD	C0	1090		JMP CE7	; A várakozási ciklusra
C0C5	A5	B3		1100	CRR	LDA POS	; A kurzor
C0C7	C5	A6		1110		CMP HOSSZ	; az utolsó oszlopban ?
C0C9	F0	CE		1120		BEQ CD1	; Igen, a várakozási ciklusra
C0CB	20	75	C1	1130		JSR CROUT	; A kurzor kikapcsolása
C0CE	E6	B3		1140	CE6	INC POS	; A kurzorpozíció növelése
C0D0	20	AC	C1	1150	CE8	JSR CRP	; A kurzor bekapcsolása
C0D3	4C	13	C0	1160		JMP ENTRY	; A várakozási ciklusra
C0D6	A5	B3		1170	CRL	LDA POS	; A kurzor az első oszlopban van ?
C0D8	F0	BF		1180		BEQ CD1	; Igen, a várakozási ciklusra
C0DA	20	75	C1	1190	CRL1	JSR CROUT	; A kurzor kikapcsolása
C0DD	C6	B3		1200	CE7	DEC POS	; A pozíció csökkentése
C0DF	4C	D0	C0	1210		JMP CE8	; A várakozási ciklusra
C0E2	A5	B3		1220	INST	LDA POS	; A kurzorpozíció
C0E4	C5	B2		1230		CMP LEN	; viszonya a szöveg hosszához
C0E6	B0	B1		1240		BCS CD1	; Nagyobb
C0E8	A5	B2		1250		LDA LEN	; A szöveg hossza
C0EA	C5	A6		1260		CMP HOSSZ	; egyenlő a mező hosszával ?
C0EC	F0	AB		1270		BEQ CD1	; Igen, a várakozási ciklusra
C0EE	20	75	C1	1280		JSR CROUT	; A kurzor kikapcsolása
C0F1	A9	20		1290		LDA #32	; Szóköz Karakter
C0F3	20	D2	FF	1300		JSR PRINT	; Kiírása
C0F6	A4	B3		1310		LDY POS	; A kurzorpozíció
C0F8	20	CF	C1	1320		JSR CB1	; után Következő Karakterek Kiírása
C0FB	C6	B3		1330		DEC POS	; Csökkentés a számítások miatt
C0FD	A4	B2		1340		LDY LEN	; A szöveg hossza
C0FF	B9	54	C2	1350	CD4	LDA BUF,Y	; A puffer hozzáigazítása
C102	99	55	C2	1360		STA BUF+1,Y	; a képernyőhöz
C105	88			1370		DEY	; A mutató
C106	C4	B3		1380		CPY POS	; elérte a kurzorpozíciót ?
C108	D0	F5		1390		BNE CD4	; Még nem
C10A	C8			1400		INY	; Az alulcsordulás kompenzálása
C10B	A9	20		1410		LDA #32	; Szóköz Karakter
C10D	99	54	C2	1420		STA BUF,Y	; a pufferbe
C110	E6	B2		1430		INC LEN	; A hossz növelése
C112	D0	BA		1440		BNE CE6	; A várakozási ciklusra
C114	85	FF		1450	PUT	STA #FF	; A karakter beépítése
C116	20	75	C1	1460		JSR CROUT	; A kurzor kikapcsolása
C119	A5	B3		1470		LDA POS	; A kurzorpozíció
C11B	C5	A6		1480		CMP HOSSZ	; elérte a mező hosszát ?

C11D	F0	10		1490	BEQ	CD5	; Igen, ugrás
C11F	C5	B2		1500	CMP	LEN	; A kurzor a szöveg mögött van ?
C121	B0	0F		1510	BCS	PUT1	; Igen, ugrás
C123	A5	FF		1520	LDA	\$FF	; A Karakter visszaolvasása
C125	20	D2	FF	1530	JSR	PRINT	; és Kiírása a képernyőre
C128	A4	B3		1540	LDY	POS	; A kurzorpozíció szerint
C12A	99	54	C2	1550	STA	BUF,Y	; helyezni el a karaktert a pufferben
C12D	E6	B3		1560	INC	POS	; A kurzorpozíció növelése
C12F	4C	13	C0	1570	JMP	ENTRY	; A várakozási ciklushoz
C132	D0	04		1580	PUT1	BNE	PUT2 ; A kurzor az üres területen van, ugrás
C134	E6	B2		1590	INC	LEN	; A hossz növelése
C136	D0	EB		1600	BNE	CE3	; A Karakter beépítése
C138	A6	B3		1610	PUT2	LDX	POS ; A kurzorpozíció
C13A	E8			1620	INX		; szerint
C13B	86	B2		1630	STX	LEN	; változik a szöveghossz mutatója
C13D	D0	E4		1640	BNE	CE3	; Beépíteni a Karaktert
C13F	20	88	C1	1650	BACK	JSR	IRQ1 ; A rutin befejezése
C142	A2	00		1660	RETURN	LDX	#\$00 ; Az IR változóba 1-et tölteni
C144	86	0E		1670	STX	\$0E	; Valós változó típus
C146	CA			1680	DEX		
C147	86	0D		1690	STX	\$0D	; Szöveg típus
C149	A5	FD		1700	LDA	\$FD	; A megadott
C14B	85	45		1710	STA	\$45	; változó
C14D	A5	FE		1720	LDA	\$FE	; neve, első-második
C14F	85	46		1730	STA	\$46	; Karakter
C151	20	E7	B0	1740	JSR	\$B0E7	; A változó megkeresése/létrehozása
C154	85	49		1750	STA	\$49	; A változó címe, LB
C156	84	4A		1760	STY	\$4A	; " , HB
C158	A5	B2		1770	LDA	LEN	; A szöveg hossza
C15A	20	75	B4	1780	JSR	\$B475	; Helyfoglalás a szövegnek
C15D	A0	02		1790	LDY	#\$02	; A mutatók (hossz, cím)
C15F	B3	61	00	1800	CD6	LDA	\$0061,Y ; betöltése
C162	91	49		1810	STA	(\$49),Y	; a változó fejlécébe
C164	88			1820	DEY		
C165	10	F8		1830	BPL	CD6	
C167	AA			1840	TAX		; A hossz nulla ?
C168	F0	0B		1850	BEQ	CROUT	; Igen, a másolást átugrani
C16A	C8			1860	INY		
C16B	B9	54	C2	1870	CD8	LDA	BUF,Y ; A változó tartalmának
C16E	31	62		1880	STA	(\$62),Y	; bemásolása a RAM-ba
C170	C8			1890	INY		
C171	C4	61		1900	CPY	\$61	; Elérte a hosszt ?
C173	D0	F6		1910	BNE	CD8	; Még nem
C175	A9	02		1920	CROUT	LDA	#\$02
C177	85	CD		1930	STA	205	; A kurzorvillogás számlálója
C179	A5	CF		1940	CD9	LDA	207 ; A kurzor állapotjelzője
C17B	D0	FC		1950	BNE	CD9	; Látható, várakozni
C17D	A9	01		1960	LDA	#\$01	; A kurzor
C17F	85	CC		1970	STA	204	; Kikapcsolása
C181	60			1980	RTS		
C182	A9	00		1990	IRQ0	LDA	#\$00 ; Az IR változó törlése
C184	48			2000	PHA		; 0 érték a verembe
C185	48			2010	PHA		
C186	F0	06		2020	BEQ	CE0	; feltétel nélküli ugrás
C188	A9	00		2030	IRQ1	LDA	#\$00 ; Az IR változó beállítása
C18A	48			2040	PHA		
C18B	A9	01		2050	LDA	#\$01	; 1 érték a verembe
C18D	48			2060	PHA		
C18E	A9	00		2070	CE0	LDA	#\$00
C190	85	0E		2080	STA	\$0E	; Típus : valós
C192	85	0D		2090	STA	\$0D	; Típus : numerikus
C194	A9	49		2100	LDA	#\$01	; A változó

C196	A0	52		2110		LDY # "R"	; neve
C198	85	45		2120		STA \$45	
C19A	84	46		2130		STY \$46	
C19C	20	E7	B0	2140		JSR \$B0E7	; A változó megkeresése/létrehozása
C19F	85	49		2150		STA \$49	; A változó címe, LB
C1A1	84	4A		2160		STY \$4A	; " , HB
C1A3	68			2170		PLA	
C1A4	A8			2180		TAY	; Az érték visszatöltése
C1A5	68			2190		PLA	
C1A6	20	91	B3	2200		JSR \$B391	; Átalakítás lebegőpontosá
C1A9	4C	D0	BB	2210		JMP \$BBD0	; A FAC betöltése a változóba
							; A kurzor pozicionálása
C1AC	A6	9E		2220	CRP	LDX XPOS	; A mező X koordinátája (sor)
C1AE	A5	9F		2230		LDA YPOS	; A mező Y koordinátája (oszlop)
C1B0	18			2240		CLC	
C1B1	65	B3		2250		ADC POS	; A kurzorpozíció hozzáadása
C1B3	A8			2260		TAY	; Az eredmény az Y-ba
C1B4	4C	0C	E5	2270		JMP CRSR	; A kurzor beállítása
							; A puffer törlése
C1B7	A9	20		2280	ERASE	LDA #32	; Szóköz karakter
C1B9	A0	28		2290		LDY #40	; 40 byte
C1BB	99	54	C2	2300	CE1	STA BUF,Y	; A puffer törlése
C1BE	88			2310		DEY	
C1BF	10	FA		2320		BPL CE1	
C1C1	60			2330	CE2	RTS	
C1C2	A6	9E		2340	OUT	LDX XPOS	; A mező X koordinátája
C1C4	A4	9F		2350		LDY YPOS	; A mező Y koordinátája
C1C6	20	0C	E5	2360		JSR CRSR	; A kurzor beállítása
C1C9	A5	B2		2370		LDA LEN	; A szöveg hossza nulla ?
C1CB	F0	0D		2380		BEQ CB0	; Igen, befejezni
C1CD	A0	00		2390		LDY #\$00	
C1CF	B9	54	C2	2400	CB1	LDA BUF,Y	; A puffer kiírása
C1D2	20	D2	FF	2410		JSR PRINT	; a mezőbe
C1D5	C8			2420		INY	
C1D6	C4	B2		2430		CPY LEN	; A szöveg vége ?
C1D8	D0	F5		2440		BNE CB1	; Még nem, vissza
C1DA	60			2450	CB0	RTS	
							; A paraméterek beolvasása
C1DB	20	FD	AE	2460	PARAM	JSR CHKCOM	; Vessző ellenőrzése
C1DE	20	8B	B0	2470		JSR \$B08B	; A változó megkeresése/létrehozása
C1E1	85	49		2480		STA \$49	; A változó címe, LB
C1E3	84	4A		2490		STY \$4A	; " , HB
C1E5	A0	02		2500		LDY #\$02	; A mutatók
C1E7	B1	49		2510	CA0	LDA (\$49),Y	; áthelyezése
C1E9	99	61	00	2520		STA \$0061,Y	; a RAM-területre
C1EC	88			2530		DEY	
C1ED	10	F8		2540		BPL CA0	
C1EF	AA			2550		TAX	; A változó eredeti hossza
C1F0	86	B2		2560		STX LEN	; a hosszmutatóba
C1F2	F0	0F		2570		BEQ CE4	; Ha nulla, átugrani a másolást
C1F4	C8			2580		INY	
C1F5	B1	62		2590	CA1	LDA (\$62),Y	; A változó tartalmának
C1F7	99	54	C2	2600		STA BUF,Y	; bemásolása a pufferbe
C1FA	C8			2610		INY	
C1FB	C4	61		2620		CPY \$61	; Elérte a változó hosszát ?
C1FD	F0	04		2630		BEQ CE4	; Igen, ugrás
C1FF	C0	28		2640		CPY #40	; Elérte a 40 karaktert ?
C201	90	F2		2650		BCC CA1	; Még nem, vissza
C203	A5	45		2660	CE4	LDA \$45	; A változó
C205	85	FD		2670		STA \$FD	; nevének
C207	A5	46		2680		LDA \$46	; átmeneti
C209	85	FE		2690		STA \$FE	; tárolása
C20B	20	F1	B7	2700	CA2	JSR CHKGET	; Az X paraméter beolvasása (sor)
C20E	E0	19		2710		CPX #25	; Nagyobb, mint 24 ?
C210	B0	3F		2720		BCS ILL1	; Igen, hiba
C212	86	9E		2730		STX XPOS	; A paraméter tárolása
C214	20	F1	B7	2740		JSR CHKGET	; Az Y paraméter beolvasása (oszlop)

C217 E0 28	2750	CPX #40	; Nagyobb, mint 39 ?
C219 B0 36	2760	BCS ILL1	; Igen, hiba
C21B 86 9F	2770	STX YPOS	; A paraméter tárolása
C21D 20 F1 B7	2780	JSR CHKGET	; A hossz beolvasása
C220 8A	2790	TXA	; Ellenőrzés, hogy
C221 F0 2E	2800	BEQ ILL1	; a mező a sor fizikai
C223 18	2810	CLC	; Keretein belül marad-e
C224 65 9F	2820	ADC YPOS	
C226 C9 28	2830	CMP #40	; Nagyobb, mint 39 ?
C228 B0 27	2840	BCS ILL1	; Igen, hiba
C22A E4 B2	2850	CPX LEN	; Az adat hossza kisebb ?
C22C B0 02	2860	BCS CB5	; Igen, ugrás
C22E 86 B2	2870	STX LEN	; Nem, lehatárolni
C230 86 A6	2880 CB5	STX HOSSZ	; A mező hossza
C232 20 F1 B7	2890	JSR CHKGET	; A típus beolvasása
C235 86 B0	2900	STX TYPE	; és tárolása
C237 A2 00	2910	LDX #00	; Alapérték, 0
C239 20 79 00	2920	JSR CHRGOT	; Az utolsó karakter újraolvasása
C23C C9 2C	2930	CMP #", "	; A karakter vessző ?
C23E D0 0E	2940	BNE CA3	; Nem
C240 20 F1 B7	2950	JSR CHKGET	; A módjelző beolvasása
C243 8A	2960	TXA	
C244 F0 08	2970	BEQ CA3	
C246 C9 01	2980	CMP #01	; 1 ?
C248 D0 07	2990	BNE ILL1	; Nem, hiba
C24A A2 C0	3000	LDX #C0	; 192
C24C D0 00	3010	BNE CA3	
C24E 86 B1	3020 CA3	STX FLAG	; A módjelző beállítása
C250 60	3030	RTS	
C251 4C 48 B2	3040 ILL1	JMP \$B248	; ILLEGAL QUANTITY ERROR
C254	3050 BUF	=*	; Puffer
C27D	3060	*=#+41	; Hossza 40 byte
C27D	3070	.END	

ZEILEN:298 SYMBOLE:68 FEHLER:0

BACK =C13F	BUF =C254	CA0 =C1E7	CA1 =C1F5	CA2 =C20B	CA3 =C24E
CB0 =C1DA	CB1 =C1CF	CB5 =C230	CC0 =C019	CC1 =C029	CC2 =C030
CC3 =C03C	CC4 =C040	CC5 =C047	CC6 =C04E	CC7 =C055	CC8 =C05C
CC9 =C06B	CD0 =C068	CD1 =C099	CD2 =C09C	CD3 =C0B0	CD4 =C0FF
CD5 =C12F	CD6 =C15F	CD8 =C16B	CD9 =C179	CE0 =C18E	CE1 =C1B8
CE2 =C1C1	CE3 =C123	CE4 =C203	CE6 =C0CE	CE7 =C0DD	CE8 =C000
CHKCOM=AEFD	CHKGET=B7F1	CHRGOT=0079	CRL =C0D6	CRL1 =C0DA	CROUT =C175
CRP =C1AC	CRR =C0C5	CRSR =E50C	DEL =C079	ENTRY =C013	ERASE =C1B7
FLAG =00B1	GETIN =F13E	HOSSZ =00A6	ILL1 =C251	INPUT =C000	INST =C0E2
IRQ0 =C182	IRQ1 =C188	LEN =00B2	OUT =C1C2	PARAM =C1DB	POS =00B3
PRINT =FFD2	PUT =C114	PUT1 =C132	PUT2 =C138	RETURN=C142	TYPE =00B0
XPOS =009E	YPOS =009F				

A BASIC-betöltő:

```
100 FOR I= 49152 TO 49747:READ A:POKE I,A:S=S+A:NEXT
110 DATA 32,183,193, 32,219,193, 32,194,193,165
120 DATA 178,133,179, 32,172,193, 32,130,193,169
130 DATA 0,133,204,133,198, 32, 62,241,240,251
140 DATA 201, 34,240,247,201, 13,208, 3, 76, 66
150 DATA 193,201, 95,208, 3, 76, 63,193,201, 44
160 DATA 240, 8,201, 58,240, 4,201, 58,208, 4
170 DATA 36,177, 80,217,201, 20,208, 3, 76,121
180 DATA 192,201, 29,208, 3, 76,197,192,201,148
190 DATA 208, 3, 76,226,192,201,157,208, 3, 76
200 DATA 214,192,166,176,240, 11,201, 32,144,181
210 DATA 201, 95,176,177, 76, 20,193,201, 46,240
220 DATA 249,201, 48,144,166,201, 58,176,162,144
230 DATA 239,165,179,240, 28,197,178,144, 27,208
240 DATA 87, 32,117,193,169,157, 32,210,255,169
250 DATA 32, 32,210,255,169,157, 32,210,255,198
260 DATA 178,198,179, 76, 19,192, 32,117,193,169
270 DATA 157, 32,210,255,164,179, 32,207,193,169
280 DATA 32, 32,210,255,164,179,185, 84,194,153
290 DATA 83,194,200,196,178,208,245,169, 32,153
300 DATA 83,194,198,178, 76,221,192,165,179,197
310 DATA 166,240,206, 32,117,193,230,179, 32,172
320 DATA 193, 76, 19,192,165,179,240,191, 32,117
330 DATA 193,198,179, 76,208,192,165,179,197,178
340 DATA 176,177,165,178,197,166,240,171, 32,117
350 DATA 193,169, 32, 32,210,255,164,179, 32,207
360 DATA 193,198,179,164,178,185, 84,194,153, 85
370 DATA 194,136,196,179,208,245,200,169, 32,153
380 DATA 84,194,230,178,208,186,133,255, 32,117
390 DATA 193,165,179,197,166,240, 16,197,178,176
400 DATA 15,165,255, 32,210,255,164,179,153, 84
410 DATA 194,230,179, 76, 19,192,208, 4,230,178
420 DATA 208,235,166,179,232,134,178,208,228, 32
430 DATA 136,193,162, 0,134, 14,202,134, 13,165
440 DATA 253,133, 69,165,254,133, 70, 32,231,176
450 DATA 133, 73,132, 74,165,178, 32,117,180,160
460 DATA 2,185, 97, 0,145, 73,136, 16,248,170
470 DATA 240, 11,200,185, 84,194,145, 98,200,196
480 DATA 97,208,246,169, 2,133,205,165,207,208
490 DATA 252,169, 1,133,204, 96,169, 0, 72, 72
500 DATA 240, 6,169, 0, 72,169, 1, 72,169, 0
510 DATA 133, 14,133, 13,169, 73,160, 82,133, 69
520 DATA 132, 70, 32,231,176,133, 73,132, 74,104
530 DATA 168,104, 32,145,179, 76,208,187,166,158
540 DATA 165,159, 24,101,179,168, 76, 12,229,169
550 DATA 32,160, 40,153, 84,194,136, 16,250, 96
560 DATA 166,158,164,159, 32, 12,229,165,178,240
570 DATA 13,160, 0,185, 84,194, 32,210,255,200
580 DATA 196,178,208,245, 96, 32,253,174, 32,139
590 DATA 176,133, 73,132, 74,160, 2,177, 73,153
600 DATA 97, 0,136, 16,248,170,134,178,240, 15
610 DATA 200,177, 98,153, 84,194,200,196, 97,240
620 DATA 4,192, 40,144,242,165, 69,133,253,165
630 DATA 70,133,254, 32,241,183,224, 25,176, 63
640 DATA 134,158, 32,241,183,224, 40,176, 54,134
650 DATA 159, 32,241,183,138,240, 46, 24,101,159
660 DATA 201, 40,176, 39,228,178,176, 2,134,178
670 DATA 134,166, 32,241,183,134,176,162, 0, 32
680 DATA 121, 0,201, 44,208, 14, 32,241,183,138
690 DATA 240, 8,201, 1,208, 7,162,192,208, 0
700 DATA 134,177, 96, 76, 72,178
720 IF S <> 84174 THEN PRINT"HIBA A DATASORBAN !":END
```


A rutin megírható BASIC-ben is. Az alábbi lista egy ilyen megoldást mutat be. A rutin a BASIC-jelleg miatt változókat használ a paraméterek tárolására. Ezt és még néhány apróságot leszámítva teljesen kompatibilis a gépi változattal. A rutin szubrutin, ezért GOSUB-bal hívható. A program listája:

```

1000 GOSUB 1630
1010 PRINT A$;
1020 P=LEN(A$)+1
1030 GOSUB 1670
1040 IR=0

1050 POKE 204,0
1060 GET X$: IF X$="" THEN 1060
1070 IF X$=CHR$(34) THEN 1060
1080 IF W=0 AND (X$="," OR X$=";" OR X$=":") THEN 1060
1090 IF X$=CHR$(13) THEN GOSUB 1640:RETURN
1100 IF X$="+" THEN GOSUB 1640:IR=1:RETURN
1110 IF X$=CHR$(20) THEN GOSUB 1200:GOTO 1050
1120 IF X$=CHR$(29) THEN GOSUB 1340:GOTO 1050
1130 IF X$=CHR$(157) THEN GOSUB 1390:GOTO 1050
1140 IF X$=CHR$(148) THEN GOSUB 1440:GOTO 1050
1150 IF M THEN 1180

1160 IF X$<" " OR X$>"+" THEN 1060
1170 GOSUB 1510:GOTO 1050

1180 IF X$<"0" OR X$>"9" THEN 1060
1190 GOSUB 1510:GOTO 1050

1200 IF P=LEN(A$)+1 AND P>1 THEN 1280
1210 IF P>LEN(A$)+1 THEN 1400
1220 IF P=1 THEN RETURN
1230 GOSUB 1640
1240 PRINT CHR$(157)RIGHT$(A$,LEN(A$)-P+1)CHR$(32)
1250 A$=LEFT$(A$,P-2)+RIGHT$(A$,LEN(A$)-P+1)
1260 P=P-1:GOSUB 1670
1270 RETURN

1280 IF LEN(A$)=0 THEN RETURN
1290 A$=LEFT$(A$,LEN(A$)-1)
1300 GOSUB 1640
1310 PRINT CHR$(157)CHR$(32)CHR$(157);
1320 P=P-1
1330 RETURN

1340 IF P>H THEN RETURN
1350 GOSUB 1640
1360 PRINT CHR$(29);
1370 P=P+1
1380 RETURN

1390 IF P=1 THEN RETURN
1400 GOSUB 1640
1410 PRINT CHR$(157);
1420 P=P-1
1430 RETURN
1440 IF P>LEN(A$) THEN RETURN
1450 IF LEN(A$)=H THEN RETURN
1460 GOSUB 1640
1470 PRINT " RIGHT$(A$,LEN(A$)-P+1);
1480 GOSUB 1670
1490 A$=LEFT$(A$,P-1)+" "+RIGHT$(A$,LEN(A$)-P+1)
1500 RETURN

1510 IF P=H+1 THEN RETURN
1520 IF P>LEN(A$) THEN 1580
1530 GOSUB 1640

```

```

1540 PRINT X$;
1550 P=P+1
1560 A$=LEFT$(A$,P-2)+X$+RIGHT$(A$,LEN(A$)-P+1)
1570 RETURN

1580 GOSUB 1640
1590 PRINT X$;
1600 A$=A$+LEFT$(S$,P-LEN(A$)-1)+X$
1610 P=P+1
1620 RETURN

1630 POKE 781,X:POKE 782,Y:SYS 58636:RETURN

1640 POKE 205,2
1650 IF PEEK(207) THEN 1650
1660 POKE 204,1:RETURN

1670 Y1=Y
1680 Y=Y1+P-1:GOSUB 1630
1690 Y=Y1:RETURN

```

4.1.2. Beolvasás állományból — a GET# megkerülése

Az INPUT# karakterláncok, a GET# csak egy karakter beolvasására alkalmas. De az INPUT# is csak 88 karakter beolvasására képes egyszerre. Ez a korlát akadályozhatja a felhasználását, mert egy rekord akár 254 karakter hosszú is lehet. Ha hosszabb adatot kell egyszerre beolvasni, a GET# utasítást kell használni, ami viszont nagyon lassúvá teszi a programot. A problémát kikerülhetjük: olyan gépi rutint írunk, amely tetszőleges hosszúságú rekordot képes azonnal a BASIC változóba helyezni. A rutint az INPUT# gyorsasága és a GET# rugalmassága jellemzi.

A rutin három paramétert használ: a csatorna száma (logikai file-szám), a beolvasás hossza, illetve a változó, amibe az adat kerül. Az utasítás szintaxisa:

SYS tárcím, logikai file-szám, hossz, változó

Például: SYS 49152,3,120,A\$

A rutin assembly listája:

C000	100	*=\$C000					
FFCF	110	INPUT	=\$FFCF				
FFC6	120	CHKIN	=\$FFC6				
FFCC	130	CLRCHN	=\$FFCC				
AEFD	140	CHKCOM	=\$AEFD				
B7F1	150	CHKGET	=\$B7F1				
B6A3	160	FRESTR	=\$B6A3				
C000	20	F1 B7	170	FETCH	JSR	CHKGET	; a csatorna száma
C003	20	C6 FF	180		JSR	CHKIN	; kijelölés inputra
C006	20	F1 B7	190		JSR	CHKGET	; a hossz beolvasása
C009	8A		200		TXA		
C00A	48		210		PHA		; és átmeneti tárolása a veremben
C00B	20	FD AE	220		JSR	CHKCOM	; vessző ellenőrzése
C00E	20	8B B0	230		JSR	\$B08B	; a változó megkeresése
C011	85	49	240		STA	\$49	; a változó címe, LB
C013	84	4A	250		STY	\$4A	; , HB

C015	68	260	PLA		; a hossz visszatöltése
C016	20 75 B4	270	JSR	\$B475	; a szövegmutató kiszámítása
C019	A0 02	280	LDY	#02	
C01B	B9 61 00	290 B3	LDA	\$0061,Y	; a mutatók
C01E	91 49	300	STA	(\$49),Y	; áthelyezése a RAM-területre
C020	88	310	DEY		
C021	10 F8	320	BPL	B3	
C023	C8	330	INY		
C024	20 CF FF	340 B4	JSR	INPUT	; egy karakter a soros buszról
C027	91 62	350	STA	(\$62),Y	; tárolás a változóban
C029	C8	360	INY		
C02A	C4 61	370	CPY	\$61	; elérte a megadott hosszt ?
C02C	D0 F6	380	BNE	B4	; még nem, vissza
C02E	20 CC FF	390	JSR	CLRCHN	; az input csatorna törlése
C031	60	400	RTS		
C032		410	.END		

ZEILEN:32 SYMBOLE:9 FEHLER:0

B3 =C01B B4 =C024 CHKCOM=AEFD CHKGET=B7F1 CHKIN =FFC6 CLRCHN=FFCC
 FETCH =C000 FRESTR=B6A3 INPUT =FFCF

A BASIC-betöltő:

```
100 FOR I= 49152 TO 49201:READ A:POKE I,A:S=S+A:NEXT
110 DATA 32,241,183, 32,198,255, 32,241,183,138
120 DATA 72, 32,253,174, 32,139,176,133, 73,132
130 DATA 74,104, 32,117,180,160, 2,185, 97, 0
140 DATA 145, 73,136, 16,248,200, 32,207,255,145
150 DATA 98,200,196, 97,208,246, 32,204,255, 96
170 IF S <> 6791 THEN PRINT"HIBA A DATASORBAN !":END
```

4.1.3. Karakterlánc-generátor

Az alábbi kis programmal helyet takaríthatunk meg a BASIC-tárban. A program tetszőleges tartalmú és hosszúságú karakterláncot állít elő a megadott változóban. A BASIC-programban az ilyen füzéreket direkt módon kell megadni, és ez sok helyet pazarol a programtárban. A rutin maga nagyon rövid, ezért már két-három szövegdefiníció esetén kifizetődő a használata. A BASIC programtárból egyébként sem használ fel helyet. Az utasítás formátuma:

SYS tárcím, hossz, karakter, változó

Például: SYS 49152,120,32,A\$

A paraméterek:

- (1) *hossz*: 1—255 között változhat
- (2) *karakter*: a lánc egy elemének ASCII-kódja (0—255)
- (3) *változó*: a szöveges célváltozó neve

A rutin assembly listája:


```

C000      100      *=$C000
AEFD      110 CHKCOM =$AEFD
B7F1      120 CHKGET =$B7F1

C000 20 F1 B7      130 TEXT      JSR CHKGET      ; a hosszparaméter beolvasása
C003 8A              140              TXA
C004 48              150              PHA              ; a paraméter átmeneti tárolása
C005 20 F1 B7      160              JSR CHKGET      ; a kód beolvasása
C008 86 02          170              STX $02              ; és tárolása
C00A 20 FD AE      180              JSR CHKCOM      ; vessző ellenőrzése
C00D 20 8B B0      190              JSR $B08B      ; a változó megkeresése
C010 85 49          200              STA $49              ; a változó címe, LB
C012 84 4A          210              STY $4A              ; HB
C014 68              220              PLA              ; a hossz visszatöltése
C015 20 75 B4      230              JSR $B475      ; a szövegmutató kiszámítása
C018 A0 02          240              LDY #$02              ; offset
C01A B9 61 00      250 B1          LDA $0061,Y      ; a változó mutatói
C01D 91 43          260              STA ($49),Y      ; a mutatók a RAM-területre
C01F 88              270              DEY
C020 10 F8          280              BPL B1
C022 C8              290              INY
C023 A5 02          300 B2          LDA $02              ; a kód
C025 91 62          310              STA ($62),Y      ; feltöltés
C027 C8              320              INY
C028 C4 61          330              CPY $61              ; elérte a megadott hosszt ?
C02A D0 F7          340              BNE B2              ; még nem, vissza
C02C 60              350              RTS
C02D              360              .END

```

ZEILEN:27 SYMBOLE:5 FEHLER:0

B1 =C01A B2 =C023 CHKCOM=AEFD CHKGET=B7F1 TEXT =C000

A BASIC-betöltő:

```

100 FOR I= 49152 TO 49196:READ A:POKE I,A:S=S+A:NEXT
110 DATA 32,241,183,138, 72, 32,241,183,134, 2
120 DATA 32,253,174, 32,139,176,133, 73,132, 74
130 DATA 104, 32,117,180,160, 2,185, 97, 0,145
140 DATA 73,136, 16,248,200,165, 2,145, 98,200
150 DATA 196, 97,208,247, 96
170 IF S <> 5625 THEN PRINT"HIBA A DATASORBAN !":END

```

4.1.4. Intarziás keresés adathalmazban

Nagyobb adathalmazban gyakran kell keresni valamilyen adatot. Ezt a keresést programozhatjuk is. Ez először is objektív, megbízható adatot szolgáltat, másodszor nem fárasztja ki a felhasználót. Általában a rekord kis része érdekel bennünket, de sem a helyét, sem a pozícióját nem ismerjük. Az alábbi rutin folyamatosan keres a megadott fűzér tartalmában egy annál rövidebb szöveget. Ha megtalálta, numerikus változóban adja vissza annak pozícióját. Ha nem találta, a változó értéke nulla. Az utasítás formátuma:

SYS tárcím, változó1, változó2, kezdőpozíció, célváltozó

Például: SYS 49152,A\$,B\$,P,X

- (1) *változó1*: a keresés tárgya
- (2) *változó2*: a keresendő szöveg
- (3) *pozíció*: a tárgyszövegben a keresés kezdőpozíciója
- (4) *célváltozó*: ide kerül a keresés eredménye

A rutin az A\$-ban keresi a B\$-t a P pozíciótól kezdődően. Az eredményt az X-ben tárolja.

Ha a megadott szöveg nem fordul elő az A\$-ban, X = 0.

A rutin assembly listája:

C000				100		*= \$C000	
AD9E				110	FRMEVL	= \$AD9E	
AEFD				120	CHKCOM	= \$AEFD	
B7F1				130	CHKGET	= \$B7F1	
B6A3				140	FRESTR	= \$B6A3	
C000	20	FD	AE	150	POS	JSR	CHKCOM ; vessző ellenőrzése
C003	20	9E	AD	160		JSR	FRMEVL ; a Kifejezés Kiértékelése
C006	20	8F	AD	170		JSR	\$AD8F ; alfanumerikus ellenőrzés
C009	A5	64		180		LDA	\$64
C00B	48			190		PHA	; a szöveg címe
C00C	A5	65		200		LDA	\$65
C00E	48			210		PHA	; a verembe
C00F	20	FD	AE	220		JSR	CHKCOM ; vessző ellenőrzése
C012	20	9E	AD	230		JSR	FRMEVL ; a Kifejezés Kiértékelése
C015	20	A3	B6	240		JSR	FRESTR ; szövegkezelés
C018	F0	58		250		BEQ	ILL ; a hossz nulla, hiba
C01A	85	04		260		STA	\$04 ; a hossz tárolása
C01C	86	FB		270		STX	\$FB ; a változó
C01E	84	FC		280		STY	\$FC ; címe
C020	68			290		PLA	
C021	A8			300		TAY	; az előző változó címe vissza
C022	68			310		PLA	
C023	20	AA	B6	320		JSR	\$B6AA ; belépés a FRESTR-be
C026	F0	4A		330		BEQ	ILL ; a hossz nulla, hiba
C028	85	03		340		STA	\$03 ; a hossz tárolása
C02A	86	FD		350		STX	\$FD ; a változó címének
C02C	84	FE		360		STY	\$FE ; tárolása
C02E	20	F1	B7	370		JSR	CHKGET ; a pozíció beolvasása
C031	8A			380		TXA	; a pozíció az akkuba
C032	F0	3E		390		BEQ	ILL ; nulla, hiba
C034	CA			400		DEX	
C035	86	06		410		STX	\$06 ; a pozíció tárolása
C037	A5	03		420		LDA	\$03 ; a második változó hosszából
C039	38			430		SEC	
C03A	E5	04		440		SBC	\$04 ; az első hosszának kivonása
C03C	90	28		450		BCC	J2 ; a Keresendő szöveg hosszabb
C03E	69	00		460		ADC	#\$00 ; a túlcsordulás eltüntetése
C040	85	05		470		STA	\$05 ; a maradék hossza
C042	A5	06		480		LDA	\$06 ; a pozíció
C044	18			490		CLC	
C045	65	FD		500		ADC	\$FD ; hozzáadása a címhez
C047	85	FD		510		STA	\$FD
C049	90	02		520		BCC	J3 ; nem volt túlcsordulás, ugrás
C04B	E6	FE		530		INC	\$FE
C04D	A0	00		540	J3	LDY	#\$00
C04F	B1	FB		550	J4	LDA	(\$FB),Y ; a Keresendő karakterlánc egy tagja
C051	D1	FD		560		CMP	(\$FD),Y ; ellenőrzés a másik szövegben
C053	D0	0B		570		BNE	J5 ; nem egyezik, ugrás
C055	C8			580		INY	
C056	C4	04		590		CPY	\$04 ; a ciklus lefutott ?
C058	90	F5		600		BCC	J4 ; még nem, vissza
C05A	A4	06		610		LDY	\$06 ; a pozíció
C05C	C8			620		INY	
C05D	4C	75	C0	630	J6	JMP	J8

```

C060 E6 06      640 J5      INC $06      ; a pozíció növelése
C062 C6 05      650      DEC $05      ; a visszaszámlálás értéke csökken
C064 D0 04      660      BNE J7       ; még nem nulla, tovább
C066 A0 00      670 J2      LDY #$00    ; a pozíció nulla
C068 F0 F3      680      BEQ J6       ; befejezni
C06A E6 FD      690 J7      INC $FD     ; a mutató
C06C D0 DF      700      BNE J3       ;
C06E E6 FE      710      INC $FE      ; növelése
C070 D0 DB      720      BNE J3       ;

C072 4C 48 B2    730 ILL      JMP $B248    ; ILLEGAL QUANTITY ERROR

C075 A9 00      740 J8      LDA #$00    ; a pozíció felső byte-ja
C077 20 91 B3    750      JSR $B391    ; átalakítás lebegőpontosá
C07A 20 FD AE    760      JSR CHKCOM   ; vessző ellenőrzése
C07D 20 8B B0    770      JSR $B08B    ; a kijelölt változó megkeresése
C080 85 49      780      STA $49      ; a változó címe, LB
C082 84 4A      790      STY $4A      ; , HB
C084 4C D0 BB    800      JMP $BBD0    ; a FAC átvitele a változóba
C087      810      .END

```

ZEILEN:72 SYMBOLE:13 FEHLER:0

CHKCOM=AEFD CHKGET=B7F1 FRESTR=B6A3 FRMEVL=A09E ILL =C072 J2 =C066
J3 =C04D J4 =C04F J5 =C060 J6 =C05D J7 =C06A J8 =C075
POS =C000

A BASIC-betöltő:

```

100 FOR I= 49152 TO 49287:READ A:POKE I,A:S=S+A:NEXT
110 DATA 32,253,174, 32,158,173, 32,143,173,165
120 DATA 100, 72,165,101, 72, 32,253,174, 32,158
130 DATA 173, 32,163,182,240, 88,133, 4,134,251
140 DATA 132,252,104,168,104, 32,170,182,240, 74
150 DATA 133, 3,134,253,132,254, 32,241,183,138
160 DATA 240, 62,202,134, 6,165, 3, 56,229, 4
170 DATA 144, 40,105, 0,133, 5,165, 6, 24,101
180 DATA 253,133,253,144, 2,230,254,160, 0,177
190 DATA 251,209,253,208, 11,200,196, 4,144,245
200 DATA 164, 6,200, 76,117,192,230, 6,198, 5
210 DATA 208, 4,160, 0,240,243,230,253,208,223
220 DATA 230,254,208,219, 76, 72,178,169, 0, 32
230 DATA 145,179, 32,253,174, 32,139,176,133, 73
240 DATA 132, 74, 76,208,187,249
260 IF S <> 18669 THEN PRINT"HIBA A DATASORBAN !":END

```

4.1.5. Pszeudo MID\$ — a standard utasítás módosítása

Az alábbi rutint a C64-es szakirodalmában több helyen is megtalálhatjuk, mint ahogyan az ebben a fejezetben ismertettek közül másokat is. Azonban ezeket a rend kedvéért nem hagyhatjuk ki a felsorolásból.

A MID\$ utasítás eredeti értelme szerint adott szövegből kiemel egy megadott pozíciójú és hosszúságú részletet. Az új utasítás pedig értéket ad ennek a részletnek, de végrehajtja a régi értelmezés szerinti feladatot is. Ha megnézzük az assembly listát, ennek semmi alapja; a régi rutinra nincs utalás. Hogyan lehetséges akkor, hogy mégis használható az eredeti szintaxis is? A megoldást az interpreter feldolgozási rendszere mutatja meg. Az értékadást az interpre-

ter-ciklus hajtja végre, a függvényt pedig a FRMEVL (FROM EVALUATION) rutin értékeli ki. Az interpreter-ciklus az új rutint találja meg, az FRMEVL pedig a ROM-rutint. Az utasítás formátuma:

MID\$ (tárgyszöveg, pozíció, hossz) = füzér

Például: MID\$ (A\$, 3, 2) = "AB"

A bővítést a SYS 49152 utasítással kapcsolhatjuk be, a SYS 49163-mal pedig kikapcsolhatjuk. A rutin assembly listája:

C000	100	*=\$C000	
AD9E	110	FRMEVL	=\$AD9E
0073	120	CHRGET	=\$0073
0079	130	CHRGOT	=\$0079
B7F1	140	CHKGET	=\$B7F1
B6A3	150	FRESTR	=\$B6A3
			; A bővítés bekapcsolása
C000 A9 11	160	EXPAND	LDA #MID<
C002 A0 C0	170		LDY #MID>
C004 8D 03 03	180	G2	STA \$0308
C007 8C 03 03	190		STY \$0309
C00A 60	200		RTS
			; A bővítés kikapcsolása
C00B A9 E4	210	EXPCLR	LDA #\$E4
C00D A0 A7	220		LDY #\$A7
C00F D0 F3	230		BNE G2
			; Az eredeti dekódoló
			; rutin címe
C011 20 73 00	240	MID	JSR CHRGET
C014 C9 CA	250		CMP #\$CA
C016 F0 06	260		BEQ C1
C018 20 79 00	270		JSR CHRGOT
C01B 4C E7 A7	280		JMP \$A7E7
			; A következő karakter beolvasása
			; MID\$ token ?
			; Igen, ugrás
			; Az utolsó karakter újraolvasása
			; Az eredeti dekódoló rutinra
C01E 20 73 00	290	C1	JSR CHRGET
C021 20 FA AE	300		JSR \$AEFA
C024 20 8B B0	310		JSR \$B0BB
C027 85 64	320		STA \$64
C029 84 65	330		STY \$65
C02B 85 49	340		STA \$49
C02D 84 4A	350		STY \$4A
C02F 20 A3 B6	360		JSR FRESTR
C032 A0 00	370		LDY #\$00
C034 B1 64	380		LDA (\$64),Y
C036 48	390		PHA
C037 F0 28	400		BEQ C2
C039 20 52 AA	410		JSR \$AA52
C03C A0 01	420		LDY #\$01
C03E B1 49	430		LDA (\$49),Y
C040 85 05	440		STA \$05
C042 C8	450		INY
C043 B1 49	460		LDA (\$49),Y
C045 85 06	470		STA \$06
C047 20 F1 B7	480		JSR CHKGET
C04A 8A	490		TXA
C04B F0 14	500		BEQ C2
C04D CA	510		DEX
C04E 86 04	520		STX \$04
C050 20 73 00	530		JSR CHRGOT
C053 C9 29	540		CMP #\$29
C055 D0 04	550		BNE C3
C057 A9 FF	560		LDA #\$FF
C059 D0 09	570		BNE C4
C05B 20 F1 B7	580	C3	JSR CHKGET
C05E 8A	590		TXA
			; A hossz nulla ?

C05F D0 03	600	BNE C4	; Nem, ugrás
C061 4C 48 B2	610 C2	JMP \$B248	; ILLEGAL QUANTITY ERROR
C064 85 03	620 C4	STA \$03	; A hossz tárolása
C066 68	630	PLA	; A változó hosszának visszaolvasása
C067 38	640	SEC	
C068 E5 04	650	SBC \$04	; A pozíció levonása
C06A C5 03	660	CMP \$03	; A megadott hossz kisebb ?
C06C B0 02	670	BCS C5	; Igen
C06E 85 03	680	STA \$03	; A hossz lehatárolása
C070 20 F7 AE	690 C5	JSR \$AEF7	; Bezáró zárójel ellenőrzése
C073 A3 B2	700	LDA #\$B2	; Egyenlőségjel token
C075 20 FF AE	710	JSR \$AEFF	; A token ellenőrzése
C078 20 9E AD	720	JSR FRMEVL	; A kifejezés kiértékelése
C07B 20 A3 B6	730	JSR FRESTR	; Szövegkezelés
C07E A0 02	740	LDY #\$02	; A beszúrando szöveg paramétere
C080 B1 64	750	LDA (\$64),Y	; A szövegkifejezés címe, alsó byte
C082 85 51	760	STA \$51	
C084 88	770	DEY	
C085 B1 64	780	LDA (\$64),Y	; A szövegkifejezés címe, felső byte
C087 85 50	790	STA \$50	
C089 88	800	DEY	
C08A B1 64	810	LDA (\$64),Y	; A hossz beolvasása
C08C F0 D3	820	BEQ C2	; A hossz nulla, hiba
C08E C5 03	830	CMP \$03	; Hosszabb, mint a paraméter ?
C090 B0 02	840	BCS C6	; Nem, ugrás
C092 85 03	850	STA \$03	; A paraméter módosítása
C094 A5 05	860 C6	LDA \$05	; A szöveg címe, alsó byte
C096 18	870	CLC	
C097 65 04	880	ADC \$04	; A pozíció hozzáadása
C099 85 05	890	STA \$05	; és tárolása
C09B 90 02	900	BCC C7	; Nincs túlcsordulás, ugrás
C09D E6 06	910	INC \$06	; A felső byte növelése
C09F A4 03	920 C7	LDY \$03	; A hosszparaméter
C0A1 88	930 C8	DEY	
C0A2 B1 50	940	LDA (\$50),Y	; A beszúrando szöveg
C0A4 91 05	950	STA (\$05),Y	; Az eredeti szöveg
C0A6 C0 00	960	CPY #\$00	; A ciklus lefutott ?
C0A8 D0 F7	970	BNE C8	; Még nem, vissza
C0AA 4C AE A7	980	JMP \$A7AE	; Az interpreter ciklusra
C0AD	990	.END	

ZEILEN:90 SYMBOLE:17 FEHLER:0

C1	=C01E	C2	=C061	C3	=C05B	C4	=C064	C5	=C070	C6	=C094
C7	=C09F	C8	=C0A1	CHKGET=B7F1		CHRGET=0073		CHRGOT=0079		EXPAND=C000	
EXPCLR=C00B		FRESTR=B6A3		FRMEVL=AD9E		G2	=C004	M10	=C011		

BASIC-betöltője:

```

100 FOR I= 49152 TO 49324:READ A:POKE I,A:S=S+A:NEXT
110 DATA 169, 17,160,192,141, 8, 3,140, 9, 3
120 DATA 96,169,228,160,167,208,243, 32,115, 0
130 DATA 201,202,240, 6, 32,121, 0, 76,231,167
140 DATA 32,115, 0, 32,250,174, 32,139,176,133
150 DATA 100,132,101,133, 73,132, 74, 32,163,182
160 DATA 160, 0,177,100, 72,240, 40, 32, 82,170
170 DATA 160, 1,177, 73,133, 5,200,177, 73,133
180 DATA 6, 32,241,183,138,240, 20,202,134, 4
190 DATA 32,121, 0,201, 41,208, 4,169,255,208
200 DATA 9, 32,241,183,138,208, 3, 76, 72,178
210 DATA 133, 3,104, 56,229, 4,197, 3,176, 2
220 DATA 133, 3, 32,247,174,169,178, 32,255,174
230 DATA 32,158,173, 32,163,182,160, 2,177,100
240 DATA 133, 81,136,177,100,133, 80,136,177,100
250 DATA 240,211,197, 3,176, 2,133, 3,165, 5
260 DATA 24,101, 4,133, 5,144, 2,230, 6,164
270 DATA 3,136,177, 80,145, 5,192, 0,208,247
280 DATA 76,174,167
300 IF S <> 19819 THEN PRINT"HIBA A DATASORBAN !":END

```


4.1.6. Az USR függvény

Az USR függvény alapállapotban kihasználatlan. A vektortáblázatban levő mutatója (\$0310—11) az ILLEGAL QUANTITY hibaüzenetre mutat. Az USR egyparaméteres numerikus függvény, segítségével a BASIC interpreterben nem szereplő aritmetikai függvényeket (pl. faktoriális) definiálhatunk. Példaprogramunk egy módosított PEEK függvény. A PEEK nem tud hozzáférni a ROM-ok alatti RAM-hoz. Az USR függvény erre képessé tehető. Az USR rutin után egy másik szubrutin található, amely a ROM-ok alatti RAM írását oldja meg. Ez tulajdonképpen csak a CHAREN alatti RAM esetében érdekes, mert az írás általában a RAM-ba irányul. Az USR rutin hívása:

USR (*tárcím*)

Például: A = USR (53248)

A WRITE rutin hívása:

SYS 720, *tárcím*, *érték*

Például: SYS 720,53248,192

A rutin assembly listája:

02A6	100	*=\$02A6	
AEFD	110	CHKCOM	=\$AEFD
B7F7	120	GETADR	=\$B7F7
02A6 A5 14	130	USRCMD LDA \$14	; a nulláslap
02A8 48	140	PHA	
02A9 A5 15	150	LDA \$15	; regiszterpárjának mentése a verembe
02AB 48	160	PHA	
02AC 20 F7 B7	170	JSR GETADR	; a paraméter beolvasása (\$14-15)
02AF 78	180	SEI	; a megszakítások letiltása
02B0 A5 01	190	LDA \$01	; processzorport
02B2 29 FC	200	AND #\$FC	; a bitminta beállítása
02B4 85 01	210	STA \$01	; visszaírás
02B6 A0 00	220	LDY #\$00	
02B8 B1 14	230	LDA (\$14),Y	; a byte beolvasása
02BA A8	240	TAY	; tárolás az Y regiszterben
02BB A5 01	250	LDA \$01	
02BD 09 03	260	ORA #\$03	; a processzorport visszaállítása
02BF 85 01	270	STA \$01	
02C1 58	280	CLI	; a megszakítások engedélyezése
02C2 68	290	PLA	
02C3 85 15	300	STA \$15	; a regiszterek
02C5 68	310	PLA	
02C6 85 14	320	STA \$14	; visszatöltése
02C8 4C A2 B3	330	JMP \$B3A2	; átalakítás lebegőpontosá
02D0	340	*=720	
02D0 20 FD AE	350	WRITE JSR CHKCOM	; vessző ellenőrzése
02D3 20 EB B7	360	JSR \$B7EB	; GETADR és GETBYT
02D6 78	370	SEI	; a megszakítások tiltása
02D7 A5 01	380	LDA \$01	
02D9 29 FC	390	AND #\$FC	; a processzorport beállítása
02DB 85 01	400	STA \$01	
02DD 8A	410	TXA	; a tárolandó érték


```

02DE A0 00      420      LDY #$00
02E0 91 14      430      STA ($14),Y ; elhelyezés a RAM-ban
02E2 A5 01      440      LDA $01
02E4 09 03      450      ORA #$03      ; a processzorport visszaállítása
02E6 85 01      460      STA $01
02E8 58         470      CLI          ; a megszakítások engedélyezése
02E9 60         480      RTS
02EA          490      .END

```

ZEILEN:40 SYMBOLE:4 FEHLER:0

CHKCOM=AEFD GETADR=B7F7 USRCMD=02A6 WRITE =02D0

A BASIC-betöltő:

```

100 POKE785,166:POKE786,2
110 FOR I= 678 TO 714:READ A:POKE I,A:S=S+A:NEXT
120 DATA 165, 20, 72,165, 21, 72, 32,247,183,120
130 DATA 165, 1, 41,252,133, 1,160, 0,177, 20
140 DATA 168,165, 1, 9, 3,133, 1, 88,104,133
150 DATA 21,104,133, 20, 76,162,179
160 FOR I= 720 TO 745:READ A:POKE I,A:S=S+A:NEXT
170 DATA 32,253,174, 32,235,183,120,165, 1, 41
180 DATA 252,133, 1,138,160, 0,145, 20,165, 1
190 DATA 9, 3,133, 1, 88, 96
200 IF S <> 6128 THEN PRINT"HIBA A DATASORBAN !":END

```

4.1.7. Univerzális menü

Felhasználói programokban igen gyakran előfordul, hogy a program egy csomópontjában több választási lehetőségünk is van. A legtöbb program ilyen esetben menüszerűen felsorolja a választási lehetőségeket — a programozók a menük számtalan változatát találták már ki. A következő példaprogram egy egyszerű, de látványos és könnyen kezelhető menüfajta mutat be. A menü bizonyos határok között tetszőlegesen módosítható. A rutin több menüt is tartalmazhat egyszerre, ezekre a hívó utasításban megadott paraméterrel hivatkozhatunk.

A menü megjelenési formáját tekintve egyszerű. Egy keretben megjelennek a menüpontok. A választás a kurzor le-föl billentyű segítségével oldható meg, és amikor a helyes menüpont látható inverzben, a RETURN lenyomásával közölhetjük döntésünket a programmal. A menü ekkor eltűnik. A kiválasztott menüpont sorszámát a tár 255. (\$FF) címén kaphatjuk meg. A menü tartalmát az assembler listában kell beállítanunk. A menüpontok meghatározásán felül be kell állítanunk a VEC1 — VEC5 vektorokat, és az STxL, STxH vektorokat is.

A rutin hívása: SYS 49152, *paraméter*

Például: SYS 49151,1

A rutin assembly listája:

C000		100	*= \$C000	
FFD2		110	PRINT	= \$FFD2
E50C		120	CRSR	= \$E50C
F13E		130	GETIN	= \$F13E
E544		140	CLR	= \$E544
B7F1		150	CHKGET	= \$B7F1
C000	20 F1 B7	160	MENU	JSR CHKGET ; A paraméter beolvasása
C003	CA	170		DEX
C004	BD 4A C1	180		LDA VEC1,X ; Cím alsó byte vektor
C007	85 FB	190		STA \$FB ; alsó byte
C009	BD 4C C1	200		LDA VEC2,X ; Cím alsó byte vektor
C00C	85 FC	210		STA \$FC ; felső byte
C00E	BD 4E C1	220		LDA VEC3,X ; Cím felső byte vektor
C011	85 FD	230		STA \$FD ; alsó byte
C013	BD 50 C1	240		LDA VEC4,X ; Cím felső byte vektor
C016	85 FE	250		STA \$FE ; felső byte
C018	BD 52 C1	260		LDA VEC5,X ; A menüpontok száma
C01B	85 24	270		STA \$24
C01D	A9 08	280		LDA #\$08 ; A módváltás blokkolása
C01F	20 D2 FF	290		JSR PRINT
C022	A9 0E	300		LDA #\$0E ; A kisbetű-nagybetű mód
C024	20 D2 FF	310		JSR PRINT ; bekapcsolása
C027	20 44 E5	320		JSR CLR ; Képernyőtörölés
C02A	20 38 C0	330		JSR KERET ; A keret kirajzolása
C02D	20 9D C0	340		JSR STRON ; A menüpontok kiírása
C030	A0 00	350		LDY #\$00 ; Az első menüpont
C032	20 B3 C0	360		JSR RVSON ; invertálása
C035	4C 02 C1	370		JMP GET ; A várakozási ciklusra
				; A keret kirajzolása
C038	A5 24	380	KERET	LDA \$24 ; A menüpontok száma
C03A	0A	390		ASL A ; Szorzás kettővel
C03B	18	400		CLC
C03C	69 03	410		ADC #\$03 ; 3 hozzáadása
C03E	85 F7	420		STA \$F7 ; átmeneti tárolás
C040	A9 13	430		LDA #25 ; 25 sor
C042	38	440		SEC
C043	E5 F7	450		SBC \$F7 ; A keret kiterjedésének levonása
C045	4A	460		LSR A ; Osztás kettővel
C046	85 F9	470		STA \$F9 ; A kezdősorszám tárolása
C048	A9 08	480		LDA #\$08 ; A kezdőoszlop
C04A	85 FA	490		STA \$FA ; tárolása
C04C	18	500		CLC
C04D	69 03	510		ADC #\$03 ; 3 hozzáadása
C04F	8D 6C C1	520		STA STC2 ; A menüpontok kezdőoszlopa
C052	A5 F9	530		LDA \$F9 ; A keret kezdősora
C054	18	540		CLC
C055	69 03	550		ADC #\$03 ; 3 hozzáadása
C057	A0 00	560		LDY #\$00
C059	99 62 C1	570	D5	STA STC1,Y ; A menüpontok sorainak koordinátái
C05C	18	580		CLC
C05D	69 02	590		ADC #\$02 ; 2 hozzáadása, a következőt
C05F	C8	600		INY
C060	C4 24	610		CPY \$24 ; Elérte a menüpontok számát ?
C062	90 F5	620		BCC D5 ; Még nem, vissza
C064	A6 F9	630		LDX \$F9 ; A keret bal felső
C066	A4 FA	640		LDY \$FA ; sarkának koordinátái
C068	20 0C E5	650		JSR CRSR ; A kurzor beállítása
C06B	A9 6D	660		LDA #KR1< ; A keret felső
C06D	A0 C1	670		LDY #KR1> ; sorának címe
C06F	20 39 C1	680		JSR STRING ; A szöveg kiírása
C072	E6 F9	690		INC \$F9 ; A sorszámláló növelése
C074	A2 00	700		LDX #\$00
C076	86 F8	710	D3	STX \$F8 ; A ciklusszámláló tárolása
C078	A6 F9	720		LDX \$F9 ; Az aktuális sor
C07A	A4 FA	730		LDY \$FA ; és oszlop
C07C	20 0C E5	740		JSR CRSR ; A kurzor beállítása
C07F	A9 86	750		LDA #KR2< ; A keret

C081	A0	C1	760		LDY #KR2>	; Középső részének
C083	20	39	770	C1	JSR STRING	; Kiírása
C086	E6	F9	780		INC \$F9	; A sorszámláló növelése
C088	A6	F8	790		LDX \$F8	; A ciklusszámláló visszatöltése
C08A	E8		800		INX	; és növelése
C08B	E4	F7	810		CPX \$F7	; Elérte a felső határt ?
C08D	D0	E7	820		BNE D3	; Még nem, vissza
C08F	A6	F9	830		LDX \$F9	
C091	A4	FA	840		LDY \$FA	
C093	20	0C	850	E5	JSR CRSR	; A kurzor beállítása
C096	A9	9F	860		LDA #KR3<	; A keret
C098	A0	C1	870		LDY #KR3>	; alsó sorának
C09A	4C	39	880	C1	JMP STRING	; Kiírása
						; A menüpontok kiírása
C09D	A0	00	890	STRON	LDY #\$00	
C09F	84	FF	900	D4	STY \$FF	; A ciklusszámláló tárolása
C0A1	BE	62	910	C1	LDX STC1,Y	; Az aktuális sor
C0A4	AC	6C	920	C1	LDY STC2	; és oszlop
C0A7	20	0C	930	E5	JSR CRSR	; A kurzor beállítása
C0AA	20	EE	940	C0	JSR D2	; A menüpont kiírása
C0AD	C8		950		INY	
C0AE	C4	24	960		CPY \$24	; Elérte a felső határt ?
C0B0	D0	ED	970		BNE D4	; Még nem, vissza
C0B2	60		980		RTS	
						; A menüpont invertálása
C0B3	84	FF	990	RVSON	STY \$FF	; A mutató tárolása
C0B5	A9	04	1000		LDA #LINE<	; A menüpont fölötti
C0B7	A0	C2	1010		LDY #LINE>	; vonal címe
C0B9	A2	12	1020		LDX #\$12	; RVS ON karakter
C0BB	D0	08	1030		BNE D1	; Feltétel nélküli ugrás
C0BD	84	FF	1040	RVS OFF	STY \$FF	; A mutató tárolása
C0BF	A9	17	1050		LDA #SPC<	; Az üres sor
C0C1	A0	C2	1060		LDY #SPC>	; címe
C0C3	A2	92	1070		LDX #\$92	; RVS OFF karakter
C0C5	86	26	1080	D1	STX \$26	; A regiszterek
C0C7	85	22	1090		STA \$22	; átmeneti
C0C9	84	23	1100		STY \$23	; tárolása
C0CB	A4	FF	1110		LDY \$FF	; A mutató visszatöltése
C0CD	B3	62	1120	C1	LDA STC1,Y	; A menüpont sorának koordinátája
C0D0	AA		1130		TAX	
C0D1	48		1140		PHA	; Tárolás a veremben
C0D2	AC	6C	1150	C1	LDY STC2	; A menüpont oszlopának koordinátája
C0D5	98		1160		TYA	
C0D6	48		1170		PHA	; Tárolás a veremben
C0D7	CA		1180		DEX	; A sorszám csökkentése eggyel
C0D8	20	0C	1190	E5	JSR CRSR	; A kurzor beállítása
C0DB	A5	22	1200		LDA \$22	; A kiírandó szöveg címe
C0DD	A4	23	1210		LDY \$23	; vagy a vonal vagy az üres sor
C0DF	20	39	1220	C1	JSR STRING	; A szöveg kiírása
C0E2	68		1230		PLA	; A koordináták
C0E3	A8		1240		TAY	; visszatöltése
C0E4	68		1250		PLA	; a veremből
C0E5	AA		1260		TAX	
C0E6	20	0C	1270	E5	JSR CRSR	; A kurzor beállítása
C0E9	A5	26	1280		LDA \$26	; A REVERSE mód
C0EB	20	D2	1290	FF	JSR PRINT	; beállítása/törölése
C0EE	A4	FF	1300	D2	LDY \$FF	; A mutató visszatöltése
C0F0	B1	FB	1310		LDA (\$FB),Y	; A szöveg címének alsó byte-ja
C0F2	48		1320		PHA	; Átmeneti tárolás a veremben
C0F3	B1	FD	1330		LDA (\$FD),Y	; A szöveg címének felső byte-ja
C0F5	A8		1340		TAY	; az Y regiszterbe
C0F6	68		1350		PLA	; Az alsó byte visszaolvasása
C0F7	20	39	1360	C1	JSR STRING	; A megcímezett szöveg kiírása
C0FA	A9	92	1370		LDA #\$92	; RVS OFF
C0FC	20	D2	1380	FF	JSR PRINT	; Kiírása
C0FF	A4	FF	1390		LDY \$FF	; A menüpont mutatója
C101	60		1400		RTS	
						; Várakozási ciklus
C102	20	3E	1410	GET	JSR GETIN	; Egy karakter a billentyűzetről
C105	F0	FB	1420		BEQ GET	; Nincs karakter, várakozni

C107	C9	11		1430	CMP	#\$11	; Kurzor le ?
C109	F0	0D		1440	BEQ	LE	; Igen
C10B	C9	91		1450	CMP	#\$91	; Kurzor föl ?
C10D	F0	1B		1460	BEQ	FOL	; Igen
C10F	C9	0D		1470	CMP	#\$0D	; Return ?
C111	D0	EF		1480	BNE	GET	; Nem, vissza
C113	E6	FF		1490	INC	\$\$\$; A mutató beállítása
C115	4C	44	E5	1500	JMP	CLR	; A képernyő törlése, vége
C118	A4	FF		1510	LDY	\$\$\$; A menüpont mutatója
C11A	20	BD	C0	1520	JSR	RVSOF	; A reverse mód kikapcsolása
C11D	C8			1530	INY		; A mutató növelése
C11E	C4	24		1540	CPY	24	; Elérte a maximumot ?
C120	90	02		1550	BCC	G1	; Még nem
C122	A0	00		1560	LDY	\$\$\$; Az első menüpont mutatója
C124	20	B3	C0	1570	JSR	RVSOF	; A menüpont invertálása
C127	4C	02	C1	1580	JMP	GET	; Vissza a várakozási ciklushoz
C12A	A4	FF		1590	LDY	\$\$\$; A menüpont mutatója
C12C	20	BD	C0	1600	JSR	RVSOF	; A reverse mód kikapcsolása
C12F	88			1610	DEY		; A mutató csökkentése
C130	C0	FF		1620	CPY	\$\$\$; Alulcsordult ?
C132	D0	F0		1630	BNE	G1	; Még nem, ugrás
C134	A4	24		1640	LDY	24	; Az utolsó menüpont mutatója
C136	88			1650	DEY		
C137	D0	EB		1660	BNE	G1	; Feltétel nélküli ugrás
C133	85	20		1670	STA	20	; Cím alsó
C13B	84	21		1680	STY	21	; Cím felső
C13D	A0	00		1690	LDY	\$\$\$	
C13F	B1	20		1700	LDA	(\$20),Y	; Egy karakter beolvasása
C141	F0	06		1710	BEQ	AC0	; Szöveg vége ?
C143	20	D2	FF	1720	JSR	PRINT	; A karakter kiírása
C146	C8			1730	INY		
C147	D0	F6		1740	BNE	AC1	; Folytatni
C149	60			1750	RTS		
C14A	54	5C		1760	.BYTE	ST1L<,ST2L<	; Vektor operátorok
C14C	C1	C1		1770	.BYTE	ST1L>,ST2L>	
C14E	58	5F		1780	.BYTE	ST1H<,ST2H<	
C150	C1	C1		1790	.BYTE	ST1H>,ST2H>	
C152	04	03		1800	.BYTE	4,3	; A menüpontok száma
C154	B8	CB	DE	1810	.BYTE	ST1<,ST2<,ST3<,ST4<	; Menüpont vektorok
C158	C1	C1	C1	1820	.BYTE	ST1>,ST2>,ST3>,ST4>	
C15C	2A	3D	50	1830	.BYTE	ST5<,ST6<,ST7<	
C15F	C2	C2	C2	1840	.BYTE	ST5>,ST6>,ST7>	
C16C				1850	STC1	*=**+10	; Sorkoordináta verem
C16C	0B			1860	STC2	.BYTE 11	; Oszlopkoordináta
C16D	B0	C0	C0	1870	KR1	.TEXT" _____"	; A Keret
C185	00			1880	.BYTE	00	
C186	DD	20	20	1890	KR2	.TEXT" _____ "	
C19E	00			1900	.BYTE	00	
C19F	AD	C0	C0	1910	KR3	.TEXT" _____"	
C1B7	00			1920	.BYTE	00	
C1B8	20	C6	49	1930	ST1	.TEXT" File szerkesztes "	; Menüpontok
C1CA	00			1940	.BYTE	00	
C1CB	20	20	20	1950	ST2	.TEXT" Karbantartas "	
C1DD	00			1960	.BYTE	00	
C1DE	20	20	CC	1970	ST3	.TEXT" Lemezmuveletek "	
C1F0	00			1980	.BYTE	00	
C1F1	20	20	20	1990	ST4	.TEXT" Fomenue "	
C203	00			2000	.BYTE	00	
C204	A4	A4	A4	2010	LINE	.TEXT" _____"	; A vonal és a törlő sor
C216	00			2020	.BYTE	00	

```

C217 20 20 20      2030 SPC      .TEXT"          "
C229 00              2040      .BYTE $00

                                ; Menüpontok
C22A 20 20 20      2050 ST5     .TEXT"      Adatbevitel  "
C23C 00              2060      .BYTE $00
C23D 20 20 20      2070 ST6     .TEXT"      Listazo      "
C24F 00              2080      .BYTE $00
C250 20 20 20      2090 ST7     .TEXT"      Menu          "
C262 00              2100      .BYTE $00
C263                2110      .END

```

ZEILEN:202 SYMBOLE:45 FEHLER:0

AC0	=C149	AC1	=C13F	CHKGET=B7F1	CLR	=E544	CRSR	=E50C	D1	=C0C5	
D2	=C0EE	D3	=C076	D4	=C09F	D5	=C059	FOL	=C12A	G1	=C124
GET	=C102	GETIN	=F13E	KERET	=C038	KR1	=C160	KR2	=C186	KR3	=C19F
LE	=C118	LINE	=C204	MENU	=C000	PRINT	=FFD2	RVSOFF	=C0BD	RVSON	=C0B3
SPC	=C217	ST1	=C1B8	ST1H	=C158	ST1L	=C154	ST2	=C1CB	ST2H	=C15F
ST2L	=C15C	ST3	=C1DE	ST4	=C1F1	ST5	=C22A	ST6	=C23D	ST7	=C250
STC1	=C162	STC2	=C16C	STRING=C139	STRON	=C09D	VEC1	=C14A	VEC2	=C14C	
VEC3	=C14E	VEC4	=C150	VEC5	=C152						

A BASIC-betöltő:

```

100 FOR I= 49152 TO 49768 :READ X:POKE I,X:S=S+X:NEXT
110 DATA 32,241,183,202,189, 74,193,133,251,189, 76,193
120 DATA 133,252,189, 78,193,133,253,189, 80,193,133,254
130 DATA 189, 82,193,133, 36,169, 8, 32,210,255,169, 14
140 DATA 32,210,255, 32, 68,229, 32, 56,192, 32,157,192
150 DATA 160, 0, 32,179,192, 76, 2,193,165, 36, 10, 24
160 DATA 105, 3,133,247,169, 25, 56,229,247, 74,133,249
170 DATA 169, 8,133,250, 24,105, 3,141,108,193,165,249
180 DATA 24,105, 3,160, 0,153, 98,193, 24,105, 2,200
190 DATA 196, 36,144,245,166,249,164,250, 32, 12,229,169
200 DATA 109,160,193, 32, 57,193,230,249,162, 0,134,248
210 DATA 166,249,164,250, 32, 12,229,169,134,160,193, 32
220 DATA 57,193,230,249,166,248,232,228,247,208,231,166
230 DATA 249,164,250, 32, 12,229,169,159,160,193, 76, 57
240 DATA 193,160, 0,132,255,190, 98,193,172,108,193, 32
250 DATA 12,229, 32,238,192,200,196, 36,208,237, 96,132
260 DATA 255,169, 4,160,194,162, 18,208, 8,132,255,169
270 DATA 23,160,194,162,146,134, 38,133, 34,132, 35,164
280 DATA 255,185, 98,193,170, 72,172,108,193,152, 72,202
290 DATA 32, 12,229,165, 34,164, 35, 32, 57,193,104,168
300 DATA 104,170, 32, 12,229,165, 38, 32,210,255,164,255
310 DATA 177,251, 72,177,253,168,104, 32, 57,193,169,146
320 DATA 32,210,255,164,255, 96, 32, 62,241,240,251,201
330 DATA 17,240, 13,201,145,240, 27,201, 13,208,233,230
340 DATA 255, 76, 68,229,164,255, 32,189,192,200,196, 36
350 DATA 144, 2,160, 0, 32,179,192, 76, 2,193,164,255
360 DATA 32,189,192,136,192,255,208,240,164, 36,136,208
370 DATA 235,133, 32,132, 33,160, 0,177, 32,240, 6, 32
380 DATA 210,255,200,208,246, 96, 84, 92,193,193, 88, 95
390 DATA 193,193, 4, 3,184,203,222,241,193,193,193,193
400 DATA 42, 61, 80,194,194,194, 11, 13, 15, 16,244, 32
410 DATA 171,255, 76, 13, 11,176,192,192,192,192,192,192
420 DATA 192,192,192,192,192,192,192,192,192,192,192,192
430 DATA 192,192,192,192,174, 0,221, 32, 32, 32, 32, 32
440 DATA 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32
450 DATA 32, 32, 32, 32, 32,221, 0,173,192,192,192,192
460 DATA 192,192,192,192,192,192,192,192,192,192,192,192
470 DATA 192,192,192,192,192,192,189, 0, 32,198, 73, 76
480 DATA 69, 32, 63, 90, 69, 82, 75, 69, 83, 90, 84, 69
490 DATA 83, 32, 0, 32, 32, 32,203, 65, 82, 66, 65, 78
500 DATA 84, 65, 82, 84, 65, 83, 32, 32, 32, 0, 32, 32
510 DATA 204, 69, 77, 69, 90, 77, 85, 86, 69, 76, 69, 84
520 DATA 69, 75, 32, 32, 0, 32, 32, 32, 32, 32, 32,198
530 DATA 79, 77, 69, 78, 85, 32, 32, 32, 32, 32, 32, 0
540 DATA 164,164,164,164,164,164,164,164,164,164,164,164
550 DATA 164,164,164,164,164,164, 0, 32, 32, 32, 32, 32

```



```

560 DATA 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32
570 DATA 32, 0, 32, 32, 32, 193, 68, 65, 84, 66, 69, 86
580 DATA 73, 84, 69, 76, 32, 32, 32, 32, 0, 32, 32, 32
590 DATA 32, 32, 204, 73, 83, 84, 65, 90, 79, 32, 32, 32
600 DATA 32, 32, 32, 0, 32, 32, 32, 32, 32, 32, 32, 205
610 DATA 69, 78, 85, 32, 32, 32, 32, 32, 32, 32, 0, 162
620 DATA 109, 160, 194, 32, 189
630 IF S <> 73759 THEN PRINT "HIBA A DATASORBAN !":END
640 PRINT "OK !!":END

```

4.1.8. Magyar ékezetes karakterek

A C64-es alapkiépítésben a szabványnak számító angolszász ABC-t alkalmazza. A betűk bitmintáját a CHAREN ROM tárolja. A C64-es hardvere viszont megengedi a CHAREN-terület (\$D000—DFFF) átcímzését a RAM-ba. Ez lehetőséget nyújt arra, hogy a karaktergenerátor tartalmát saját elképzelésünk szerint módosítsuk.

A módosítással kapcsolatos teendőket a VIC leírásánál már ismertettük, ezért itt csak a gyakorlati tennivalókat tárgyaljuk.

1. Másoljuk le a karaktergenerátor ROM-ot az alatta levő RAM-ba.
2. Módosítsuk a másolatot a megfelelő helyeken.
3. Változtassuk meg a CIA2 A portjának (\$DD00) 0—1. bitjét.
4. Közöljük az interpreterrel a képernyő-memória kezdőcímét.

Ez utóbbi a tár 648-as címén levő mutatóval lehetséges. A továbbiakban három szubrutin végzi a másolást és a be- illetve kikapcsolást. A rutinok hívása:

- másolás: SYS 49152
- bekapcsolás: SYS 49155
- kikapcsolás: SYS 49154

Az ékezetes karakterek csak a kisbetű-nagybetű üzemmódban láthatók. A karakterek az alábbi billentyűk hatására jelennek meg:

- á: Com. + M
- é: Com. + £
- í: Com. + K
- ó: Com. + B
- ö: Com. + N
- ő: Com. + @
- ú: Shift + @
- ü: Com. + Q
- ű: Com. + G
- ä: Com. + Y
- Á: Com. + A
- É: Com. + E
- Í: Com. + R

Ó: Com. + W

Ö: Com. + S

Ő: Com. + I

Ú: Com. + H

Û: Com. + P

Ű: Com. + L

A rutinok assembly listái:

C000		100	*= \$C000	
C000	4C 09 C0	110	JMP COPY	; a CHAREN lemásolása és módosítása
C003	4C 68 C1	120	JMP ON	; átkapcsolás a módosított CHAREN-re
C006	4C 81 C1	130	JMP OFF	; visszakapcsolás normál üzemmódra
C009	78	140	COPY SEI	; a megszakítások letiltása
C00A	A5 01	150	LDA \$01	; a tárfelosztás módosítása
C00C	29 FB	160	AND #\$FB	; \$D000-DFFF, CHAREN
C00E	85 01	170	STA \$01	; írásra : RAM , olvasásra : ROM
C010	A9 00	180	LDA #\$00	
C012	85 FD	190	STA \$FD	; a kezdőcím
C014	A9 D0	200	LDA #\$D0	
C016	85 FE	210	STA \$FE	; beállítása
C018	A2 10	220	LDX #\$10	; 16 lap
C01A	A0 00	230	LDY #\$00	
C01C	B1 FD	240	AA0 LDA (\$FD),Y	; a CHAREN ROM
C01E	91 FD	250	STA (\$FD),Y	
C020	C8	260	INY	; lemásolása a
C021	D0 F9	270	BNE AA0	
C023	E6 FE	280	INC \$FE	; RAM-ba
C025	CA	290	DEX	
C026	D0 F4	300	BNE AA0	
C028	A9 98	310	LDA #CHAR<	; a mutatók
C02A	85 FD	320	STA \$FD	
C02C	A9 C0	330	LDA #CHAR>	; táblázata
C02E	85 FE	340	STA \$FE	
C030	A0 00	350	LDY #\$00	
C032	B9 7E C0	360	AA3 LDA TBL1,Y	; a Karakter bitmintájának mutatója
C035	84 FF	370	STY \$FF	; a táblázaton belüli mutató tárolása
C037	A0 00	380	LDY #\$00	
C039	84 23	390	STY \$23	
C03B	0A	400	ASL A	; a Karakter
C03C	26 23	410	ROL \$23	
C03E	0A	420	ASL A	; címének kiszámítása
C03F	26 23	430	ROL \$23	
C041	0A	440	ASL A	; a mutató szorzása 8-cal
C042	26 23	450	ROL \$23	
C044	85 22	460	STA \$22	
C046	A5 23	470	LDA \$23	
C048	18	480	CLC	
C049	69 D8	490	ADC #\$D8	; cím offset, Kisbetű/nagybetű
C04B	85 23	500	STA \$23	
C04D	69 04	510	ADC #\$04	; cím offset, reverse terület
C04F	85 25	520	STA \$25	
C051	A5 22	530	LDA \$22	; az alsó byte-ok
C053	85 24	540	STA \$24	; azonosak
C055	A0 00	550	LDY #\$00	
C057	B1 FD	560	AA1 LDA (\$FD),Y	; a bitminta egy byte-ja
C059	91 22	570	STA (\$22),Y	; az új CHAREN-be
C05B	49 FF	580	EOR #\$FF	; minden bit megfordítása, reverse
C05D	91 24	590	STA (\$24),Y	; a CHAREN-be
C05F	C8	600	INY	
C060	C0 08	610	CPY #\$08	; a nyolcadik byte ?
C062	D0 F3	620	BNE AA1	; még nem, vissza
C064	A5 FD	630	LDA \$FD	
C066	18	640	CLC	; a cím
C067	69 08	650	ADC #\$08	; növelése 8-cal

C069	85	FD		660		STA \$FD	
C06B	90	02		670		BCC AA2	
C06D	E6	FE		680		INC \$FE	
C06F	A4	FF		690	AA2	LDY \$FF	; a mutató visszatöltése
C071	C8			700		INY	
C072	C0	1A		710		CPY #\$1A	; az utolsó karakter ?
C074	D0	BC		720		BNE AA3	; még nem, vissza
C076	A5	01		730		LDA \$01	
C078	09	04		740		ORA #\$04	; a tárfelosztás visszaállítása
C07A	85	01		750		STA \$01	
C07C	58			760		CLI	; a megszakítások engedélyezése
C07D	60			770		RTS	
C07E	6A	6B	67	780	TBL1	.BYTE \$6A,\$6B,\$67,\$68,\$61,\$7F,\$7A,\$64,\$65	
C087	6E	6F	70	790		.BYTE \$6E,\$6F,\$70,\$71,\$72,\$73,\$74,\$75,\$76	
C090	7E	7B	7C	800		.BYTE \$7E,\$7B,\$7C,\$78,\$79,\$6D,\$1C,\$77	
C098	00	66	00	810	CHAR	.BYTE \$00,\$66,\$00,\$3C,\$66,\$66,\$3C,\$00	
C0A0	00	66	00	820		.BYTE \$00,\$66,\$00,\$66,\$66,\$66,\$3E,\$00	
C0A8	06	0C	3C	830		.BYTE \$06,\$0C,\$3C,\$06,\$3E,\$66,\$3E,\$00	
C0B0	06	0C	3C	840		.BYTE \$06,\$0C,\$3C,\$66,\$7E,\$60,\$3C,\$00	
C0B8	0C	18	00	850		.BYTE \$0C,\$18,\$00,\$38,\$18,\$18,\$3C,\$00	
C0C0	06	0C	00	860		.BYTE \$06,\$0C,\$00,\$3C,\$66,\$66,\$3C,\$00	
C0C8	0C	18	00	870		.BYTE \$0C,\$18,\$00,\$66,\$66,\$66,\$3E,\$00	
C0D0	1B	36	00	880		.BYTE \$1B,\$36,\$00,\$3C,\$66,\$66,\$3C,\$00	
C0D8	1B	36	00	890		.BYTE \$1B,\$36,\$00,\$66,\$66,\$66,\$3E,\$00	
C0E0	66	00	3C	900		.BYTE \$66,\$00,\$3C,\$66,\$66,\$66,\$3C,\$00	
C0E8	66	00	66	910		.BYTE \$66,\$00,\$66,\$66,\$66,\$66,\$3C,\$00	
C0F0	06	0C	3C	920		.BYTE \$06,\$0C,\$3C,\$66,\$7E,\$66,\$66,\$00	
C0F8	0C	18	7E	930		.BYTE \$0C,\$18,\$7E,\$60,\$78,\$60,\$7E,\$00	
C100	06	08	3C	940		.BYTE \$06,\$08,\$3C,\$18,\$18,\$18,\$3C,\$00	
C108	06	0C	3C	950		.BYTE \$06,\$0C,\$3C,\$66,\$66,\$66,\$3C,\$00	
C110	0C	18	66	960		.BYTE \$0C,\$18,\$66,\$66,\$66,\$66,\$3C,\$00	
C118	1B	14	3C	970		.BYTE \$1B,\$14,\$3C,\$66,\$66,\$66,\$3C,\$00	
C120	33	22	66	980		.BYTE \$33,\$22,\$66,\$66,\$66,\$66,\$3C,\$00	
C128	00	00	00	990		.BYTE \$00,\$00,\$00,\$1F,\$1F,\$18,\$18,\$18	
C130	00	00	00	1000		.BYTE \$00,\$00,\$00,\$F8,\$F8,\$18,\$18,\$18	
C138	18	18	18	1010		.BYTE \$18,\$18,\$18,\$1F,\$1F,\$00,\$00,\$00	
C140	00	00	00	1020		.BYTE \$00,\$00,\$00,\$FF,\$FF,\$00,\$00,\$00	
C148	00	08	0C	1030		.BYTE \$00,\$08,\$0C,\$FE,\$FE,\$0C,\$08,\$00	
C150	00	00	00	1040		.BYTE \$00,\$00,\$00,\$00,\$00,\$00,\$00,\$FF	
C158	1C	30	3C	1050		.BYTE \$1C,\$30,\$3C,\$66,\$66,\$3C,\$0C,\$38	
C160	66	00	3C	1060		.BYTE \$66,\$00,\$3C,\$06,\$3E,\$66,\$3E,\$00	
C168	AD	11	D0	1070	ON	LDA \$D011	; VIC 1. ellenőrző-vezérlő regiszter
C16B	29	EF		1080		AND #\$EF	; %11101111, a Képernyő Kikapcsolása
C16D	8D	11	D0	1090		STA \$D011	
C170	AD	00	DD	1100		LDA \$DD00	; CIA2 A port
C173	29	FC		1110		AND #\$FC	; az alsó két bit törlése
C175	8D	00	DD	1120		STA \$DD00	; VA14-15 magas, a 4. 16K-s lap
C178	A9	CC		1130		LDA #\$CC	; a 204. lap
C17A	8D	88	02	1140		STA \$0288	; a Képernyőmemória címe (lap)
C17D	A9	37		1150		LDA #\$37	; a báziscím értéke
C17F	D0	17		1160		BNE AA5	; feltétel nélküli ugrás
C181	AD	11	D0	1170	OFF	LDA \$D011	
C184	29	EF		1180		AND #\$EF	; a Képernyő Kikapcsolása
C186	8D	11	D0	1190		STA \$D011	
C189	AD	00	DD	1200		LDA \$DD00	
C18C	09	03		1210		ORA #\$03	; VA14-15 alacsony, az 1. 16K-s lap
C18E	8D	00	DD	1220		STA \$DD00	
C191	A9	04		1230		LDA #\$04	; a 4. lap
C193	8D	88	02	1240		STA \$0288	; a Képernyőmemória Kezdőcíme (lap)
C196	A9	17		1250		LDA #\$17	; a báziscím
C198	8D	18	D0	1260	AA5	STA \$D018	; a CHAREN és a Képernyőtár báziscíme
C19B	20	44	E5	1270		JSR \$E544	; Képernyőtörlés
C19E	AD	11	D0	1280		LDA \$D011	
C1A1	09	10		1290		ORA #\$10	; a Képernyő visszaKapcsolása
C1A3	8D	11	D0	1300		STA \$D011	
C1A6	60			1310		RTS	
C1A7				1320		.END	

ZEILEN:123 SYMBOLE:10 FEHLER:0

AA0	=C01C	AA1	=C057	AA2	=C06F	AA3	=C032	AA5	=C196	CHAR	=C096
COPY	=C009	OFF	=C181	ON	=C168	TBL1	=C07E				

A program BASIC-betöltője:

```

100 FOR I= 49152 TO 49574:READ A:POKE I,A:S=S+A:NEXT
110 DATA 76, 9,192, 76,104,193, 76,129,193,120
120 DATA 165, 1, 41,251,133, 1,169, 0,133,253
130 DATA 169,208,133,254,162, 16,160, 0,177,253
140 DATA 145,253,200,208,249,230,254,202,208,244
150 DATA 169,152,133,253,169,192,133,254,160, 0
160 DATA 185,126,192,132,255,160, 0,132, 35, 10
170 DATA 38, 35, 10, 38, 35, 10, 38, 35,133, 34
180 DATA 165, 35, 24,105,216,133, 35,105, 4,133
190 DATA 37,165, 34,133, 36,160, 0,177,253,145
200 DATA 34, 73,255,145, 36,200,192, 8,208,243
210 DATA 165,253, 24,105, 8,133,253,144, 2,230
220 DATA 254,164,255,200,192, 26,208,188,165, 1
230 DATA 9, 4,133, 1, 88, 96,106,107,103,104
240 DATA 97,127,122,100,101,110,111,112,113,114
250 DATA 115,116,117,118,126,123,124,120,121,109
260 DATA 28,119, 0,102, 0, 60,102,102, 60, 0
270 DATA 0,102, 0,102,102,102, 62, 0, 6, 12
280 DATA 60, 6, 62,102, 62, 0, 6, 12, 60,102
290 DATA 126, 96, 60, 0, 12, 24, 0, 56, 24, 24
300 DATA 60, 0, 6, 12, 0, 60,102,102, 60, 0
310 DATA 12, 24, 0,102,102,102, 62, 0, 27, 54
320 DATA 0, 60,102,102, 60, 0, 27, 54, 0,102
330 DATA 102,102, 62, 0,102, 0, 60,102,102,102
340 DATA 60, 0,102, 0,102,102,102,102, 60, 0
350 DATA 6, 12, 60,102,126,102,102, 0, 12, 24
360 DATA 126, 96,120, 96,126, 0, 6, 8, 60, 24
370 DATA 24, 24, 60, 0, 6, 12, 60,102,102,102
380 DATA 60, 0, 12, 24,102,102,102,102, 60, 0
390 DATA 27, 20, 60,102,102,102, 60, 0, 51, 34
400 DATA 102,102,102,102, 60, 0, 0, 0, 0, 31
410 DATA 31, 24, 24, 24, 0, 0, 0,248,248, 24
420 DATA 24, 24, 24, 24, 24, 31, 31, 0, 0, 0
430 DATA 0, 0, 0,255,255, 0, 0, 0, 0, 8
440 DATA 12,254,254, 12, 8, 0, 0, 0, 0, 0
450 DATA 0, 0, 0,255, 28, 48, 60,102,102, 60
460 DATA 12, 56,102, 0, 60, 6, 62,102, 62, 0
470 DATA 173, 17,208, 41,239,141, 17,208,173, 0
480 DATA 221, 41,252,141, 0,221,169,204,141,136
490 DATA 2,169, 55,208, 23,173, 17,208, 41,239
500 DATA 141, 17,208,173, 0,221, 9, 3,141, 0
510 DATA 221,169, 4,141,136, 2,169, 23,141, 24
520 DATA 208, 32, 68,229,173, 17,208, 3, 16,141
530 DATA 17,208, 96
550 IF S <> 37051 THEN PRINT"HIBA A DATASORBAN !":END

```


4.1.9. A képernyőgörgetés módosítása

A képernyőscroll soronkénti (tehát 8 pontsoros) léptetéssel működik. Ez az úsztatás (scroll) kissé zavaró a szemnek. Lényegesen barátságosabb az, ha lágyan, pontsoronként úszik be a képernyőbe az új sor. Az itt ismertetett rutin segítségével ez megoldható. A rutin „beépül” a KERNAL úsztató rutinjába. Ehhez azonban a KERNAL és BASIC ROM-okat le kell másolni a RAM-ba. (A BASIC-et azért, mert a KERNAL-t külön nem lehet RAM állapotra kapcsolni — egyébként más nem indokolja a lemásolást.) A megfelelő módosítás után az új rutin működésképes.

Az új scroll trükkje azon alapul, hogy a VIC lehetőséget biztosít a képernyő 7 pontsoros elmozdítására. Megfelelő időzítéssel az emberi szem becsapható, és ezt használja ki a rutin. Először 7 pontsorról feljebb mozgatja a háttérrel, majd hirtelen visszakapcsol és egy egész sort emel. Ez az utóbbi a csalás, mert csak egy pontsor feljebb mozdulását látjuk, valójában azonban lefelé 7, felfelé 8 pontsoros mozgás zajlik le néhány msec alatt. Ez olyan gyorsan történik, hogy az emberi szem nem képes követni.

A rutin a SYS 49152 utasítással hívható. A módosítás kikapcsolható a tárkonfiguráció visszaállításával: POKE 1,55.

A rutin BASIC-kezelője:

```

10 REM FOLYAMATOS SCROLL
100 POKE 53280,11:POKE 53281,0:PRINT CHR$(30)CHR$(147)
110 GOSUB 500
120 SYS 49152
130 PRINT "FOLYAMATOS GORGETES":PRINT:PRINT
140 PRINT "BEKAPCSOLAS : POKE 1,53"
150 PRINT
160 PRINT "KIKAPCSOLAS : POKE 1,55"
170 PRINT:PRINT "BILLENTYURE TOVABB !"
180 WAIT 198,1:GET X$
190 POKE 1,53
200 LIST
500 FOR I= 49152 TO 49273:READ A:POKE I,A:S=S+A:NEXT
510 DATA 169,160, 32, 7,192,169,224,133,252,169
520 DATA 32,133,253,160, 0,132,251,177,251,145
530 DATA 251,136,208,249,230,252,198,253,208,243
540 DATA 169, 41,162,192,141, 1,233,142, 2,233
550 DATA 96,224, 0,208, 58,120,169, 6,141,121
560 DATA 192,160, 2,169,248,205, 18,208,208,251
570 DATA 169,249,205, 18,208,208,251,136,208,239
580 DATA 173, 17,208, 41,240, 24,109,121,192,141
590 DATA 17,208,206,121,192, 16,220,160, 40,204
600 DATA 18,208,208,251,169,128, 44, 17,208,208
610 DATA 242,240, 15,224, 13,208, 11,173, 17,208
620 DATA 41,240, 24,105, 7,141, 17,208, 76,240
630 DATA 233, 0
640 IF S (>) 18072 THEN PRINT"HIBA A DATASORBAN !":END
650 RETURN

```

A gépi szubrutin assembly listája:

C000			90	*=C000	
C000	A9	A0	100	SCROLL	LDA #\$A0 ; A BASIC ROM Kezdőcíme
C002	20	07	110		JSR COPY ; A másolás végrehajtása
C005	A9	E0	120		LDA #\$E0 ; A KERNAL ROM Kezdőcíme
					; Másoló ciklus
C007	85	FC	130	COPY	STA \$FC ; A kezdőcím felső tárolása
C009	A9	20	140		LDA #\$20 ; Ciklusszámláló, 32 lap
C00B	85	FD	150		STA \$FD
C00D	A0	00	160		LDY #\$00 ; Kezdőcím, alsó byte
C00F	84	FB	170		STY \$FB
C011	B1	FB	180	AA0	LDA (\$FB),Y ; A másolás végrehajtása
C013	31	FB	190		STA (\$FB),Y
C015	88		200		DEY ; A belső ciklus lefutott ?
C016	D0	F9	210		BNE AA0 ; Még nem, vissza
C018	E6	FC	220		INC \$FC ; A felső byte növelése
C01A	C6	FD	230		DEC \$FD ; A külső ciklus lefutott ?
C01C	D0	F3	240		BNE AA0 ; Még nem, vissza
C01E	A9	29	250		LDA #MOD< ; A bővítés
C020	A2	C0	260		LDX #MOD> ; Kezdőcíme
C022	8D	01	270		STA \$E901 ; A lemásolt
C025	8E	02	280		STX \$E902 ; KERNAL-ba
C028	60		290		RTS ; Az úsztató-rutin bővítése
C029	E0	00	300	MOD	CPX #\$00
C02B	D0	3A	310		BNE AA1
C02D	78		320		SEI ; A megszakítások letiltása
C02E	A9	06	330		LDA #\$06
C030	8D	79	340		STA CNT ; A számláló inicializálása
C033	A0	02	350	AA5	LDY #\$02 ; Várakozás Két félképre
C035	A9	F8	360	AA4	LDA #\$F8 ; 248
C037	CD	12	370	AA2	CMP \$D012 ; Elérte a rasztensor ?
C03A	D0	FB	380		BNE AA2 ; Még nem, várakozni
C03C	A9	F9	390		LDA #\$F9 ; 249
C03E	CD	12	400	AA3	CMP \$D012 ; Elérte a rasztensor ?
C041	D0	FB	410		BNE AA3 ; Még nem, várakozni
C043	88		420		DEY
C044	D0	EF	430		BNE AA4
C046	AD	11	440		LDA \$D011 ; VIC 1. ellenőrző vezérlő regiszter
C049	29	F0	450		AND #\$F0 ; Az alsó félbyte törlése
C04B	18		460		CLC
C04C	6D	79	470		ADC CNT ; A számláló hozzáadása
C04F	8D	11	480		STA \$D011 ; és beállítás
C052	CE	79	490		DEC CNT ; A számláló csökkentése
C055	18	DC	500		BPL AA5 ; Még pozitív, vissza
C057	A0	28	510	AA7	LDY #\$28 ; 40
C059	CC	12	520	AA6	CPY \$D012 ; A rasztensor elérte ?
C05C	D0	FB	530		BNE AA6 ; Még nem, várakozni
C05E	A9	80	540		LDA #\$80
C060	2C	11	550		BIT \$D011 ; A 7. bit magas ?
C063	D0	F2	560		BNE AA7 ; Igen, várakozni
C065	F0	0F	570		BEQ AA8 ; Nem, az eredeti rutinra
C067	E0	0D	580	AA1	CPX #\$0D
C069	D0	0B	590		BNE AA8
C06B	AD	11	600		LDA \$D011 ; VIC 1. ellenőrző-vezérlő regiszter
C06E	29	F0	610		AND #\$F0 ; Az alsó félbyte levágása
C070	18		620		CLC
C071	69	07	630		ADC #\$07 ; 7 hozzáadása
C073	8D	11	640		STA \$D011 ; és beállítás
C076	4C	F0	650	AA8	JMP \$E9F0 ; Az eredeti rutinra
C079	00		660	CNT	.BYTE \$00 ; A számláló
C07A			670		.END

ZEILEN:59 SYMBOLE:13 FEHLER:0

AA0 =C011 AA1 =C067 AA2 =C037 AA3 =C03E AA4 =C035 AA5 =C033
AA6 =C059 AA7 =C057 AA8 =C076 CNT =C079 COPY =C007 MOD =C029

4.1.10. A LIST utasítás módosítása

A programlista kinyomtatásakor zavart okoz, ha a papír perforációjára is ír a nyomtatófej. Ha azt szeretnénk, hogy a lap perforációjára ne kerüljön nyomtatott sor, módosítanunk kell a LIST-rutint.

A LIST-rutint legegyszerűbb úgy átírni, hogy a BASIC ROM-ot lemásoljuk az alatta lévő RAM-ba, néhány helyen módosítjuk, majd lekapcsolunk rá. A módosítások arra szolgálnak, hogy az eredeti LIST-rutin bizonyos helyeken kitérőt tegyen a mi rutinunkban. A bővítés számolja a kinyomtatott sorokat, és egy megadott számú sort elérve lapot emel, majd kezdi előlről a számlálást.

Programunk egy bizonyos szám után lapot emel, azaz kiír néhány CR karaktert, amelyek soremelésekre kényszerítik a nyomtatót.

A program hívása: SYS 49152

A program assembly listája:

0079		100	CHRGOT	=\$0079	
C000		110		*=\$C000	
C000	A2 20	120		LDX #\$20	; 32 lap
C002	A9 A0	130		LDA #\$A0	
C004	A0 00	140		LDY #\$00	
C006	84 FB	150		STY \$FB	
C008	85 FC	160		STA \$FC	; a BASIC ROM
C00A	B1 FB	170	AA0	LDA (\$FB),Y	
C00C	91 FB	180		STA (\$FB),Y	
C00E	C8	190		INY	; lemásolása a RAM-ba
C00F	D0 F9	200		BNE AA0	
C011	E6 FC	210		INC \$FC	
C013	CA	220		DEX	
C014	D0 F4	230		BNE AA0	
C016	A3 40	240		LDA #MOD-1<	
C018	8D 42 A0	250		STA \$A042	; a LIST utasítás
C01B	A9 C0	260		LDA #MOD>	
C01D	8D 43 A0	270		STA \$A043	; új címe-1
C020	A9 6C	280		LDA #COUNT<	
C022	8D D8 A6	290		STA \$A6D8	; a bővítő rutin
C025	A9 C0	300		LDA #COUNT>	
C027	8D D9 A6	310		STA \$A6D9	; beépítése az eredeti rutinba
C02A	A9 20	320		LDA #\$20	; JSR kód
C02C	8D D7 A6	330		STA \$A6D7	
C02F	A9 EA	340		LDA #\$EA	; NOP kód
C031	8D BB A6	350		STA \$A6BB	
C034	8D BC A6	360		STA \$A6BC	
C037	A9 60	370		LDA #\$60	; RTS kód
C039	8D 14 A7	380		STA \$A714	; melegstart helyett RTS
C03C	A9 36	390		LDA #\$36	; lekapcsolás
C03E	85 01	400		STA \$01	; a RAM-ra
C040	60	410		RTS	
C041	A9 00	420	MOD	LDA #\$00	
C043	8D DB 03	430		STA \$03DB	; a sor számlálók
C046	8D DC 03	440		STA \$03DC	; inicializálása
C049	A5 7A	450		LDA \$7A	
C04B	48	460		PHA	; a programszámláló
C04C	A5 7B	470		LDA \$7B	
C04E	48	480		PHA	; verembe mentése
C04F	20 79 00	490		JSR CHRGOT	; az utolsó karakter újraolvasása
C052	20 9C A6	500		JSR \$A69C	; az eredeti LIST-rutinra
C055	A9 00	510		LDA #\$00	
C057	8D DB 03	520		STA \$03DB	; a számlálók


```

C05A 8D DC 03      530      STA $03DC      ; inicializálása
C05D 20 D7 AA      540      JSR $AAD7      ; CR kiírása
C060 68            550      PLA
C061 85 7B          560      STA $7B        ; a programszámláló
C063 68            570      PLA
C064 85 7A          580      STA $7A        ; visszatöltése
C066 20 F8 A8      590      JSR $A8F8      ; a következő utasítás megkeresése
C069 4C 73 00      600      JMP CHRGOT     ; a rutin vége

C06C 8C A1 C0      610 COUNT STY CNT4      ; a mutató ideiglenes tárolása
C06F AE 9D C0      620      LDX CNT0      ; a határérték
C072 EC A0 C0      630      CPX CNT3      ; egyenlő a számlálóval ?
C075 F0 06          640      BEQ FEED      ; igen, lapot emelni
C077 EE A0 C0      650      INC CNT3      ; még nem, növelni a számlálót
C07A 4C 96 C0      660      JMP AA2       ; folytatni a listázást

C07D AE 9E C0      670 FEED  LDX CNT1      ; az üres sorok száma
C080 EC 9F C0      680      CPX CNT2      ; egyenlő a számlálóval ?
C083 F0 03          690      BEQ AA1       ; igen, új lapot kezdeni
C085 20 D7 AA      700      JSR $AAD7      ; CR kiírása
C088 EE 9F C0      710      INC CNT2      ; a számláló növelése
C08B 4C 7D C0      720      JMP FEED      ; és vissza

C08E A9 00          730 AA1   LDA #$00      ; a számlálók
C090 8D A0 C0      740      STA CNT3      ; inicializálása
C093 8D 9F C0      750      STA CNT2
C096 AC A1 C0      760 AA2   LDY CNT4      ; az offset visszatöltése
C099 C8            770      INY
C09A B1 5F          780      LDA ($5F),Y   ; a következő karakter
C09C 60            790      RTS

C09D 3E            800 CNT0  .BYTE 62      ; a sorok száma egy lapon
C09E 0A            810 CNT1  .BYTE 10      ; a soremelések száma
C09F 00            820 CNT2  .BYTE 0       ; a soremelések számlálása
C0A0 00            830 CNT3  .BYTE 0       ; a lapra írt sorok száma
C0A1 00            840 CNT4  .BYTE 0       ; átmeneti tároló
C0A2              850      .END

```

ZEILEN:76 SYMBOLE:12 FEHLER:0

AA0 =C00A AA1 =C08E AA2 =C096 CHRGOT=C079 CNT0 =C09D CNT1 =C09E
CNT2 =C09F CNT3 =C0A0 CNT4 =C0A1 COUNT =C06C FEED =C07D MOD =C041

A BASIC-betöltő:

```

100 FOR I= 49152 TO 49313:READ A:POKE I,A:S=S+A:NEXT
110 IF S <> 22978 THEN PRINT"HIBA A DATASORBAN !":END
120 SYS49152
130 :
140 DATA 162, 32,169,160,160, 0,132,251,133,252
150 DATA 177,251,145,251,200,208,249,230,252,202
160 DATA 208,244,169, 64,141, 66,160,169,192,141
170 DATA 67,160,169,108,141,216,166,169,192,141
180 DATA 217,166,169, 32,141,215,166,169,234,141
190 DATA 187,166,141,188,166,169, 96,141, 20,167
200 DATA 169, 54,133, 1, 96,169, 0,141,219, 3
210 DATA 141,220, 3,165,122, 72,165,123, 72, 32
220 DATA 121, 0, 32,156,166,169, 0,141,219, 3
230 DATA 141,220, 3, 32,215,170,104,133,123,104
240 DATA 133,122, 32,248,168, 76,121, 0,140,161
250 DATA 192,174,157,192,236,160,192,240, 6,238
260 DATA 160,192, 76,150,192,174,158,192,236,159
270 DATA 192,240, 9, 32,215,170,238,159,192, 76
280 DATA 125,192,169, 0,141,160,192,141,159,192
290 DATA 172,161,192,200,177, 95, 36
300 DATA 62 : REM A KIIRT SOROK SZAMA
310 DATA 10 : REM AZ EMELT SOROK SZAMA
320 DATA 0, 0, 0

```

4.1.11. A változók kiírása

A legtöbb BASIC-bővítés tartalmazza. Programfejlesztés során nagyon hasznos, mert együtt látható a legtöbb változó, amely a változóterületen van. A változók értékeiből következtetni lehet a program működésére, esetleges hibájára.

A rutin kiírja a változó nevét és típusát, majd tartalmát. A változók a deklaráció sorrendjében jelennek meg. A rutin a SYS 49152 utasítással hívható. Assembly listája:

C000	100	*=\$C000	
E386	110	WARM	=\$E386
FFD2	120	PRINT	=\$FFD2
AAD7	130	CROUT	=\$AAD7
A82C	140	STOP	=\$A82C
B395	150	FLOAT	=\$B395
BBA6	160	YFAC	=\$BBA6
BDD7	170	ASCII	=\$BDD7
C000 A5 2D	180	DUMP	LDA \$2D ; A programvégműtató
C002 A4 2E	190		LDY \$2E ; beolvasása
C004 85 45	200	AA6	STA \$45 ; és tárolása
C006 84 46	210		STY \$46
C008 C4 30	220		CPY \$30 ; összehasonlítás a változók vége
C00A D0 02	230		BNE AA0
C00C C5 2F	240		CMP \$2F ; mutatóval
C00E 90 03	250	AA0	BCC AA1 ; Ha kisebb, ugrás a rutinra
C010 4C 86 E3	260		JMP WARM ; Ha nagyobb, a BASIC melegstartra
C013 69 02	270	AA1	ADC #\$02 ; Kettő hozzáadása az alsó byte-hoz
C015 90 01	280		BCC AA2 ; Ha nincs túlcsordulás, ugrás
C017 C8	290		INY ; A felső byte növelése
C018 85 22	300	AA2	STA \$22 ; A változókezdete
C01A 84 23	310		STY \$23 ; rögzítése
C01C 20 49 C0	320		JSR NAME ; A változó nevének kiírása
C01F 8A	330		TXA ; A név első byte-jának vizsgálata
C020 10 0C	340		BPL AA3 ; Valós vagy szöveg típusú változó
C022 98	350		TYA ; A név második byte-jának vizsgálata
C023 10 18	360		BPL AA4 ; Ha nem változó, címnövelés 7-tel
C025 20 7F C0	370		JSR SPACE ; A szóköz és az egyenlőségjel kiírása
C028 20 88 C0	380		JSR INTEG ; Egész típusú változó kiírása
C02B 4C 3D C0	390		JMP AA4 ; A cím növelése 7-tel
C02E 20 7F C0	400	AA3	JSR SPACE ; Szóköz és egyenlőségjel kiírása
C031 98	410		TYA
C032 30 06	420		BMI AA5 ; Ha szövegváltozó, ugrás
C034 20 9A C0	430		JSR REAL ; Valós típusú változó kiírása
C037 4C 3D C0	440		JMP AA4 ; A cím növelése 7-tel
C03A 20 A0 C0	450	AA5	JSR STRING ; Szövegváltozó kiírása
C03D A5 45	460	AA4	LDA \$45 ; A jelenlegi cím
C03F A4 46	470		LDY \$46 ; beolvasása
C041 18	480		CLC
C042 69 07	490		ADC #\$07 ; 7 hozzáadása a címhez
C044 90 BE	500		BCC AA6 ; Ha nincs túlcsordulás, vissza
C046 C8	510		INY ; A felső byte növelése
C047 B0 BB	520		BCS AA6 ; Vissza egy új változó kilistázáshoz
C049 20 D7 AA	530	NAME	JSR CROUT ; Return kiírása
C04C 20 2C AB	540		JSR STOP ; A STOP billentyű lekérdezése
C04F A0 00	550		LDY #\$00
C051 B1 45	560		LDA (\$45),Y ; A név első byte-jának
C053 AA	570		TAX ; beolvasása és tárolása
C054 10 06	580		BPL AB0 ; Ha kisebb, mint 128, ugrás
C056 C8	590		INY

C057	B1	45	600		LDA (\$45),Y ; A név második byte-ja
C059	30	01	610		BMI AB0 ; Ha nagyobb, mint 127, akkor ugrás
C05B	60		620	AB4	RTS
C05C	A0	00	630	AB0	LDY #\$00
C05E	B1	45	640		LDA (\$45),Y ; A név első byte-jának beolvasása
C060	29	7F	650		AND #\$7F ; és az offset levágása
C062	20	D2	660		JSR PRINT ; Az első byte kiírása
C065	C8		670		INY
C066	B1	45	680		LDA (\$45),Y ; A második byte beolvasása
C068	A8		690		TAY ; Tárolás az Y regiszterben
C069	29	7F	700		AND #\$7F ; Az offset levágása
C06B	F0	03	710		BEQ AB1 ; Ha nulla, a kiírást átugrani
C06D	20	D2	720		JSR PRINT ; A második byte kiírása
C070	8A		730	AB1	TXA ; Az első byte vizsgálata
C071	10	04	740		BPL AB2 ; Kisebb, mint 128, valós vagy szöveg
C073	A9	25	750		LDA #\$25 ; "%"
C075	D0	05	760		BNE AB3 ; Feltétel nélküli ugrás
C077	9C		770	AB2	TYA ; A második byte vizsgálata
C078	10	E1	780		BPL AB4 ; Kisebb, mint 128, valós
C07A	A9	24	790		LDA #\$24 ; "\$"
C07C	4C	D2	800	AB3	JMP PRINT ; A % vagy a \$ kiírása
					; Szóköz és egyenlőségjel kiírása
C07F	A9	20	810	SPACE	LDA #\$20 ; Szóköz
C081	20	D2	820		JSR PRINT ; Kiírása
C084	A3	3D	830		LDA #\$3D ; Egyenlőségjel
C086	D0	F4	840		BNE AB3 ; Kiírása
					; Egész típusú változó kiírása
C088	A0	00	850	INTEG	LDY #\$00
C08A	B1	22	860		LDA (\$22),Y ; Az alsó byte beolvasása
C08C	AA		870		TAX
C08D	C8		880		INY
C08E	B1	22	890		LDA (\$22),Y ; A felső byte beolvasása
C090	A8		900		TAY
C091	8A		910		TXA
C092	20	95	920		JSR FLOAT ; Átalakítás lebegőpontosá
C095	A0	01	930	AB5	LDY #\$01
C097	4C	D7	940		JMP ASCII ; FAC/ASCII átalakítás és kiírás
					; Valós típusú változó kiírása
C09A	20	A6	950	REAL	JSR YFAC ; Egy változó átvitele a FAC-ba
C09D	4C	95	960		JMP AB5 ; Átalakítás és kiírás
					; Szöveges változó kiírása
C0A0	20	BF	970	STRING	JSR AB6 ; Idézőjel kiírása
C0A3	A0	02	980		LDY #\$02
C0A5	B1	22	990		LDA (\$22),Y ; A cím felső byte beolvasása
C0A7	85	25	1000		STA \$25 ; és tárolása
C0A9	88		1010		DEY
C0AA	B1	22	1020		LDA (\$22),Y ; A cím alsó beolvasása
C0AC	85	24	1030		STA \$24 ; és tárolása
C0AE	88		1040		DEY
C0AF	B1	22	1050		LDA (\$22),Y ; A hossz beolvasása
C0B1	85	26	1060		STA \$26 ; és tárolása
C0B3	F0	0A	1070		BEQ AB6 ; A hossz nulla, ugrás
C0B5	B1	24	1080	AB7	LDA (\$24),Y ; A szöveg karaktereinek beolvasása
C0B7	20	D2	1090		JSR PRINT ; és kiírása
C0BA	C8		1100		INY
C0BB	C4	26	1110		CPY \$26 ; Elérte a hosszt ?
C0BD	D0	F6	1120		BNE AB7 ; Még nem
C0BF	A9	22	1130	AB6	LDA #\$22 ; Idézőjel
C0C1	4C	D2	1140		JMP PRINT ; Kiírása
C0C4			1150		.END

ZEILEN: 106 SYMBOLE: 28 FEHLER: 0

AA0	=C00E	AA1	=C013	AA2	=C018	AA3	=C02E	AA4	=C03D	AA5	=C03A
AA6	=C004	AB0	=C05C	AB1	=C070	AB2	=C077	AB3	=C07C	AB4	=C05B
AB5	=C095	AB6	=C0BF	AB7	=C0B5	ASCII	=BDD7	CROUT	=AAD7	DUMP	=C000
FLOAT	=B335	INTEG	=C088	NAME	=C043	PRINT	=FFD2	REAL	=C09A	SPACE	=C07F
STOP	=A82C	STRING	=C0A0	WARM	=E386	YFAC	=BBA6				

A BASIC-betöltő:

```

100 FOR I= 49152 TO 49347:READ A:POKE I,A:S=S+A:NEXT
110 DATA 165, 45,164, 46,133, 69,132, 70,196, 48
120 DATA 208, 2,197, 47,144, 3, 76,134,227,105
130 DATA 2,144, 1,200,133, 34,132, 35, 32, 73
140 DATA 192,136, 16, 12,152, 16, 24, 32,127,192
150 DATA 32,136,192, 76, 61,192, 32,127,192,152
160 DATA 48, 6, 32,154,192, 76, 61,192, 32,160
170 DATA 192,165, 69,164, 70, 24,105, 7,144,190
180 DATA 200,176,187, 32,215,170, 32, 44,168,160
190 DATA 0,177, 69,170, 16, 6,200,177, 69, 48
200 DATA 1, 96,160, 0,177, 69, 41,127, 32,210
210 DATA 255,200,177, 69,168, 41,127,240, 3, 32
220 DATA 210,255,138, 16, 4,169, 37,208, 5,152
230 DATA 16,225,169, 36, 76,210,255,169, 32, 32
240 DATA 210,255,169, 61,208,244,160, 0,177, 34
250 DATA 170,200,177, 34,168,138, 32,149,179,160
260 DATA 1, 76,215,189, 32,166,187, 76,149,192
270 DATA 32,191,192,160, 2,177, 34,133, 37,136
280 DATA 177, 34,133, 36,136,177, 34,133, 38,240
290 DATA 10,177, 36, 32,210,255,200,196, 38,208
300 DATA 246,169, 34, 76,210,255
320 IF S <> 22920 THEN PRINT"HIBA A DATASORBAN !":END

```

4.1.12. A BASIC program visszaállítása

A NEW-val törölt BASIC-program nem törlődik azonnal fizikailag is a tárban. A program visszaállítható egészen addig, amíg valamilyen tárműveletet nem végeztünk, pl. változódekларációt. Az ismertetett gépi program ezt végzi el:

- megkeresi a BASIC sorok végét,
- a sorvégmutatót követő címet elhelyezi a sor láncolási byte-jaiban,
- veszi a következő sort,
- figyeli a program végét jelentő három \$00 byte-ot,
- beállítja a programmutatókat a nulláslapon.

A program a SYS 49152 utasítással hívható. Assembly listája:

C000	100	*=\$C000	
C000 A5 2B	110 RENEW	LDA \$2B	; BASIC-kezdet, alsó byte
C002 85 FB	120	STA \$FB	
C004 A5 2C	130	LDA \$2C	; BASIC kezdet, felső byte
C006 85 FC	140	STA \$FC	
C008 A0 03	150 AA1	LDY #\$03	; Offset a sor első byte-jára
C00A C8	160 AA0	INY	
C00B B1 FB	170	LDA (\$FB),Y	; A sor vég relatív címének
C00D D0 FB	180	BNE AA0	; Kiszámítása
C00F C8	190	INY	
C010 98	200	TYA	
C011 18	210	CLC	
C012 65 FB	220	ADC \$FB	; A sor végcímének Kiszámítása
C014 AA	230	TAX	; Alsó byte
C015 A0 00	240	LDY #\$00	
C017 98	250	TYA	
C018 65 FC	260	ADC \$FC	
C01A 48	270	PHA	; Felső byte a verembe

C01B 8A	280	TXA
C01C 91 FB	290	STA (\$FB),Y ; A láncolási mutató
C01E C8	300	INY
C01F 68	310	PLA
C020 91 FB	320	STA (\$FB),Y ; tárolása
C022 85 FC	330	STA \$FC ; A munkaregiszter
C024 86 FB	340	STX \$FB ; új értéke
C026 B1 FB	350	LDA (\$FB),Y ; Program vége ?
C028 D0 DE	360	BNE AA1 ; Nem, vissza
C02A 88	370	DEY
C02B B1 FB	380	LDA (\$FB),Y ; Program vége ?
C02D D0 D9	390	BNE AA1 ; Nem, vissza
C02F A5 FB	400	LDA \$FB
C031 A6 FC	410	LDX \$FC ; A programvég mutató
C033 18	420	CLC
C034 69 02	430	ADC #\$02 ; Kiszámítása
C036 90 01	440	BCC AA2
C038 E8	450	INX
C039 85 2D	460	STA \$2D ; és betöltése
C03B 86 2E	470	STX \$2E ; a regiszterbe
C03D 20 60 A6	480	JSR \$A660 ; CLR utasítás
C040 4C 86 E3	490	JMP \$E386 ; BASIC melegstart
C043	500	.END

ZEILEN:41 SYMBOLE:4 FEHLER:0

AA0 =C00A AA1 =C00B AA2 =C039 RENEW =C000

A BASIC-betöltő:

```

100 FOR I= 49152 TO 49218 :READ X:POKE I,X:S=S+X:NEXT
110 DATA 165, 43,133,251,165, 44,133,252,160, 3,200,177
120 DATA 251,208,251,200,152, 24,101,251,170,160, 0,152
130 DATA 101,252, 72,136,145,251,200,104,145,251,133,252
140 DATA 134,251,177,251,208,222,136,177,251,208,217,165
150 DATA 251,166,252, 24,105, 2,144, 1,232,133, 45,134
160 DATA 46, 32, 96,166, 76,134,227
170 IF S <> 10353 THEN PRINT "HIBA A DATASORBAN !":END
180 PRINT "OK !!":END

```

4.1.13. A képernyő tartalmának kiírása — HARDCOPY

A HARDCOPY nem túl gyakran használt rutin. Viszonylag kevés felhasználói program alkalmazza; inkább érdekességként említhető. Példaként szolgálhat a nyomtató megnyitására, képernyőkód — ASCII átalakításra, a STOP billentyű figyelésére stb. A program csak a karakteres képernyőt képes kinyomtatni, a nagyfelbontású képernyő kinyomtatásának programja erősen nyomtatófüggő. Az ismertetett program minden C64-eshez illeszthető nyomtatóval működik.

Indítása: SYS 49152. A program futás közben a STOP billentyű lenyomásával megállítható. Assembly listája:

C000	100	*=\$C000
FFC0	110	OPEN =\$FFC0
FFC3	120	CLOSE =\$FFC3
FFC9	130	CHKOUT =\$FFC9
FFCC	140	CLRCHN =\$FFCC

ZEILEN:53 SYMBOLE:13 FEHLER:0

AA0	=C01E	AA1	=C02A	AA2	=C038	AA3	=C03C	AA4	=C04E	AA5	=C056
CHKOUT	=FFC9	CLOSE	=FFC3	CLRCHN	=FFCC	HCOPY	=C000	OPEN	=FFC0	PRINT	=FFD2
STOP	=FFE1										

A BASIC-betöltő:

```

100 FOR I= 49152 TO 49245:READ A:POKE I,A:S=S+A:NEXT
110 DATA 169, 4,133,186,169,126,133,184,169, 0
120 DATA 160, 4,133,113,132,114,133,183,133,185
130 DATA 32,192,255,166,184, 32,201,255,162, 25
140 DATA 169, 13, 32,210,255, 32,225,255,240, 46
150 DATA 160, 0,177,113,133,103, 41, 63, 6,103
160 DATA 36,103, 16, 2, 9,128,112, 2, 9, 64
170 DATA 32,210,255,200,132, 40,208,230,152, 24
180 DATA 101,113,133,113,144, 2,230,114,202,208
190 DATA 205,169, 13, 32,210,255, 32,204,255,169
200 DATA 126, 76,195,255
220 IF S <> 12023 THEN PRINT"HIBA A DATASORBAN !":END

```

4.1.14. Önálló utasítások előállítása — hardcopy parancsmódban

Gépi nyelvű szubrutinjaink indítására a SYS utasítást is alkalmazhatjuk, de a megfelelő rutin indítását ez körülményessé teszi. Sokkal könnyebb egy kulcsszót megjegyezni, mint egy számértéket, amelyet esetleg még paraméterek is követnek. Ezt a nehézséget leküzdhetjük, mert a BASIC interpreter fontosabb rutinjai egy változtatható ugrótáblán keresztül érhetők el. Az ugrótábla megváltoztatása lehetővé teszi, hogy az interpreter felismerjen általunk előállított utasításkulcsszót is.

Ahhoz, hogy parancsmódban használható utasítást hozzunk létre, az ugrótábla \$0302 — 0303 címén található vektort kell módosítani. A módosítás elve egyszerű: az eredeti ROM-rutin elé egy olyan rutint kell illeszteni, amely figyeli az új parancs kiadását. Az itt bemutatott példa a nagyfelbontású képernyő tartalmát jeleníti meg nyomtató segítségével. A nyomtató típusa többféle lehet, eredetileg EPSON nyomtató a megfelelő, de több más típusú nyomtató is érti az EPSON programnyelvét.

A hardcopy program pontról pontra leképezi a nagyfelbontású képernyő bittérképét. A bittérkép szervezési elve azonban más, mint amit a nyomtató követel meg. A mátrixnyomtató is 8x8 pontos egységeket vár, de míg a bittérkép egy byte-ja a 8x8-as mátrixban egy sort ad meg, a nyomtató számára egy oszlopot kell megadni. A programnak ezért el kell végeznie ezt az átalakítást.

A VIC leírásánál ismertettük a HIRES-tár elhelyezkedését és annak címzését. Az ott említett 16 kbyte-os lapot a CIA2 A portjának 0. és 1. bitjével címezhetjük meg. A program ennek a két bitnek a felhasználásával önállóan címzi meg a HIRES-tárat, bármelyik 16 kbyte-os lapot használjuk is. Ha a HIRES-tár a KERNAL ROM alatt van, a ROMOFF és ROMON rutinok segítségével férhetünk hozzá a tényleges információkhoz.

A bővítést a SYS 49152 utasítással kapcsolhatjuk be. Ezt követően a !C parancs kiadásával nyomtathatjuk ki a nagyfelbontású képernyőt. A gépi szubrutin akkor is lefut, ha a nyomtató nincs bekapcsolva. A program assembly listája:

DD00		100	PRA2	=\$DD00	
FFCC		110	CLRCHN	=\$FFCC	
FFBA		120	SETFLS	=\$FFBA	
FFBD		130	SETNAM	=\$FFBD	
FFC0		140	OPEN	=\$FFC0	
FFC9		150	CHKOUT	=\$FFC9	
FFD2		160	PRINT	=\$FFD2	
FFC3		170	CLOSE	=\$FFC3	
0073		180	CHRGET	=\$73	
001B		200	ESC	=27	
C000		210		*=C000	
C000	A3 0B	220		LDA #TEST<	
C002	A2 C0	230		LDX #TEST>	
C004	8D 02 03	240		STA \$0302	; Az utasításbeolvasás
C007	8E 03 03	250		STX \$0303	; vektora
C00A	60	260		RTS	
C00B	20 60 A5	270	TEST	JSR \$A560	; Egy sor beolvasása az input-pufferbe
C00E	86 7A	280		STX \$7A	; A CHRGET mutatója
C010	84 7B	290		STY \$7B	; az input-puffer kezdetére
C012	20 73 00	300		JSR CHRGET	; Egy karakter beolvasása
C015	AA	310		TAX	
C016	F0 F3	320		BEQ TEST	; Nincs karakter, vissza
C018	30 18	330		BCC AA8	; Számjegy, programsorként kezelni
C01A	C9 21	340		CMP #!"	; Saját utasítás ?
C01C	F0 0A	350		BEQ AA9	; Igen, ugrás
C01E	A2 FF	360		LDX #\$FF	
C020	86 3A	370		STX \$3A	; A parancsmód beállítása
C022	20 79 A5	380		JSR \$A579	; Az utasítássor kódolása
C025	4C E1 A7	390		JMP \$A7E1	; Az utasítássor végrehajtása
C028	20 73 00	400	AA9	JSR CHRGET	; Egy karakter beolvasása
C02B	C9 43	410		CMP #"C"	; Hardcopy ?
C02D	F0 0A	420		BEQ HCOPY	; Igen, ugrás
C02F	4C 08 AF	430		JMP \$AF08	; Nem, SYNTAX ERROR
C032	A2 FF	440	AA8	LDX #\$FF	
C034	86 3A	450		STX \$3A	; A parancsmód beállítása
C036	4C 3C A4	460		JMP \$A43C	; Programsorok beiktatása és törlése
C039	AD 00 DD	470	HCOPY	LDA PRA2	; A CIA2 A portja, a VA14-15 bitek
C03C	49 03	480		EOR #\$03	; %00000011, a 0-1. bitek invertálása
C03E	29 03	490		AND #\$03	; %00000011, a felső bitek törlése
C040	38	500		SEC	; Az eredmény 4. bitje magas
C041	6A	510		ROR A	; A bitek áttolása
C042	6A	520		ROR A	
C043	6A	530		ROR A	; a felső félbyte-ba
C044	85 FE	540		STA \$FE	; A felső címbyte tárolása
C046	A9 00	550		LDA #\$00	; Az alsó címbyte nulla
C048	85 FD	560		STA \$FD	
C04A	20 CC FF	570		JSR CLRCHN	; Az aktív I/O csatorna törlése
C04D	A9 04	580		LDA #\$04	; Logikai file-szám
C04F	AA	590		TAX	; Egységszám
C050	A0 FF	600		LDY #\$FF	; Másodlagos cím
C052	20 BA FF	610		JSR SETFLS	; A paraméterek beállítása
C055	A9 00	620		LDA #\$00	; Nincs név
C057	20 BD FF	630		JSR SETNAM	; A név paramétereinek beállítása
C05A	20 C0 FF	640		JSR OPEN	; A logikai file megnyitása
C05D	A2 04	650		LDX #\$04	; A 4-es logikai file
C05F	20 C9 FF	660		JSR CHKOUT	; Kijelölése kimenetre
C062	A9 1B	670		LDA #ESC	; Epson Standard Code
C064	20 D2 FF	680		JSR PRINT	; Kiírása
C067	A9 41	690		LDA #"A"	; A soremelés beállítása
C069	20 D2 FF	700		JSR PRINT	
C06C	A9 08	710		LDA #\$08	; 8/72 inch
C06E	20 D2 FF	720		JSR PRINT	
C071	A0 00	730		LDY #\$00	
C073	A9 18	740		LDA #25	; 25 képernyősor
C075	85 55	750		STA \$55	

C079	A9	20		770		LDA #\$20	
C07B	20	D2	FF	780	AA0	JSR PRINT	; Kiírása
C07E	CA			790		DEX	
C07F	D0	FA		800		BNE AA0	
C081	A9	1B		810		LDA #ESC	; Epson Standard Code
C083	20	D2	FF	820		JSR PRINT	
C086	A9	4B		830		LDA #"K"	; A grafikus mód beállítása
C088	20	D2	FF	840		JSR PRINT	
C08B	A9	40		850		LDA #\$40	
C08D	20	D2	FF	860		JSR PRINT	
C090	A9	01		870		LDA #\$01	
C092	20	D2	FF	880		JSR PRINT	
C095	A9	28		890		LDA #40	; 40 Karakter
C097	85	56		900		STA \$56	
C099	A9	08		910	AA5	LDA #\$08	; 8 pontsor
C09B	85	57		920		STA \$57	
C09D	20	E7	C0	930		JSR ROMOFF	; A ROM kikapcsolása
C0A0	B1	FD		940	AA3	LDA (\$FD),Y	; Egy byte a HIRES-tárból
C0A2	C8			950		INY	
C0A3	D0	02		960		BNE AA1	
C0A5	E6	FE		970		INC \$FE	
C0A7	A2	08		980	AA1	LDX #\$08	; 8 pontsor
C0A9	2A			990	AA2	ROL A	
C0AA	36	4A		1000		ROL \$4A,X	; A pontsorok átalakítása
C0AC	CA			1010		DEX	; oszlopokká
C0AD	D0	FA		1020		BNE AA2	
C0AF	C6	57		1030		DEC \$57	
C0B1	D0	ED		1040		BNE AA3	
C0B3	20	EF	C0	1050		JSR ROMON	; A ROM visszakapcsolása
C0B6	A2	08		1060		LDX #\$08	; 8 pontsor
C0B8	B5	4A		1070	AA4	LDA \$4A,X	; Az oszlopok kiírása
C0BA	20	D2	FF	1080		JSR PRINT	
C0BD	CA			1090		DEX	
C0BE	D0	F8		1100		BNE AA4	
C0C0	C6	56		1110		DEC \$56	; Utolsó karakter a sorban ?
C0C2	D0	D5		1120		BNE AA5	; Még nem, vissza
C0C4	A9	0D		1130		LDA #\$0D	; A sort lezáró RETURN
C0C6	20	D2	FF	1140		JSR PRINT	; Kiírása
C0C9	C6	55		1150		DEC \$55	; A sorszámoló csökkentése
C0CB	D0	AA		1160		BNE AA6	; Még van sor, vissza
C0CD	A9	1B		1170		LDA #ESC	; Epson Standard Code
C0CF	20	D2	FF	1180		JSR PRINT	
C0D2	A9	41		1190		LDA #"A"	; A soremelés
C0D4	20	D2	FF	1200		JSR PRINT	
C0D7	A9	0C		1210		LDA #\$0C	; eredeti értékek helyreállítása
C0D9	20	D2	FF	1220		JSR PRINT	
C0DC	A9	04		1230		LDA #\$04	; A 4-es logikai file
C0DE	20	C3	FF	1240		JSR CLOSE	; lezárva
C0E1	20	CC	FF	1250		JSR CLRCHN	; Az aktív csatorna törlése
C0E4	4C	86	E3	1260		JMP \$E386	; Melegstart
							; A ROM kikapcsolása
C0E7	78			1270	ROMOFF	SEI	; A megszakítások tiltása
C0E8	A5	01		1280		LDA \$01	; Processzorport
C0EA	29	FD		1290		AND #\$FD	; %11111101
C0EC	85	01		1300		STA \$01	
C0EE	60			1310		RTS	
							; A ROM visszakapcsolása
C0EF	A5	01		1320	ROMON	LDA \$01	
C0F1	09	02		1330		ORA #\$02	; %00000010
C0F3	85	01		1340		STA \$01	
C0F5	58			1350		CLI	; A megszakítások engedélyezése
C0F6	60			1360		RTS	
C0F7				1370		.END	

ZEILEN:128 SYMBOLE:23 FEHLER:0

AA0	=C07B	AA1	=C0A7	AA2	=C0A9	AA3	=C0A0	AA4	=C0B8	AA5	=C099
AA6	=C077	AA8	=C032	AA9	=C028	CHKOUT	=FFC9	CHRGET	=0073	CLOSE	=FFC3
CLRCHN	=FFCC	ESC	=001B	HCOPY	=C039	OPEN	=FFC0	PRA2	=DD00	PRINT	=FFD2

BASIC-betöltője:

```

100 FOR I= 43152 TO 49398 :READ X:POKE I,X:S=S+X:NEXT
110 DATA 169, 11,162,132,141, 2, 3,142, 3, 3, 96, 32
120 DATA 96,165,134,122,132,123, 32,115, 0,170,240,243
130 DATA 144, 24,201, 33,240, 10,162,255,134, 58, 32,121
140 DATA 165, 76,225,167, 32,115, 0,201, 67,240, 10, 76
150 DATA 8,175,162,255,134, 58, 76,156,164,173, 0,221
160 DATA 73, 3, 41, 3, 56,106,106,106,133,254,169, 0
170 DATA 133,253, 32,204,255,169, 4,170,160,255, 32,186
180 DATA 255,169, 0, 32,189,255, 32,192,255,162, 4, 32
190 DATA 201,255,169, 27, 32,210,255,169, 65, 32,210,255
200 DATA 169, 8, 32,210,255,160, 0,169, 25,133, 85,162
210 DATA 13,169, 32, 32,210,255,202,208,250,169, 27, 32
220 DATA 210,255,169, 75, 32,210,255,169, 64, 32,210,255
230 DATA 169, 1, 32,210,255,169, 40,133, 86,169, 8,133
240 DATA 87, 32,231,192,177,253,200,208, 2,230,254,162
250 DATA 8, 42, 54, 74,202,208,250,198, 87,208,237, 32
260 DATA 239,192,162, 8,181, 74, 32,210,255,202,208,248
270 DATA 198, 86,208,213,169, 13, 32,210,255,198, 85,208
280 DATA 170,169, 27, 32,210,255,169, 65, 32,210,255,169
290 DATA 12, 32,210,255,169, 4, 32,195,255, 32,204,255
300 DATA 76,134,227,120,165, 1, 41,253,133, 1, 96,165
310 DATA 1, 9, 2,133, 1, 88, 96
320 IF S <> 32280 THEN PRINT"HIBA A DATASORBAN !":END
330 PRINT"OK !!":END

```

4.1.15. Tömbváltozók szelektív törlése — programba építhető BASIC-bővítés

A BASIC interpreter bővítésére négy lehetőség kínálkozik. Az első és legegyszerűbb módszer az, hogy a megfelelő bővítőrutint a SYS utasítással érhetjük el. A második módszer az, ha az interpretert csak parancsmódban kiadható utasításokkal bővítjük. Ilyen bővítéseket tartalmaz pl. a HELP PLUS. Az ilyen utasítások minden esetben valamilyen megkülönböztetett karakterrel kezdődnek: #, ! stb. A megkülönböztetett karakterekkel azonosítható utasításokat azonban nemcsak parancsmódban használhatjuk, elképzelhető olyan megoldás is, amelyben ezek az utasítások program- és parancsmódban egyaránt használhatók. Az itt bemutatott példaprogram egy ilyen megoldást szemléltet. A negyedik megoldás jóval bonyolultabb az előzőeknél, de megvan az az előnye, hogy minden tekintetben illeszkedik a BASIC-hez. Ez a megoldás tulajdonképpen utánozza a BASIC működési elvét. Az utasításnak van saját tokenje, és szöveges megfelelője is (mint \$80 — END).

A példaprogram lehetővé teszi azt, hogy a BASIC-beli CLR parancs használata, tehát a teljes változókészlet törlése nélkül szüntethessünk meg olyan változótömböket, amelyekre már nincsen szükség. A megfelelő tömböket az utasítás paraméterezésében jelölhetjük ki. Az utasítás használatát megkönnyíti az elérési módja. A körülményes SYS utasítás helyett egy egyszerű elnevezést használhatunk; ez az elnevezés a ! és az E betű egymáshoz rendeléséből adódik. A ! hatására az interpreter egy külön rutinban vizsgálja meg, hogy a számára kiadott

utasítás végrehajtható-e a bővítésben. Ha az utasítás szintaxisa helyes, a megadott tömb törlődik a memóriából. Ha a megadott változóhoz nem tartozott létező tömb, a rutin veszi a következő változót, vagy hibajelzés nélkül véget ér.

A bővítést egy kis inicializáló rutin kapcsolja be, amely a SYS 49152 utasítással hívható. A törlést ezt követően a !E parancs kiadása kezdeményezi. A !E után változólista következhet, amely tetszőleges tagból állhat. A lista elemeit vesszőkkel választhatjuk el egymástól. A lista elemeinek követnie kell a BASIC-változók szintaxisát, viszont nem kell megadni az indexes változóknál megszokott zárójeleket. A rutin tehát valós változótömböt pl. A, B, C név megadása után törölhet, de alkalmas egész és szöveges típusú változótömb törlésére is (A%, C\$ stb.).

A rutin működési elve egyszerű: a törölni kívánt tömbre másolja az utána következőt. Ha az utolsónak deklarált tömbről van szó, a tömbterület végmutatóit állítja át csupán. Mivel a szöveges változók valódi tartalma más területen helyezkedik el, ezért szöveges típusú tömb törlése esetén szükséges meghívni az interpreter „szemétkezelő” (Garbage Collection — \$B526) rutinját is. A törlőrutin mindhárom esetben (valós, egész és szöveges típus) meghívja, de ez nem okoz problémát akkor sem, ha elméletileg nincs is mindig szükség rá. A bővítőrutin assembly listája:

C000		100	*= \$C000	
AEFD		110	CHKCOM = \$AEFD	
0073		120	CHRGET = \$73	
0079		130	CHRGOT = \$79	
AF08		140	SYNTAX = \$AF08	
C000 A3 0B		150	INIT	LDA #NEXEC<
C002 A2 C0		160		LDX #NEXEC>
C004 8D 08 03		170		STA \$0308 ; Az utasításvégrehajtás
C007 8E 03 03		180		STX \$0309 ; vektora
C00A 60		190		RTS
C00B 20 73 00	200	NEXEC	JSR CHRGET	; A következő Karakter beolvasása
C00E C9 21	210		CMP # "!"	; Saját utasítás ?
C010 D0 0A	220		BNE AC0	; Nem, ugrás
C012 20 73 00	230		JSR CHRGET	; A következő Karakter beolvasása
C015 C9 45	240		CMP # "E"	; Erase ?
C017 F0 09	250		BEQ ERASE	; Igen, ugrás
C019 4C 08 AF	260		JMP SYNTAX	; Nem, SYNTAX ERROR
C01C 20 79 00	270	AC0	JSR CHRGOT	; Az utolsó Karakter újraolvasása
C01F 4C E7 A7	280		JMP \$A7E7	; Utasításvégrehajtás
C022 20 73 00	290	ERASE	JSR CHRGET	; A következő Karakter beolvasása
C025 85 45	300	AB9	STA \$45	; és tárolása
C027 20 13 B1	310		JSR \$B113	; Alfabetikus ellenőrzés
C02A 60 03	320		BCS AB0	; Nem számjegy, ugrás
C02C 4C 08 AF	330		JMP SYNTAX	; Számjegy, SYNTAX ERROR
C02F A2 00	340	AB0	LDX #\$00	; A típus numerikus
C031 86 0D	350		STX \$0D	; Típusjelző
C033 20 73 00	360		JSR CHRGET	; A következő Karakter beolvasása
C036 90 05	370		BCC AB1	; Számjegy, ugrás
C038 20 13 B1	380		JSR \$B113	; Alfabetikus ellenőrzés
C03B 90 0B	390		BCC AB2	; Nem alfabetikus jel, ugrás
C03D AA	400	AB1	TAX	; A második Karakter átmeneti tárolása
C03E 20 73 00	410	AB3	JSR CHRGET	; A következő Karakter beolvasása
C041 90 FB	420		BCC AB3	; Számjegy, ugrás
C043 20 13 B1	430		JSR \$B113	; Alfabetikus ellenőrzés

C046	B0	F6	440		BCS	AB3	; Alfabetikus jel, ugrás
C048	C9	24	450	AB2	CMP	#*\$"	; Szöveges változó ?
C04A	D0	06	460		BNE	AB4	; Nem, ugrás
C04C	A9	FF	470		LDA	#\$FF	; A típusjelző beállítása
C04E	85	0D	480		STA	\$0D	
C050	D0	0A	490		BNE	AB5	; Feltétlen ugrás
C052	C9	25	500	AB4	CMP	#"%"	; Egész típusú változó ?
C054	D0	0D	510		BNE	AB6	; Nem, ugrás
C056	A9	80	520		LDA	#\$80	; A típusok megkülönböztető jele
C058	05	45	530		ORA	\$45	; A név első karakterének
C05A	85	45	540		STA	\$45	; beállítása
C05C	8A		550	AB5	TXA		; A második karakter
C05D	09	80	560		ORA	#\$80	; beállítása
C05F	AA		570		TAX		
C060	20	73 00	580		JSR	CHRGET	; A következő karakter beolvasása
C063	86	46	590	AB6	STX	\$46	; A név második karaktere
C065	A6	2F	600		LDX	\$2F	; A tömbváltozóterület
C067	A5	30	610		LDA	\$30	; Kezdőcíme a \$22-23-ba
C069	86	22	620	AA2	STX	\$22	
C06B	85	23	630		STA	\$23	
C06D	C5	32	640		CMP	\$32	; Az index egyenlő a
C06F	D0	04	650		BNE	AA0	
C071	E4	31	660		CPX	\$31	; tömbterület végével ?
C073	F0	62	670		BEQ	AA3	; Igen, nincs ilyen tömb, a következőt
C075	A0	00	680	AA0	LDY	#\$00	
C077	B1	22	690		LDA	(\$22),Y	; A tömb nevének első karaktere
C079	C8		700		INY		
C07A	C5	45	710		CMP	\$45	; Azonos a keresettel ?
C07C	D0	06	720		BNE	AA1	; Nem, ugrás
C07E	A5	46	730		LDA	\$46	; A második karakter
C080	D1	22	740		CMP	(\$22),Y	; is azonos ?
C082	F0	0F	750		BEQ	AA4	; Igen, ugrás
C084	C8		760	AA1	INY		; Nincs azonosság, kiszámítani
C085	B1	22	770		LDA	(\$22),Y	; a következő tömb címét
C087	18		780		CLC		
C088	65	22	790		ADC	\$22	; A jelenlegi cím hozzáadása
C08A	AA		800		TAX		; átmeneti tárolás
C08B	C8		810		INY		
C08C	B1	22	820		LDA	(\$22),Y	; A cím felső byte-jának
C08E	65	23	830		ADC	\$23	; előállítása
C090	90	D7	840		BCC	AA2	; Feltétlen ugrás
C092	60		850		RTS		
C093	C8		860	AA4	INY		; Kiszámítani a következő
C094	B1	22	870		LDA	(\$22),Y	; tömb kezdőcímét, azaz a
C096	18		880		CLC		; törlendő terület végcímét
C097	65	22	890		ADC	\$22	
C099	85	24	900		STA	\$24	; A végcím alsó byte-ja
C09B	C8		910		INY		
C09C	B1	22	920		LDA	(\$22),Y	
C09E	65	23	930		ADC	\$23	
C0A0	85	25	940		STA	\$25	; A végcím felső byte-ja
C0A2	A5	24	950		LDA	\$24	; A végcím a tömbterület felső határa
C0A4	C5	31	960		CMP	\$31	
C0A6	D0	06	970		BNE	AA9	; Nem, végrehajtani a törlést
C0A8	A6	25	980		LDX	\$25	
C0AA	E4	32	990		CPX	\$32	
C0AC	F0	1E	1000		BEQ	AB7	; Igen, csak a végmutatót módosítani
C0AE	A0	00	1010	AA9	LDY	#\$00	
C0B0	B1	24	1020	AA7	LDA	(\$24),Y	; A másolás végrehajtása
C0B2	91	22	1030		STA	(\$22),Y	
C0B4	E6	22	1040		INC	\$22	; A 'Hová' mutató
C0B6	D0	02	1050		BNE	AA5	
C0B8	E6	23	1060		INC	\$23	; növelése
C0BA	E6	24	1070	AA5	INC	\$24	; A 'Honnán' mutató
C0BC	D0	02	1080		BNE	AA6	
C0BE	E6	25	1090		INC	\$25	; növelése
C0C0	A5	24	1100	AA6	LDA	\$24	; A 'Honnán' mutató elérte a
C0C2	C5	31	1110		CMP	\$31	; tömbterület felső határát ?


```

C0C4 D0 EA      1120      BNE AA7      ; Még nem, vissza
C0C6 A5 25      1130      LDA $25
C0C8 C5 32      1140      CMP $32
C0CA 90 E4      1150      BCC AA7      ; Még nem, vissza
C0CC A5 22      1160 AB7    LDA $22      ; A végmutató
C0CE 85 31      1170      STA $31
C0D0 A5 23      1180      LDA $23      ; beállítása
C0D2 85 32      1190      STA $32
C0D4 20 26 B5    1200      JSR $B526    ; Az érvénytelen szövegek megszüntetése
C0D7 20 79 00    1210 AA3    JSR CHRGOT ; Az utolsó Karakter újraolvasása
C0DA D0 03      1220      BNE AB8      ; Van még Karakter, ugrás
C0DC 4C 1C C0    1230      JMP AC0      ; Befejezni

C0DF 20 FD AE    1240 AB8    JSR CHKCOM ; A vessző ellenőrzése
C0E2 20 79 00    1250      JSR CHRGOT ; Az utolsó Karakter újraolvasása
C0E5 4C 25 C0    1260      JMP AB9      ; Venni a Következő paramétert
C0E8              1270      .END
    
```

ZEILEN:118 SYMBOLE:27 FEHLER:0

AA0	=C075	AA1	=C084	AA2	=C069	AA3	=C0D7	AA4	=C093	AA5	=C0BA
AA6	=C0C0	AA7	=C0B0	AA9	=C0AE	AB0	=C02F	AB1	=C03D	AB2	=C048
AB3	=C03E	AB4	=C052	AB5	=C05C	AB6	=C063	AB7	=C0CC	AB8	=C0DF
AB9	=C025	AC0	=C01C	CHKCOM	=AEFD	CHRGOT	=0073	CHRGOT	=0073	ERASE	=C022
INIT	=C000	NEXEC	=C00B	SYNTAX	=AF08						

A rutin BASIC-betöltője:

```

100 FOR I= 49152 TO 49383 :READ X:POKE I,X:S=S+X:NEXT
110 DATA 163, 11,162,192,141, 8, 3,142, 9, 3, 96, 32
120 DATA 115, 0,201, 33,208, 10, 32,115, 0,201, 69,240
130 DATA 9, 76, 8,175, 32,121, 0, 76,231,167, 32,115
140 DATA 0,133, 69, 32, 19,177,176, 3, 76, 8,175,162
150 DATA 0,134, 13, 32,115, 0,144, 5, 32, 19,177,144
160 DATA 11,170, 32,115, 0,144,251, 32, 19,177,176,246
170 DATA 201, 36,208, 6,169,255,133, 13,208, 10,201, 37
180 DATA 208, 13,169,128, 5, 69,133, 69,138, 9,128,170
190 DATA 32,115, 0,134, 70,166, 47,165, 48,134, 34,133
200 DATA 35,197, 50,208, 4,228, 49,240, 98,160, 0,177
210 DATA 34,200,197, 69,208, 6,165, 70,209, 34,240, 15
220 DATA 200,177, 34, 24,101, 34,170,200,177, 34,101, 35
230 DATA 144,215, 36,200,177, 34, 24,101, 34,133, 36,200
240 DATA 177, 34,101, 35,133, 37,165, 36,197, 49,208, 6
250 DATA 166, 37,228, 50,240, 30,160, 0,177, 36,145, 34
260 DATA 230, 34,208, 2,230, 35,230, 36,208, 2,230, 37
270 DATA 165, 36,197, 49,208,234,165, 37,197, 50,144,228
280 DATA 165, 34,133, 49,165, 35,133, 50, 32, 38,161, 32
290 DATA 121, 0,208, 3, 76, 28,192, 32,253,174, 32,121
300 DATA 0, 76, 37,192
310 IF S <> 24085 THEN PRINT"HIBA A DATASORBAN !":END
320 PRINT"OK !!":END
    
```

4.2. BASIC változók kezelése gépi kódban

A BASIC és a gépi program közötti kommunikáció nem merülhet ki abban, hogy előre kijelölt tárrekeszeket közösen használnak, mert a BASIC-program ezeket körülményesen dolgozza fel. Sokkal gyorsabb megoldás, ha a gépi program eleve a BASIC számára érthető formában helyezi el az általa végzett művelet eredményét. Kézenfekvő, hogy azonos nevű BASIC-változókat használjanak. A gépi program szempontjából ez a megoldás sem hátrányos, mert az interpreter gépi programja tartalmazza mindazokat a rutinokat, amelyek a BASIC-változót kezelik.

Néhány ilyen rutin belépési pontja az interpreterben:

1. Y/A→FAC	\$B391	16 bites szám a FAC-ba
2. FAC→LB/HB	\$B7F7	A FAC címformátumra (a \$14—15-be)
3. FAC→változó	\$BBD0	A FAC kijelölt című változóba
4. FAC→ASCII	\$BDDD	FAC—ASCII átalakítás
5. ASCII→FAC	\$BCF3	ASCII—FAC átalakítás
6. FAC→ARG	\$BC0C	A FAC az ARG-ba
7. ARG→FAC	\$BBFC	Az ARG a FAC-ba
8. FAC→A#4	\$BBC7	A FAC az akku #4-be
9. FAC→A#3	\$BBCA	A FAC az akku #3-ba
10. (A/Y)→FAC	\$BBA2	Az A/Y címen levő változó a FAC-ba
11. FAC→INTEGER	\$BC9B	A FAC 16 bites egészé alakítása

Műveletek lebegőpontos számokkal:

1. FAC = INT (FAC)	\$BCCC	A FAC egész részének képzése
2. FAC = FAC + 0,5	\$B849	0,5 hozzáadása
3. FAC = FAC * 10	\$BAE2	Szorzás 10-zel
4. FAC = FAC / 10	\$BAFE	Osztás 10-zel
5. FAC = (A/Y) - FAC	\$B850	Kivonás külső változó és a FAC között
6. FAC = ARG - FAC	\$B853	Kivonás a két aritmetikai regiszter között
7. FAC = (A/Y) + FAC	\$B867	Összeadás külső változó és a FAC között
8. FAC = ARG + FAC	\$B86A	Összeadás a két aritmetikai regiszter között
9. FAC = (A/Y) * FAC	\$BA28	Szorzás külső változó és a FAC között
10. FAC = ARG * FAC	\$BA2B	Szorzás a két aritmetikai regiszter között
11. FAC = (A/Y) / FAC	\$BB0F	Osztás külső változó és a FAC között
12. FAC = ARG / FAC	\$BB12	Osztás a két aritmetikai regiszter között
13. FAC = (A/Y) ↑ FAC	\$BF78	Hatványozás külső változó és a FAC között
14. FAC = ARG ↑ FAC	\$BF7B	Hatványozás a két aritmetikai regiszter között

4.2.1. Változók definiálása

A változók definiálásakor első feladatunk az interpreter jelzőit beállítani, hogy kijelöljük a változó típusát. A típust jelző byte-ok a \$0D és \$0E címeken találhatók:

\$0D: \$00: numerikus változó
 \$FF: szöveges változó
 \$0E: \$00: valós
 \$80: egész

A \$FF — \$80 kombinációnak nincs értelme.

A típusdeklarációt követően meg kell adni a változó nevét. A név kialakításánál figyelembe kell venni a változó típusát. A név byte-jait a \$45 — 46 regiszterpárban kell tárolni. A változót a \$B0E7 rutin hozza létre, ha még nem létezik. A változó mutatóit az A/Y regiszterekben adja vissza. Ezt követően a megfelelő rutinok alkalmazásával a változó tartalma a RAM-területre tölthető.

A *valós típusú változó* létrehozása minden tekintetben követi az imént említett módszert. A rutin assembly listája:

A változó neve : AB

```
RLDEF  LDX #$00      ; típusdeklaráció
        STX $0D      ; 0=numerikus
        STX $0E      ; 0=valós
        LDA HIGH     ; a változó
        LDY LOW      ; értéke
        JSR $B391     ; átalakítás lebegőpontosá
        LDA #"A"     ; a változó
        LDY #"B"     ; neve
        STA $45      ; a változó nevének
        STY $46      ; tárolása
        JSR $B0E7     ; a változó címének meghatározása
        STA $49      ; a változó címe
        STY $4A      ; a változóterületen
        JMP $BBD0     ; a FAC átvitele a változóba
```

Az *egész típusú változó* definiálására szolgáló rutin assembler listája:

A változó neve : AB%

```
INTDEF  LDX #$00      ; típusdeklaráció
        STX $0D      ; 0=numerikus
        LDX #$80     ; 128=egész
        STX $0E      ; a változó
        LDA LOW      ; értéke
        LDY HIGH     ;
        STA $64      ; a név
        STY $65      ; tárolása
        LDA #$C1     ; a változó
        LDY #$C2     ; neve
        STA $45      ; a változó nevének
        STY $46      ; tárolása
        JSR $B0E7     ; a változó címének meghatározása
        STA $49      ; a változó címe
        STY $4A      ; a változóterületen
        JMP $A9CA     ; EGÉSZ értékadás
```


A *szöveges változó* definiálása némileg eltér az előzőektől. A szöveges változó elhelyezését megelőzően meg kell hívni a FRESTR — \$B6A3, majd a hossz előzetes megadását követően a \$B475 rutint. Ez a két rutin készíti elő a szöveges változó helyét és mutatóit. Ezt követően be kell másolnunk a változóterületre a változó mutatóit, majd a szövegterületre a változó tartalmát. A rutin assembly listája:

A változó neve : AB\$

```
STRDEF LDX #$00          ; típusdeklaráció
        STX $0E
        DEX
        STX $0D          ; 255=szöveg
        LDA #$41          ; a változó
        LDY #$C2          ; neve
        STA $45          ; a változó nevének
        STY $46          ; tárolása
        JSR $B0E7         ; a változó címének meghatározása
        STA $49          ; a változó címe
        STY $4A          ; a változóterületen
        JSR $B6A3         ; FRESTR, szövegkezelés
        LDA #HOSSZ        ; a változó hossza
        JSR $B475         ; a szövegmutató kiszámítása
        LDY #$02          ; offset
A0      LDA $0061,Y        ; a szöveg mutatói a nulláslapon
        STA ($49),Y       ; a szövegváltozó a RAM-ban
        DEY
        BPL A0
        INY
A1      LDA #$20          ; a változó tartalma (SZÖKÖZ)
        STA ($62),Y       ; a változó feltöltése
        INY
        CPY $61          ; a ciklus lefutott ?
        BNE A1            ; még nem, vissza
        RTS
```

4.2.2. A változók átalakítása

A valós változó 16 bites egészszé való átalakításakor a név megadása és a típusdeklaráció egyezik az előzőekben tárgyaltakkal. A változó megkeresése (\$B0E7) után annak tartalmát a FAC-ba kell tölteni (\$BBA2). Ezután a FAC tartalmát a \$BC9B rutin átalakítja 16 bites egészszé. Az egész szám a \$64—65 regiszterpárban található. A rutin assembly listája:

A változó neve : AB

```
REAL    LDA #"A"          ; név
        LDY #"B"
        LDX #$00
        STX $0D          ; 0=numerikus
        STX $0E          ; 0=valós
        STA $45          ; a változó nevének
        STY $46          ; tárolása
        JSR $B0E7         ; a változó címének meghatározása
        JSR $BBA2         ; a változó betöltése a FAC-ba
        JSR $BC9B         ; a FAC átalakítása címformátumra
```

```

LDA $64      ; felső byte
LDY $65      ; alsó byte
STA $FE      ; tárolás
STY $FD
RTS

```

Nagy adattömegek tárolása esetén, ha a tárolt számok értéke nem haladja meg a 2^{32} számot, helyet takaríthatunk meg azzal, hogy a számokat nem lebegőpontos vagy főleg nem ASCII formában tároljuk, hanem átalakítjuk 32 bites bináris formára. Ez azonban csak egész számokra használható. Ha a számok olyan tizedes törtek, amelyek széles spektrumban helyezkednek el, célszerűbb a lebegőpontos formát használni.

A lebegőpontos számábrázolás is bináris ábrázolás, de az exponens változó értéke miatt nemcsak egész számokra alkalmazható. A lebegőpontos szám első byte-ja az exponens, a többi négy a mantissza. A mantissza 32 bites bináris szám, amelyre az exponens hatványkitevőként kerül.

A lebegőpontos számot a következő összefüggéssel számíthatjuk át valós számmá:

$$Y = \text{mantissza} * 2^{\uparrow (\text{exponens} - 160)}$$

Ennek ismeretében a lebegőpontos egész számot, amelynek értéke kisebb, mint 2^{32} , könnyen átalakíthatjuk 32 bites bináris egészé.

A FAC-ban tárolt változó exponenséből kiszámítjuk a cikluszámláló értékét: 160-ból kivonjuk az exponenst. A ciklusmag sem bonyolult: a mantisszát minden egyes lépésben egy bittel jobbra kell mozgatni (osztani kettővel). Amint a ciklus lefutott, a mantissza helyén a 32 bites bináris szám áll.

Ha az átalakítást adattárolásra használjuk, gondoskodni kell a visszaalakításról is. A 32 bites bináris egész lebegőpontos számmá alakítása sem különösebben bonyolult feladat. Ki kell számítani a szám exponensét, és a mantisszát megfelelő alakra kell hozni. Ez történhet egyszerre is. A mantissza akkor megfelelő alakú, ha a legfelső bitje magas (a mantissza 7. bitje). A feladat adott, addig kell kettővel szorozni a 32 bites számot, amíg a legfelső bit magas nem lesz. Az exponens a szorzások számából számítható ki. A ciklusok számából kivonunk 160-at, és minden bitet invertálunk. A keletkezett szám az exponens.

A mellékelt assembly programban mindkét említett átalakító rutin megtalálható. A BINFLO rutin binárisból lebegőpontosba számol, a FLOBIN fordítva. A rutinok részét képezik az adatcseréhez szükséges beolvasó és tároló részek is. A rutinok a bináris számot szöveges változóban tárolják, illetve keresik. Mindkét gépi rutin két paraméterrel dolgozik: szöveges típusú és lebegőpontos változóneveket kér.

A BINFLO hívása:

SYS *tárcím,* *szöveges változó,* *lebegőpontos változó*

Például: SYS 49152, C\$, K

A FLOBIN hívása:

SYS *tárcím,* *lebegőpontos változó,* *szöveges változó*

Például: SYS 49155, K, C\$

A BINFLO rutin ?ILLEGAL QUANTITY ERROR üzenetet ad ki, ha a megadott szöveges változó nem létezik még, vagy tartalma szerint üres. A rutinok assembly listája:

C000		100	*=\$C000	
AEFD		110	CHKCOM =\$AEFD	
B6A3		120	FRES TR =\$B6A3	
C000 4C 06 C0	130	JMP BINFLO	; Binárisból lebegőpontosba	
C003 4C 5A C0	140	JMP FLOBIN	; Lebegőpontosból binárisba	
C006 20 FD AE	150	BINFLO JSR CHKCOM	; Vessző ellenőrzése	
C009 20 8B B0	160	JSR \$B08B	; A változó megkeresése/létrehozása	
C00C 85 49	170	STA \$49	; A változó címe, LB	
C00E 84 4A	180	STY \$4A	; " , HB	
C010 A0 02	190	LDY #\$02		
C012 B1 49	200	AA0 LDA (\$49),Y	; A mutatók lemásolása	
C014 99 61 00	210	STA \$0061,Y	; a nulláslapra	
C017 88	220	DEY		
C018 10 F8	230	BPL AA0		
C01A AA	240	TAX	; A hossz nulla ?	
C01B F0 3A	250	BEQ ERROR	; Igen, hiba	
C01D A0 03	260	LDY #\$03		
C01F B1 62	270	AA1 LDA (\$62),Y	; A szöveg első négy	
C021 99 6A 00	280	STA \$006A,Y	; Karakterének betöltése az ARG-ba	
C024 88	290	DEY		
C025 10 F8	300	BPL AA1		
C027 A5 6A	310	AA4 LDA \$6A	; Az ARGH03 7. bitje magas ?	
C029 30 14	320	BMI AA2	; Igen, az átalakítás kész	
C02B C8	330	INY	; A lépésszámláló növelése	
C02C C0 20	340	CPY #\$20	; 32. lépés ?	
C02E B0 0B	350	BCS AA3	; Igen, befejezni az átalakítást	
C030 06 6D	360	ASL \$6D	; A mantissza	
C032 26 6C	370	ROL \$6C		
C034 26 6B	380	ROL \$6B	; balraléptetése	
C036 26 6A	390	ROL \$6A		
C038 4C 27 C0	400	JMP AA4	; Feltétel nélküli ugrás	
C03B A9 00	410	AA3 LDA #\$00	; Az exponens nulla	
C03D F0 06	420	BEQ AA5	; Feltétel nélküli ugrás	
C03F 98	430	AA2 TYA	; Lépésszámláló	
C040 38	440	SEC		
C041 E3 A0	450	SBC #\$A0	; 160 levonása	
C043 49 FF	460	EOR #\$FF	; Minden bit megfordítása	
C045 85 69	470	AA5 STA \$69	; Az exponens tárolása	
C047 20 FD AE	480	JSR CHKCOM	; Vessző ellenőrzése	
C04A 20 8B B0	490	JSR \$B08B	; A változó megkeresése/létrehozás	
C04D 85 49	500	STA \$49	; A változó címe, LB	
C04F 84 4A	510	STY \$4A	; " , HB	
C051 20 FC BB	520	JSR \$BBFC	; ARG a FAC-ba	
C054 4C D0 BB	530	JMP \$BBD0	; A FAC a változóba	
C057 4C 48 B2	540	ERROR JMP \$B248	; ILLEGAL QUANTITY ERROR	
C05A 20 FD AE	550	FLOBIN JSR CHKCOM	; Vessző ellenőrzése	
C05D 20 8B B0	560	JSR \$B08B	; A változó megkeresése/létrehozás	
C060 20 A2 BB	570	JSR \$BBA2	; A változó a FAC-ba	
C063 A5 61	580	LDA \$61	; Az exponens nulla ?	
C065 F0 11	590	BEQ AB0	; Igen, ugrás	
C067 A3 A0	600	LDA #\$A0	; A lépések számának	
C069 38	610	SEC		
C06A E5 61	620	SBC \$61	; Kiszámítása	
C06C AA	630	TAX	; Lépésszám az X-be	
C06D 46 62	640	AB1 LSR \$62	; A mantissza	
C06F 66 63	650	ROR \$63		
C071 66 64	660	RDR \$64	; jobbraléptetése	
C073 66 65	670	ROR \$65		

C075	CA	680	DEX	; A számláló lefutott ?
C076	D0 F5	690	BNE AB1	; Még nem, vissza
C078	20 0C BC	700 AB0	JSR \$BC0C	; A FAC az ARG-ba
C07B	A5 6A	710	LDA \$6A	; Az ARGEXP
C07D	29 7F	720	AND #\$7F	
C07F	85 6A	730	STA \$6A	; módosítása
C081	20 FD AE	740	JSR CHKCOM	; Vessző ellenőrzése
C084	20 8B B0	750	JSR \$B08B	; A változó megkeresése/létrehozása
C087	85 49	760	STA \$49	; A változó címe, LB
C089	84 4A	770	STY \$4A	; " " , HB
C08B	20 A3 B6	780	JSR FRESTR	; Szövegkezelés
C08E	A9 04	790	LDA #\$04	; A hossz 4 karakter
C090	20 75 B4	800	JSR \$B475	; Helyfoglalás a szövegnek
C093	A0 02	810	LDY #\$02	
C095	B9 61 00	820 AB2	LDA \$0061,Y	; A mutatók bemásolása
C098	91 49	830	STA (\$49),Y	; a változó fejlécébe
C09A	88	840	DEY	
C09B	10 F8	850	BPL AB2	
C09D	C8	860	INY	
C09E	B9 6A 00	870 AB3	LDA \$006A,Y	; Az ARG tartalmának elhelyezése
C0A1	91 62	880	STA (\$62),Y	; a szöveges változóban
C0A3	C8	890	INY	
C0A4	C4 61	900	CPY \$61	
C0A6	D0 F6	910	BNE AB3	
C0A8	60	920	RTS	
C0A9		930	.END	

ZEILEN:84 SYMBOLE:15 FEHLER:0

AA0	=C012	AA1	=C01F	AA2	=C03F	AA3	=C03B	AA4	=C027	AA5	=C045
AB0	=C078	AB1	=C06D	AB2	=C095	AB3	=C09E	BINFLO	=C006	CHKCOM	=AEFD
ERROR	=C057	FLOBIN	=C05A	FRESTR	=B6A3						

A BASIC-betöltő:

```

100 FOR I= 49152 TO 49320 :READ X:POKE I,X:S=S+X:NEXT
110 DATA 76, 6,192, 76, 90,192, 32,253,174, 32,139,176
120 DATA 133, 73,132, 74,160, 2,177, 73,153, 97, 0,136
130 DATA 16,248,170,240, 58,160, 3,177, 98,153,106, 0
140 DATA 136, 16,248,165,106, 48, 20,200,192, 32,176, 11
150 DATA 6,109, 38,108, 38,107, 38,106, 76, 39,192,169
160 DATA 0,240, 6,152, 56,233,160, 73,255,133,105, 32
170 DATA 253,174, 32,139,176,133, 73,132, 74, 32,252,187
180 DATA 76,208,187, 76, 72,178, 32,253,174, 32,139,176
190 DATA 32,162,187,165, 97,240, 17,169,160, 56,229, 97
200 DATA 170, 70, 98,102, 99,102,100,102,101,202,208,245
210 DATA 32, 12,188,165,106, 41,127,133,106, 32,253,174
220 DATA 32,139,176,133, 73,132, 74, 32,163,182,169, 4
230 DATA 32,117,180,160, 2,185, 97, 0,145, 73,136, 16
240 DATA 246,200,185,106, 0,145, 98,200,196, 97,208,246
250 DATA 96
260 IF S <> 20214 THEN PRINT "HIBA A DATASORBAN !":END
270 PRINT "OK !!":END

```

4.3. Néhány speciális BASIC program

4.3.1. Tárcímszerkesztő

Programfejlesztéskor gyakran előfordul, hogy egy tárcím tartalmától függ a program tevékenysége. Olykor úgy kell kilesni, hogy a tárcím egyes bitjének megváltozása milyen következményeket von maga után. Ez a program segítséget nyújt az ilyen munkákhoz.

A program egy tárcímet kér, amelynek tartalmát a PEEK utasítás beolvassa. Az 1000-es szubrutin feldolgozza a beolvasott byte-ot, és bináris formában kiírja a képernyőre. A program ezt követően a RETURN és az F1 — F8 billentyűkkel kezelhető. Az F1 — F7 billentyűkkel az alsó négy bit (rendre: 0 — F1; 1 — F3; 2 — F5; 3 — F7) módosítható. A megfelelő billentyű lenyomása invertálja a hozzá tartozó bitet. A bitet a kizáró VAGY logikai művelettel billentjük át a másik állapotba. Ezt követően a program a megadott tárcímre írja a byte új értékét, és visszatér a ciklus elejére. A RETURN billentyű hatására újraindul a program. Az INPUT utasításban nem decimális, hanem hexadecimális címet vár a program, a 600-as szubrutin ezt alakítja át decimálissá.

A program BASIC-listája:

```

10 POKE 53280,0:POKE 53281,0
100 INPUT "TÁRCÍM : ";R$
110 GOSUB 600 :R=Y
120 H$="0123456789ABCDEF."
130 M=PEEK(R)
140 GOSUB 1000
150 GETX$:IF X$="" THEN 150
160 IF X$=CHR$(13) THEN RUN
170 IF X$<CHR$(133) OR X$>CHR$(140) THEN 150
180 B=2*(ASC(X$)-133)
190 M=(MOR B) AND NOT (MANDB)
200 POKE R,M
210 GOTO 130

500 POKE 781,X:POKE 782,Y:SYS 58636:RETURN

600 Y=0
610 FOR I=1 TO LEN(R$)
620 A=48
630 IF MID$(R$,I,1)>"9" THEN A=55
640 Y=Y+(ASC(MID$(R$,I,1))-A)*16*(LEN(R$)-I)
650 NEXT:RETURN

1000 X=6:Y=5:GOSUB 500:Y=M
1010 X$="":FOR I=7 TO 0 STEP-1
1020 X$=X$+STR$(INT(Y/2^I))
1030 Y=Y-2^I*INT(Y/2^I)
1040 NEXT
1050 X$=X$+" ":Y=M
1060 FOR I=1 TO 0 STEP-1
1070 B=INT(Y/16^I)
1080 X$=X$+MID$(H$,B+1,1)
1090 Y=Y-B*16^I
1100 NEXT
1110 X$=X$+" "+STR$(M)
1120 PRINTX$
1130 RETURN

```


4.3.2. Speciális gépikód-szimulátor

A program mintegy „kilesi” a processzor utasításainak működését. Az olvasó ezzel a programmal ellenőrizheti a gépi nyelv utasításkészletét, az egyes utasítások hatását a tárra és a processzorra. A kötet gépi utasításokkal foglalkozó fejezetének példái is ennek segítségével készültek.

A program jellemző példa a BASIC és a gépi kód ötvözésére. A processzor utasításainak végrehajtása és a processzor regisztereinek olvasása BASIC-ből lehetetlen, még gépi nyelven is elég csavaros. Viszont a program egésze még BASIC-ben is elég terjedelmes, hát még gépi nyelven! Ezért itt a korábban már említett megoldás kapott szerepet: amit érdemes és meg lehet írni BASIC-ben, az abban van; amit nem lehet, az pedig gépi nyelvű. A gépi szekció tartalmazza azt az INPUT-rutint, amit a fejezet első példajaként ismertettünk. Így annak assembler listáját nem is közöljük teljes terjedelmében.

A program működésével kapcsolatban a következőket kell tudni: a képernyőmaszk megjelenése után a menü egyik (első) tagja inverzbe vált. A menüben a kurzort jobbra-balra irányító billentyűvel mozoghatunk. A képernyőmaszk felső sorai értelemszerűek, a processzor munkaregisztereit ábrázolják. Alatta a processzor állapotregiszterét és veremmutatóját találhatjuk. Ez alatt az a két tárcím található, amely nulláslap címezésnél és abszolút címezésnél operandusként szerepelhet az utasításkód mögött. Alatta az utasításkód és az operandusok következnek.

A következő sor a menü. A menü négy választási lehetőséget ad: START, MODIFY, MONITOR és END. A START hatására lezajlik a gépi utasítás végrehajtása és a regiszterek beolvasása, az új értékek megjelenítése. A MODIFY választása után a veremmutató kivételével valamennyi regiszter és tárcím vagy operandus tartalma felülírható. A beolvasás a gépi kódú INPUT-rutinnal történik, tehát az INPUT mezőkben előre-hátra mozoghatunk. Előre a RETURN-nel, hátra a balra mutató nyíllal. Kilépni az üzemmódból az első mezőben a balra mutató nyíllal, az utolsó mezőben a RETURN-nel lehetséges. A MONITOR arra szolgál, hogy az indexes utasítások eredményét is megnézhessük a tárban. A tárcím megadása után (hexadecimális címet kér), ugyancsak hexadecimális formában, megjelenik a tárcím tartalma. A vezérlés ezt követően automatikusan a menüre kerül.

A programmal a legtöbb gépi utasítás tesztelhető, van azonban néhány, a veremmutatót is módosító utasítás, amely kiakasztja a processzort. Ahhoz, hogy ezek az utasítások is lefussanak, a gépi rutint módosítani kell. A gépi rutin a kazettapufferben elhelyezett kommunikációs regisztereken keresztül érintkezik a BASIC-programmal. A BASIC-program:


```

1 GOTO 2000
2 POKE 781,X:POKE 782,Y:SYS 58636:RETURN
5 SYS 49152,A$,X,Y,H,M,1:RETURN
7 POKE 781,X:SYS 59903:RETURN

100 GOSUB 1000
110 GOSUB 1300
120 Q=1:GOSUB 2700
130 GET X$:IF X$="" THEN 130
140 IF X$=CHR$(13) THEN 180
150 IF X$=CHR$(29) THEN Q=Q+1:IF Q>4 THEN Q=1
160 IF X$=CHR$(257) THEN Q=Q-1:IF Q<1 THEN Q=4
170 GOSUB 2700:GOTO 130
180 ON Q GOTO 200,600,400,190
190 PRINT CHR$(147):END

200 GOSUB 1190
210 GOSUB 3200
220 GOSUB 3000
390 GOTO 120

400 GOSUB 1190
405 X=23:Y=2:GOSUB 2:POKE 781,23:SYS 59903
410 PRINT "TARCIM: $"
420 T=.:A$="":X=23:Y=11:H=4:M=1:GOSUB 5
430 IF LEN(A$)=. THEN 120
440 IF LEN(A$)=2 THEN 480
450 X$=LEFT$(A$,2)
460 GOSUB 1700
470 T=256*R
480 X$=RIGHT$(A$,2)
490 GOSUB 1700
500 T=T+R
510 T=PEEK(T)
520 X=23:Y=20:GOSUB 2
530 R=T:GOSUB 1600
540 PRINT "$"X$
550 GOTO 120

600 GOSUB 1190
610 GOSUB 2100
620 GOSUB 2400
630 GOSUB 2800
640 GOTO 120

1000 PRINT CHR$(155)CHR$(147)CHR$(14)CHR$(8)
1010 X=1:Y=2:GOSUB 2
1020 PRINT "AC : $00 #0 %00000000"
1030 X=3:Y=2:GOSUB 2
1040 PRINT "XR : $00 #0 %00000000"
1050 X=5:Y=2:GOSUB 2
1060 PRINT "YR : $00 #0 %00000000"
1070 X=7:Y=2:GOSUB 2
1080 PRINT " NV#BD1ZC"
1090 X=8:Y=2:GOSUB 2
1100 PRINT "PS : $00 #0 %00100000"
1110 X=10:Y=2:GOSUB 2
1120 PRINT "SP : $ # "
1130 X=13:Y=1:GOSUB 2
1140 PRINT "$EA : $00 #0 %00000000"
1150 X=15:Y=1:GOSUB 2
1160 PRINT "$EAEA: $00 #0 %00000000"
1170 X=18:Y=2:GOSUB 2
1180 PRINT "OP 1: $EA 2: $EA 3: $EA"
1190 X=20:Y=2:GOSUB 2
1200 PRINT " START MODIFY MONITOR END "
1210 RETURN

1300 AC$="00000000":XR$=AC$:YR$=AC$
1310 PS$="00100000":ZP$=AC$:ME$=AC$

```

```

1320 01$="EA":02$=01$:03$=01$:01=234:02=01:03=01
1330 H$="0123456789ABCDEF"
1399 RETURN

1400 REM BINARISBOL DECIMALISBA
1410 R=0:FOR I=1 TO 8:R=R+ABS((MID$(X$,8-I+1,1))>"0"))*2^(I-1):NEXT
1420 RETURN

1500 REM DECIMALISBOL BINARISBA
1510 A=R:X$="":FOR I=7 TO 0 STEP-1:B=INT(R/2^I)
1520 X$=X$+MID$(STR$(B),2)
1530 R=R-B*2^I:NEXT:I:R=A
1540 RETURN

1600 REM DECIMALISBOL HEXADECIMALISBA
1610 A=R:X$="":FOR I=1 TO 0 STEP-1:B=INT(R/16^I):X$=X$+MID$(H$,B+1,1)
1620 R=R-B*16^I:NEXT:I:R=A
1630 RETURN

1700 REM HEXADECIMALISBOL DECIMALISBA
1710 R=.:FOR I=1 TO 2:A=48:IF MID$(X$,I,1)>"9" THEN A=55
1720 R=R+(ASC(MID$(X$,I,1))-A)*16^(2-I)
1730 NEXT:I:RETURN

2000 POKE 53280,0:POKE 53281,0
2010 PRINT CHR$(147)CHR$(155)
2020 POKE 650,128
2040 IF PEEK(49153)<>6 OR PEEK(49683)<>177 THEN GOSUB 4000
2050 GOTO 100

2100 REM MODIFY INPUT
2110 X=1:A$=AC$:GOSUB 2300:AC$=A$
2120 IF IRQ THEN RETURN
2130 X=3:A$=XR$:GOSUB 2300:XR$=A$
2140 IF IRQ THEN 2110
2150 X=5:A$=YR$:GOSUB 2300:YR$=A$
2160 IF IRQ THEN 2130
2170 X=8:A$=PS$:GOSUB 2300:PS$=A$
2180 IF IRQ THEN 2150
2210 X=13:A$=ZP$:GOSUB 2300:ZP$=A$
2220 IF IRQ THEN 2170
2230 X=15:A$=ME$:GOSUB 2300:ME$=A$
2240 IF IRQ THEN 2210
2250 Y=10:A$=01$:GOSUB 2350:01$=A$
2260 IF IRQ THEN 2230
2270 Y=19:A$=02$:GOSUB 2350:02$=A$
2280 IF IRQ THEN 2250
2290 Y=28:A$=03$:GOSUB 2350:03$=A$
2295 IF IRQ THEN 2270
2299 RETURN
2300 Y=22:H=8:M=0:GOTO 5
2350 X=18:H=2:M=1:GOTO 5

2400 REM ATALAKITASOK
2410 A$=AC$:X=1:GOSUB 2600:AC$=R
2420 A$=XR$:X=3:GOSUB 2600:XR$=R
2430 A$=YR$:X=5:GOSUB 2600:YR$=R
2440 A$=PS$:X=8:GOSUB 2600:PS$=R
2450 R=SP:X=10:A$="":GOSUB 2610
2460 A$=ZP$:X=13:GOSUB 2600:ZP$=R
2470 A$=ME$:X=15:GOSUB 2600:ME$=R
2480 X$=01$:GOSUB 1700:01$=R
2490 X$=02$:GOSUB 1700:02$=R
2500 X$=03$:GOSUB 1700:03$=R
2510 X=13:Y=2:GOSUB 2:PRINT 02$
2520 X=15:Y=2:GOSUB 2:PRINT 03$+02$
2600 X$=A$:GOSUB 1400
2610 GOSUB 1600
2620 Y=9:GOSUB 2:PRINT X$ " "
2630 Y=15:GOSUB 2:PRINT MID$(STR$(R),2) " "

```

```

2640 Y=22:GOSUB 2:PRINT A$
2650 RETURN

2700 REM MENU MODUL
2710 X=20:Y=2:GOSUB 2:ON Q GOTO 2730,2740,2750,2760
2720 RETURN

2730 PRINT "  START  MODIFY  MONITOR  END  ":RETURN
2740 PRINT "  START  MODIFY  MONITOR  END  ":RETURN
2750 PRINT "  START  MODIFY  MONITOR  END  ":RETURN
2760 PRINT "  START  MODIFY  MONITOR  END  ":RETURN

2800 REM PSZEUDO REGISZTEREK BEALLITASA
2810 POKE 840,AC
2820 POKE 841,XR
2830 POKE 842,YR
2840 POKE 843,PS OR 32
2850 POKE 02,ZP
2860 POKE 256*03+02,ME
2870 POKE 49863,01
2880 POKE 49864,02
2890 POKE 49865,03
2900 RETURN

3000 REM A PSZEUDO REGISZTEREK OLVASASA
3010 AC=PEEK(840)
3020 XR=PEEK(841)
3030 YR=PEEK(842)
3040 PS=PEEK(843)
3050 SP=PEEK(844)
3060 ZP=PEEK(02)
3070 ME=PEEK(256*03+02)
3080 R=AC:X=1:GOSUB 1500:AC$=X$:A$=X$:GOSUB 2610
3090 R=XR:X=3:GOSUB 1500:XR$=X$:A$=X$:GOSUB 2610
3100 R=YR:X=5:GOSUB 1500:YR$=X$:A$=X$:GOSUB 2610
3110 R=PS:X=8:GOSUB 1500:PS$=X$:A$=X$:GOSUB 2610
3120 R=SP:X=10:A$=" ":GOSUB 2610
3130 R=ZP:X=13:GOSUB 1500:ZP$=X$:A$=X$:GOSUB 2610
3140 R=ME:X=15:GOSUB 1500:ME$=X$:A$=X$:GOSUB 2610
3150 RETURN

3200 REM FUTTATAS
3210 SYS 49155
3220 RETURN

4000 FOR I= 49152 TO 49889 :READ X:POKE I,X:S=S+X:NEXT
4010 DATA 76, 6,192, 76,180,194, 32,213,193, 32,249,193
4020 DATA 32,224,193,165,178,133,179, 32,178,193, 32,136
4030 DATA 193,169, 0,133,204,133,198, 32, 62,241,240,251
4040 DATA 201, 34,240,247,201, 13,208, 3, 76, 72,193,201
4050 DATA 95,208, 3, 76, 69,193,201, 44,240, 8,201, 59
4060 DATA 240, 4,201, 58,208, 4, 36,177, 80,217,201, 20
4070 DATA 208, 3, 76,127,192,201, 29,208, 3, 76,203,192
4080 DATA 201,148,208, 3, 76,232,192,201,157,208, 3, 76
4090 DATA 220,192,166,176,240, 11,201, 32,144,181,201, 95
4100 DATA 176,177, 76, 26,193,201, 46,240,249,201, 48,144
4110 DATA 166,201, 58,176,162,144,239,165,179,240, 28,197
4120 DATA 178,144, 27,208, 87, 32,123,193,169,157, 32,210
4130 DATA 255,169, 32, 32,210,255,169,157, 32,210,255,198
4140 DATA 178,198,179, 76, 25,192, 32,123,193,169,157, 32
4150 DATA 210,255,164,179, 32,237,193,169, 32, 32,210,255
4160 DATA 164,179,185,139,194,153,138,194,200,196,178,208
4170 DATA 245,169, 32,153,138,194,198,178, 76,227,192,165
4180 DATA 179,197,166,240,206, 32,123,193,230,179, 32,202
4190 DATA 193, 76, 25,192,165,179,240,191, 32,123,193,198
4200 DATA 179, 76,214,192,165,179,197,178,176,177,165,178
4210 DATA 197,166,240,171, 32,123,193,169, 32, 32,210,255
4220 DATA 164,179, 32,237,193,198,179,164,178,185,139,194
4230 DATA 153,140,194,136,196,179,208,245,200,169, 32,153
4240 DATA 139,194,230,178,208,186,133,255, 32,123,193,165

```



```

4250 DATA 179,197,166,240, 16,197,178,176, 15,165,255, 32
4260 DATA 210,255,164,179,153,139,194,230,179, 76, 25,192
4270 DATA 208, 4,230,178,208,235,166,179,232,134,178,208
4280 DATA 228, 32,142,193,162, 0,134, 14,202,134, 13,165
4290 DATA 253,133, 69,165,254,133, 70, 32,231,176,133, 73
4300 DATA 132, 74,165,178, 32,117,180,160, 2,185, 97, 0
4310 DATA 145, 73,136, 16,248,170,240, 11,200,185,139,194
4320 DATA 145, 98,200,196, 97,208,246,169, 2,133,205,165
4330 DATA 207,208,252,169, 1,133,204, 96,169, 0, 72, 72
4340 DATA 240, 6,169, 0, 72,169, 1, 72,169, 0,133, 14
4350 DATA 133, 13,169, 73,160, 82,133, 69,132, 70, 32,231
4360 DATA 176,133, 73,132, 74,104,168,104, 32,145,179, 76
4370 DATA 208,187, 36,177, 48, 41,162, 24,160, 30, 32, 12
4380 DATA 229,160, 0,185,128,194,240, 6, 32,210,255,200
4390 DATA 208,245,166,158,165,159, 24,101,179,168, 76, 12
4400 DATA 229,169, 32,160, 40,153,139,194,136, 16,250, 96
4410 DATA 166,158,164,159, 32, 12,229,165,178,240, 13,160
4420 DATA 0,185,139,194, 32,210,255,200,196,178,208,245
4430 DATA 96, 32,253,174, 32,139,176,133, 73,132, 74,160
4440 DATA 2,177, 73,153, 97, 0,136, 16,248,170,134,178
4450 DATA 240, 15,200,177, 98,153,139,194,200,196, 97,240
4460 DATA 4,192, 40,144,242,165, 69,133,253,165, 70,133
4470 DATA 254, 32,241,183,224, 25,176, 77,134,158, 32,241
4480 DATA 183,224, 40,176, 68,134,159, 32,241,183,138,240
4490 DATA 60, 24,101,159,201, 40,176, 53,228,178,176, 2
4500 DATA 134,178,134,166, 32,241,183,134,176,162, 0, 32
4510 DATA 121, 0,201, 44,208, 28, 32,241,183,138,240, 22
4520 DATA 201, 1,208, 4,162,128,208, 14,201, 2,208, 4
4530 DATA 162, 64,208, 6,201, 3,208, 5,162,192,134,177
4540 DATA 96, 76, 72,178, 18, 86, 73, 83, 83, 90, 65, 58
4550 DATA 95,146, 0, 32, 32, 32, 32, 32, 32, 32, 32, 32
4560 DATA 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32
4570 DATA 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32
4580 DATA 32, 32, 4,169, 46,208, 3,185,186,142, 77, 3
4590 DATA 173, 75, 3, 72,173, 72, 3,174, 73, 3,172, 74
4600 DATA 3, 40,234,234,234,234,234,141, 72, 3,142, 73
4610 DATA 3,140, 74, 3, 8,104,141, 75, 3,186,142, 76
4620 DATA 3,174, 77, 3,154, 96
4630 IF S<>97593 THEN PRINT "HIBA A DATASORBAN !":END
4640 RETURN

```

A gépi rész assembly listája:

C000	100	*=\$C000		
	...			; Értékekadások
	...			
C000	4C 06 C0	240	JMP INPUTS	; Input-rutin
C003	4C B4 C2	250	JMP RUNNER	; Szimulátor
C006	20 D5 C1	260	INPUT JSR ERASE	
C009	20 F9 C1	270	JSR PARAM	
C00C	20 E0 C1	280	JSR OUT	
			; Az input-rutin
			
C28B	3270 BUF	=*		
C2B4	3280	*=**+41		
C2B4	BA	3290	RUNNER TSX	; A veremmutató
C2B5	8E 4D 03	3300	STX 845	; elmentése
C2B8	AD 4B 03	3310	LDA 843	; A processzor státusz
C2BB	48	3320	PHA	; a verembe
C2BC	AD 48 03	3330	LDA 840	; Az akkumulátor,
C2BF	AE 49 03	3340	LDX 841	; az X regiszter,
C2C2	AC 4A 03	3350	LDY 842	; és az Y regiszter feltöltése
C2C5	28	3360	PLP	; A processzor státusz betöltése
				; a veremből

C2C6	EA	3370	NOP	
C2C7	EA	3380	OP1	NOP ; Ide kerülnek a műveleti kódok
C2C8	EA	3390	NOP	; A címzés helye
C2C9	EA	3400	NOP	
C2CA	EA	3410	NOP	
C2CB	8D 48 03	3420	STA 840	; Az akkumulátor tárolása
C2CE	8E 49 03	3430	STX 841	; Az X regiszter tárolása
C2D1	8C 4A 03	3440	STY 842	; Az Y regiszter tárolása
C2D4	08	3450	PHP	; A processzor státusz a verembe
C2D5	68	3460	PLA	; Majd onnan az akkuba
C2D6	8D 4B 03	3470	STA 843	
C2D9	BA	3480	TSX	; A veremmutató az X-be
C2DA	8E 4C 03	3490	STX 844	; Tárolás
C2DD	AE 4D 03	3500	LDX 845	; Az eredeti veremmutató
C2E0	9A	3510	TXS	; visszatöltése
C2E1	60	3520	RTS	
C2E2		3530	.END	

ZEILEN:344 SYMBOLE:75 FEHLER:0

4.3.3. BASIC és gépi kód — autostart program

Az autostartolt BASIC-programok sem képzelhetők el gépi nyelvű indító nélkül. Az autostart programoknak számos változata létezik, egyet már bemutattunk a gépi nyelvvel foglalkozó fejezetben. Az itt ismertetett változat kifejezetten BASIC-programok indítására szolgál. Az eredeti program autostartos változata mintegy 5 blokkal lesz nagyobb a lemezen, ez az indító elhelyezkedése miatt van így. Az új program neve azonos lesz az eredetivel, de előtte egy pont található. A program indítás után először az indítót írja fel a file-ba, majd az üres képernyőt, ezután átmásolja az eredeti állományt. A gépi rutin a CHRIN vektort írja át, majd a betöltés végeztével vissza is írja az eredeti értékre. A program betöltése után ezen a vektoron is keresztülhalad a programszámláló. A módosított vektor az indítórutinra mutat. A vektor visszaírása után indirekt módon utasítja az interpretert a program elindítására: a billentyűzetpufferben RUN parancsot szimulál. Az interpreter ezt úgy hajtja végre, mintha valóban begépeltek volna. A BASIC-program listája:

```

100 POKE53280,0:POKE53281,0:PRINTCHR$(47)CHR$(30)
110 INPUT "PROGRAMNEV: ";N$
120 OPEN 15,8,15,"1":CLOSE15
130 OPEN 2,8,2,"0:"+N$+",P,R"
140 GET#2,A$,B$
150 OPEN 3,8,3,"00:."+N$+",P,W"
160 PRINT#3,CHR$(36);CHR$(3);
170 PRINT#3,CHR$(128);CHR$(3);
180 FOR I=806 TO 827
190 PRINT#3,CHR$(PEEK(I));:NEXT
200 FOR I=828 TO 895
210 PRINT#3,CHR$(32);:NEXT
220 FOR I=896 TO 930: READ A
230 PRINT#3,CHR$(A);:NEXT
240 FOR I=931 TO 2047
250 PRINT#3,CHR$(32);:NEXT
260 PRINT#3,CHR$(0);
270 GET#2,A$
280 IF A$="" THEN A$=CHR$(0)

```



```

290 IF ST THEN 310
300 PRINT#3,A$;:GOTO 270
310 PRINT#3,A$: CLOSE 3
320 CLOSE2
330 PRINT" OK!"
340 END
350 REM GEPIKOD
360 DATA 169, 87,141, 36, 3,169,241
370 DATA 141, 37, 3,169, 4,133,198
380 DATA 169, 82,141,119, 2,169
390 DATA 85,141,120, 2,169, 78,141
400 DATA 121, 2,169, 13,141,122, 2
410 DATA 96

```

A gépi rész assembly listája:

0380		100	*= \$0380	
0380	A9 57	110	LDA #\$57	; a CHRIN vektorának
0382	8D 24 03	120	STA \$0324	
0385	A9 F1	130	LDA #\$F1	; helyreállítása
0387	8D 25 03	140	STA \$0325	
038A	A9 04	150	LDA #\$04	; 4 karakter a billentyűzetpufferben
038C	85 C6	160	STA \$C6	; a puffer mutatója
038E	A9 52	170	LDA #"R"	
0390	8D 77 02	180	STA \$0277	; a puffer
0393	A9 55	190	LDA #"U"	
0395	8D 78 02	200	STA \$0278	
0398	A9 4E	210	LDA #"N"	
039A	8D 79 02	220	STA \$0279	; feltöltése
039D	A9 0D	230	LDA #\$0D	; RETURN
039F	8D 7A 02	240	STA \$027A	
03A2	60	250	RTS	
03A3		260	.END	

ZEILEN:17 SYMBOLE:0 FEHLER:0

4.3.4. Programgenerátor programból — a BASIC-betöltők előállítása

Aki még nem ismeri az itt ismertetett trükköt, bizonyára elképedve nézi a néha több tucat DATA-sort, amelyek mindegyike tíz-tizenkét számot tartalmaz. Az járhat a fejében, ki az a fanatikus őrült, aki ilyen hosszú és unalmas programot képes begépelni úgy, hogy közben ráadásul nem is téved. Nos, már itt elárulhatjuk, ezeket a programokat valójában a C64-es készítette, és a programozónak csak a kezdő- és végcímet kellett megadnia. A dolog egy ötletes trükkre épül: a program befejezése előtt a képernyőre megfelelő helyeken kiírunk néhány sort, a billentyűzetpuffert pedig megfelelő számú RETURN karakterrel töltjük fel. Az END utasítás után a végrehajtás az interpreter várakozó ciklusára adódik, amely azonnal érzékeli, hogy a billentyűzetpufferben karakterek vannak, és ezeket végre is hajtja. Csak arra kell vigyázni, hogy a kiírt sorok megfelelő helyen legyenek, és az utolsó sor valamilyen RUNxx utasítás legyen. Ha a sorok egyike szabályos programsor, az interpreter (mert valójában direkt módban van) azt azonnal beépíti a programba. A változók azonban elvesznek, ezért gon-

doskodni kell tárolásukról a RAM más területén. A program újraindítása után visszaolvassa a futó változókat, és megkezdzi az új sor felépítését. Ezután az egész ciklus kezdődik előlről: a BASIC program formálisan véget ér, az interpreter direkt módban végrehajtja azt, amit még az előző program szabott meg neki, és visszaugrik a programba. Ez egészen addig tart, amíg a futóváltozó el nem éri a végcímet, amelyet még a program elején adtunk meg.

Az említett trükk technikailag:

- (1) a programnak törölnie kell a képernyőt,
- (2) a legfelső sorba be kell írnia azt a sort, amely BASIC-sorként fog szerepelni,
- (3) a következő utasítást a harmadik sorba kell írni, mert kiíródik a READY is; az utasítás valamilyen RUNxx legyen,
- (4) fel kell tölteni a billentyűzetpuffert, amely a 631-es címen kezdődik: az első byte 19, ami a HOME kódja, a második és harmadik 13, ami a RETURN kódja,
- (5) végül a nulláslapon be kell állítani a billentyűzetpufferben tárolt karakterek számát – ami esetünkben három.

Ennél bonyolultabb utasítássort is össze lehet állítani, mert a billentyűzetpuffer hossza 10 karakter, és a képernyőn is van még hely. A programot a HELP PLUS alatt érdemes futtatni, mert amikor a program kiírja záróüzenetét, a kész DATA-betöltő elől le kell törölni a MAKER programot. Erre a HELP PLUS #D parancs alkalmas (#D—99).

A programgenerátor BASIC-programja:

```

10 POKE 53280,0:POKE 53281,0
11 PRINT CHR$(147)
12 INPUT "KEZDO CIM=";A:A2=A
13 INPUT "VEGCIM=";B:B2=B
14 IF B<A THEN 10
15 A1=INT (A/256):POKE 829,A1
16 POKE 833,A1:A=A-A1*256
17 POKE 828,A:POKE 832,A
18 B1=INT (B/256):POKE 831,B1
19 B=B-B1*256:POKE 830,B
20 C=110:C1=INT (C/256)
21 POKE 835,C1:C=C-C1*256
22 POKE 834,C
23 A$="100 FOR I="+STR$(A2)
24 A$=A$+" TO "+STR$(B2)
25 A$=A$+": READ A:POKE I,A:S=S+A:NEXT"
26 PRINT CHR$(147);
27 PRINT A$
28 PRINT "RUN31"
29 POKE 631,19:POKE 632,13
30 POKE 633,13:POKE 196,3:END
31 A$=""
32 C=PEEK (833)*256+PEEK (832)
33 E=PEEK (831)*256+PEEK (830)
34 D=PEEK (835)*256+PEEK (834)
35 A$=STR$(D)+" DATA ":S$=""
36 FOR I=C TO C+9
37 A=PEEK (I):B$=MID$(STR$(A),2)
38 A$=A$+RIGHT$(S$,3-LEN (B$))+B$+" ,"
39 IF I=E THEN I=I+1:GOTO41
40 NEXT I
41 A$=LEFT$(A$,LEN (A$)-1)
42 B$="RUN31"
43 IF I>E THEN B$="RUN52"
44 C=C+10:D=D+10
45 C1=INT (C/256):POKE 833,C1

```

```

46 C=C-C1*256:POKE 832,C
47 D1=INT(D/256):POKE 835,D1
48 D=D-D1*256:POKE 834,D
49 PRINT CHR$(147);A$:PRINT B$
50 POKE 631,19:POKE 632,13
51 POKE 633,13:POKE 198,3:END
52 A=PEEK (828)+PEEK (829)*256
53 B=PEEK (830)+PEEK (831)*256
54 D=PEEK (834)+PEEK (835)*256
55 FOR I=A TO B:READ A:S=S+A:NEXT
56 A$=STR$(D+10)+" IF S <>"
57 A$=A$+STR$(S)+" THEN PRINT"
58 A$=A$+CHR$(34)+"HIBA A DATASORBAN !"
59 A$=A$+CHR$(34)+" :END"
60 PRINT CHR$(147);A$:PRINT "RUN63"
61 POKE 631,19:POKE 632,13
62 POKE 633,13:POKE 198,3:END
63 PRINT CHR$(147)
64 POKE 211,6:POKE 214,6:SYS 58732
65 PRINT "AZ ONALLO DATABETOLTO KESZ."
66 POKE 211,6:POKE 214,8:SYS 58732
67 PRINT "SZAMOZASA 100-TOL KEZDODIK."
68 END

```

4.4. A ROM-ok felhasználhatósága

Gépi nyelvű programozás során hallatlanul nagy segítséget jelenthet az, hogy egyes szubrutinokat nem kell megírni, mert a célnak teljesen megfelelő rutint a BASIC vagy KERNAL ROM tartalmazza. Ebben a részben igyekeztünk összefoglalni azokat a szubrutinokat és belépési pontokat, amelyekhez gyakorlati alkalmazás képzelhető el. Ettől függetlenül előfordulhat, hogy egyes, ritkábban használható tippek kimaradtak. A megadott belépési pontokhoz tartozó szubrutinok kezeléséhez szükséges információkat tömörített formában közreadjuk.

1. Blokkléptető rutin \$A3BF

Használt regiszterek: A, X, Y

Bemeneti adatok:

\$5F — 60: az eredeti blokk kezdete

\$5A — 5B: az eredeti blokk vége + 1

\$58 — 59: az új blokk vége + 1

A rutin helyet biztosít beszúrható adatállomány részére. Csak előre léptet, tehát törlésre nem használható.

2. Programszámláló a BASIC programkezdetre \$A68E

Használt regiszter: A

Bemeneti adatok: —

A rutin a CHRGET rutin (\$0073) programszámlálóját állítja be az aktuális programkezdetre (\$2B — 2C).

3. Szöveg kiírása \$AB1E

Használt regiszterek: A, X, Y

Bemeneti adatok:

A: a kiírni kívánt szöveg címe, alsó byte

Y: a kiírni kívánt szöveg címe, felső byte

A rutin kiírja az A/Y-ban megadott kezdőcímű szöveget az aktuális kimeneti egységre. A szövegnek \$00-ra kell végződnie.

4. Szóköz \$AB3F

kurzor jobbra

\$AB42

kérdőjel kiírása

\$AB45

Használt regiszter: A

Bemeneti adat: —

A rutin kiírja a belépési pontnak megfelelő karaktert.

5. Numerikus kifejezés kiértékelése, FRMNUM \$AD8A

Használt regiszterek: A, X, Y

Bemeneti adat: —

A rutin tetszőleges mélységű BASIC-kifejezést értékel ki, kiszámítja annak eredményét, amely a FAC-ba kerül.

6. Tetszőleges BASIC-kifejezés kiértékelése, FRMEVL \$AD9E

Használt regiszterek: A, X, Y

Bemeneti adat: —

A rutin része az előző (FRMNUM) rutinnak, ezért az ott leírtak érvényesek.

7. π , lebegőpontos állandó, 3,141592654 \$AEA8

8. Nyitó zárójel \$AEF7

záró zárójel

\$AEFA

vessző CHKCOM

\$AEFD

tetszőleges karakter ellenőrzése

\$AEFF

Használt regiszter: A, Y

Bemeneti adat: A, az ellenőrzendő karakter (csak \$AEFF)

Ellenőrzi a következő karaktert a BASIC-szövegben. Ha nem a várt karakter következik, SYNTAX ERROR üzenetet kapunk.

9. Változó létesítése vagy megkeresése \$B08B

Használt regiszterek: A, X, Y

Bemeneti adatok: BASIC-szövegben a változó neve

A rutin megkeresi vagy létrehozza a megadott változót. A változó címe visszatéréskor a A/Y regiszterekben van.

10. –32768 lebegőpontos állandó \$B1A5

11. Lebegőpontos — egész átalakítás \$B1AA

Használt regiszterek: A, X, Y

Bemeneti adat: a FAC tartalma

A lebegőpontos változó tartalmát átszámítja egész formátumra. Az eredmény az A/Y regiszterben jelenik meg.

12. Egész — lebegőpontos átalakítás \$B391

Használt regiszterek: A, X, Y

Bemeneti adatok: A/Y, egész érték, alsó-felső byte

A megadott 16 bites érték átalakítása lebegőpontos formátumra. Az eredmény a FAC-ban jelenik meg.

13. Érvénytelen szövegek megszüntetése \$B526

Használt regiszterek: A, X, Y

Bemeneti adatok: —

A rutin akkor használható, ha nagy tömegű, gyakran változó tartalmú szöveges változót használunk. A rutin tömöríti a szöveges területet.

14. Az érvénytelen ideiglenes szövegek mutatóinak megszüntetése, FRESTR \$B6A3

Használt regiszterek: A, X, Y

Bemeneti adatok: A/Y, a feldolgozandó szöveg mutatója

Bevezeti a megadott szöveget az ideiglenes verembe, meghatározza a hosszát, amely visszatéréskor az akkumulátorban van.

15. 8 bites paraméter beolvasása, GETBYT \$B79E

Használt regiszterek: A, X, Y

Bemeneti adatok: Paraméter BASIC szövegben

A rutin beolvas egy 8 bites paramétert BASIC szövegből. A paraméter az X regiszterbe kerül.

16. 16 és 8 bites paraméterek beolvasása, GETADR és GETBYT \$B7EB
 Használt regiszterek: A, X, Y
 Bemeneti adatok: paraméterek BASIC-szövegben, vesszővel elválasztva.
 A 16 bites egész a \$14—15 regiszterekbe, a 8 bites egész az X regiszterbe kerül.

17. Vessző ellenőrzése és 8 bites paraméter beolvasása, CHKCOM és GETBYT \$B7F1
 Programozás során gyakran állnak egymás után, egyszerűsítést tesz lehetővé.

18. 16 bites egész beolvasása, GETADR \$B7F7
 Használt regiszterek: A, X, Y
 Bemeneti adatok: paraméter BASIC-szövegben
 A 16 bites paraméter a \$14—15 regiszterekbe kerül.

19. Összeadás, $FAC = FAC + 0,5$ \$B849
 Használt regiszterek: A, X, Y
 Bemeneti adatok: a FAC tartalma

20. Kivonás, $FAC = (A/Y) - FAC$ \$B850
 Használt regiszterek: A, X, Y
 Bemeneti adatok: A/Y, a kisebbbítendő mutatója; FAC, a kivonandó
 A rutin a kisebbbítendőt az ARG-ba másolja. Az eredmény a FAC-ban jelenik meg.

21. Kivonás, $FAC = ARG - FAC$ \$B853
 Mint fent, de a kisebbbítendő már az ARG-ban van.

22. Összeadás, $FAC = (A/Y) + FAC$ \$B867
 Használt regiszterek: A, X, Y
 Bemeneti adatok: A/Y, az egyik az összeadandó mutatója; FAC, a másik az összeadandó

23. Összeadás, $FAC = ARG + FAC$ \$B86A
 Mint fent, de az összeadandók egyike az ARG-ban van.

24. Szorzás, $FAC = (A/Y) * FAC$ \$BA28
 Használt regiszterek: A, X, Y
 Bemeneti adatok: A/Y, a szorzandó mutatója; FAC, a szorzó

-
25. Szorzás, $FAC = ARG * FAC$ \$BA2B
Mint fent, de a szorzandó az ARG-ban van.
26. Az ARG feltöltése \$BA8C
Használt regiszterek: A, Y
Bemeneti adatok: A/Y, a betöltendő szám mutatója
27. Szorzás, $FAC = FAC * 10$ \$BAE2
Használt regiszterek: A, X, Y
Bemeneti adatok: a FAC tartalma
A kiszámított érték a FAC-ban jelenik meg.
28. A 10, mint lebegőpontos állandó \$BAF9
29. Osztás, $FAC = FAC/10$ \$BAFE
Használt regiszterek: A, X, Y
Bemeneti adatok: a FAC tartalma
Az eredmény a FAC-ban jelenik meg.
30. Osztás, $FAC = (A/Y) / FAC$ \$BB0F
Használt regiszterek: A, X, Y
Bemeneti adatok: A/Y, az osztandó mutatója; FAC, az osztó
Az eredmény a FAC-ban jelenik meg.
31. Osztás, $FAC = ARG / FAC$ \$BB12
Mint fent, de az osztandó az ARG-ban van.
32. A FAC feltöltése \$BBA2
Használt regiszterek: A, Y
Bemeneti adatok: A/Y, a betöltendő szám mutatója
33. A FAC átvitele az akku#4-be \$BBC7
Használt regiszterek: A, X, Y
Bemeneti adatok: a FAC tartalma
34. A FAC átvitele az akku#3-ba \$BBCA
Mint fent.
35. A FAC átvitele változóba \$BBD4
Használt regiszterek: A, X, Y
Bemeneti adatok: X/Y, a változó címe, a FAC tartalma

36. Az ARG átvitele a FAC-ba \$BBFC
 Használt regiszterek: A, X
 Bemeneti adatok: az ARG tartalma
37. A FAC átvitele az ARG-ba \$BC0C
 Használt regiszterek: A, X
 Bemeneti adatok: a FAC tartalma
38. ASCII szöveg (szám) átalakítása lebegőpontosá \$BCF3
 Használt regiszterek: A, X, Y
 Bemeneti adatok: BASIC-szöveg
39. Egész szám kiírása ASCII formátumban \$BDCD
 Használt regiszterek: A, X, Y
 Bemeneti adatok: A/X, a kiírandó egész értéke
- A rutin először átalakítja lebegőpontosá az egész számot, majd a lebegőpontos értéket ASCII formátumra alakítja és kiírja.
40. 0,5 lebegőpontos állandó \$BF11
41. Hatványozás, $FAC = ARG \uparrow (A/Y)$ \$BF78
 Használt regiszterek: A/Y, a kitevő mutatója
 ARG, a mantissza
42. Hatványozás, $FAC = ARG \uparrow FAC$ \$BF7B
 Használt regiszterek: FAC, kitevő
 ARG, mantissza
43. Az OPEN paramétereinek beolvasása \$E219
 Használt regiszterek: A, X, Y
 Bemeneti adatok: BASIC-paraméterek
- A rutin beolvassa a BASIC kifejezést és beállítja a KERNAL rutinok paramétereit.
44. BASIC hidegindítás \$E394
 Használt regiszterek: A, X, Y
 Bemeneti adat: —
- A rutin inicializálja a BASIC változóterületet, kiírja a bekapcsolási üzenetet, és beállítja a BASIC-vektorokat.

- 45.** Az I/O terület báziscímének beolvasása, IOBASE \$FFF3 — \$E500
Használt regiszterek: X, Y
Bemeneti adatok: —
A rutin a CIA1 kezdőcímével tér vissza az X/Y regiszterekben.
- 46.** A képernyőméret beolvasása, SCREEN \$E505
Használt regiszterek: X, Y
Bemeneti adat: —
A rutin a képernyő sorainak (X) és oszlopainak (Y) számát olvassa be.
- 47.** A kurzor beállítása \$E50C
a kurzor beolvasása \$E513
Használt regiszterek: A, X, Y
Bemeneti adatok: X/Y, sor és oszlopszám (csak \$E50C)
- 48.** Képernyő RESET \$E518
Használt regiszterek: A, X
Bemeneti adatok: —
A rutin inicializálja a képernyőt és a billentyűzet olvasásához szükséges változókat.
- 49.** A képernyő törlése \$E544
Használt regiszterek: A, X, Y
Bemeneti adat: —
- 50.** A kurzor a bal felső sarokban, HOME \$E566
Használt regiszterek: A, X, Y
Bemeneti adatok: —
- 51.** A videovezérlő RESET-je \$E5A0
Használt regiszterek: A, X
Bemeneti adatok: —
- 52.** Az X. képernyősor törlése \$E9FF
Használt regiszterek: A, X, Y
Bemeneti adat: X, a törlendő sor száma
- 53.** TALK küldése \$FFB4 — \$ED09
Használt regiszterek: A, X
Bemeneti adat: A, a megcímzett egység
Az akkumulátorban megadott számú egységet utasítja az adatküldés előkészítésére.

54. LISTEN küldése \$FFB1 — \$ED0C
Használt regiszterek: A, X
Bemeneti adat: A, a megcímezett egység
Az akkumulátorban megadott számú egységet utasítja az adatfogadás előkészítésére.
55. Másodlagos cím küldése LISTEN után, SECOND \$FF93 — \$EDB9
Használt regiszterek: A, X
Bemeneti adat: A, a másodlagos cím
56. Másodlagos cím küldése TALK után, TKSA \$FF96 — \$EDC7
Használt regiszterek: A, X
Bemeneti adat: A, a másodlagos cím
57. Egy byte továbbítása az IEC-buszon, IECOUT \$FFA8 — \$EDDD
Használt regiszter: A
Bemeneti adat: A, a továbbítandó byte
A rutin kiír az IEC buszra egy byte-ot.
58. UNTALK küldése \$FFAB — \$EDEF
Használt regiszterek: A, X
Bemeneti adatok: —
Az előzetesen adatküldésre kijelölt egységet az alapállapot felvételére utasítja.
59. UNLISTEN küldése, UNLSN \$FFAE — \$EDFE
Használt regiszterek: A, X
Bemeneti adatok: —
Az előzetesen adatfogadásra kijelölt egységet az alapállapot felvételére utasítja.
60. IECIN — egy byte az IEC-buszról, ACPTR \$FFA5 — \$EE13
Használt regiszter: A
Bemeneti adatok: —
Beolvas egy byte-ot az IEC-buszról. A byte az akkumulátorba kerül.
61. CLOCK HI \$EE85
Használt regiszter: A
Bemeneti adat: —
A soros busz CLK vonala magasra vált.

- 62. CLOCK LO** **\$EE8E**
Használt regiszter: A
Bemeneti adat: —
A soros busz CLK vonala alacsonyra vált.
- 63. DATA HI** **\$EE97**
Használt regiszter: A
Bemeneti adat: —
A soros busz DATA vonalát magasra állítja.
- 64. DATA LO** **\$EEA0**
Használt regiszter: A
Bemeneti adat: —
A soros busz DATA vonalát alacsonyra állítja.
- 65. 1 msec késleltetés** **\$EEB3**
Használt regiszterek: A, X
Bemeneti adat: —
- 66. GETIN, egy byte beolvasása** **\$FFE4 — \$F13E**
Használt regiszterek: A, Y
Bemeneti adat: —
Az előzetesen kijelölt egységről egy byte-ot olvas az akkumulátorba.
- 67. BSOUT, egy byte kiírása, CHROUT** **\$FFD2 — \$F1CA**
Használt regiszterek: A, Y (eredeti értékek a verembe mentve)
Bemeneti adatok: A, a kiírandó byte
Az előzetesen kijelölt egységhez továbbít egy byte-ot.
- 68. A bemeneti egység beállítása, CHKIN** **\$FFC6 — \$F20E**
Használt regiszterek: A, X
Bemeneti adat: X, a logikai file-szám
A megadott logikai file-hoz tartozó egység lesz a beolvasás forrása. Ezt az állapotot a CLRCHN rutin szünteti meg.
- 69. A kimeneti egység beállítása, CHKOUT** **\$FFC9 — \$F250**
Használt regiszterek: A, X
Bemeneti adat: X, a logikai file-szám

A megadott logikai file-hoz tartozó egység lesz a kiírás célja. Ezt az állapotot a CLRCHN rutin szünteti meg.

70. CLOSE, a file lezárása \$FFC3 — \$F291

Használt regiszterek: A, X, Y

Bemeneti adat: A, a file logikai száma

A rutin lezárja azt a file-t, amelynek száma az akkumulátorban van.

71. CLALL, minden file lezárása \$FFE7 — \$F32F

Használt regiszterek: A, X

Bemeneti adat: —

Minden megnyitott file-t lezár, és végrehajt egy CLRCHN-t.

72. CLRCHN, az aktív I/O csatorna törlése \$FFCC — \$F333

Használt regiszterek: A, X

Bemeneti adat: —

Visszaállítja az alaphelyzetet. Bemenet: billentyűzet, kimenet: képernyő.

73. OPEN, egy file megnyitása \$FFC0 — \$F34A

Használt regiszterek: A, X, Y

Bemeneti adatok: —

A SETFLS és a SETNAM rutinok által megadott adatok alapján megnyitja a file-t. A későbbiek folyamán a megnyitott file-t a CHKIN vagy a CHKOUT rutinok segítségével be-, ill. kimentí célokra használhatjuk.

74. LOAD, programfile betöltése \$FFD5 — \$F49E

Használt regiszterek: A, X, Y

Bemeneti adatok: A, jelző: 0 = LOAD, 1 = VERIFY

X: a kezdőcím alsó byte-ja

Y: a kezdőcím felső byte-ja

A LOAD rutin a tár feltöltésére szolgál. A rutin egyes paramétereit (egységszám, név stb.) a SETFLS rutinok előzetesen állítják be.

75. SAVE, programfile mentése \$FFD8 — \$F5DD

Használt regiszterek: A, X, Y

Bemeneti adatok: A: a program kezdőcíme (mutató a nulláslapon)

X: a végcím, alsó byte

Y: a végcím, felső byte

A SAVE rutin a tár meghatározott részét menti ki háttértárolóra. A rutin hívása előtt a

név és az egység beállítását a SETNAM és a SETFLS rutinokkal kell megoldanunk. A rutin a kezdőcímet indirekt módon kéri; a nulláslapon elhelyezett címmutató első byte-jának címét az akkumulátorban keresi.

- 76. Az idő növelése, UDTIM** **\$FFEA — \$F69B**
Használt regiszterek: A, X
Bemeneti adat: —
A nulláslapon elhelyezkedő számláló növelése (TI).
- 77. Az idő beolvasása, RDTIM** **\$FFDE — \$F6DD**
Használt regiszterek: A, X, Y
Bemeneti adatok: —
A nulláslapon levő számláló (\$A0 — A2) tartalmának beolvasása.
- 78. Az idő beállítása, SETTIM** **\$FFDB — \$F6E4**
Használt regiszterek: A, X, Y
Bemeneti adatok:
A: számláló, alsó byte (\$A2)
X: számláló, középső byte (\$A1)
Y: számláló, felső byte (\$A0)
A nulláslapon levő számláló tartalmának beállítása.
- 79. A STOP-billentyű lekérdezése, STOP** **\$FFE1 — \$F6ED**
Használt regiszterek: A, X
Bemeneti adatok: —
Ha le van nyomva, végrehajt egy CLRCHN-t.
- 80. A megszakítások inicializálása, IOINIT** **\$FF84 — \$FDA3**
Használt regiszterek: A, X
Bemeneti adatok: —
Helyreállítja a megszakításokhoz szükséges tárcímeket.
- 81. A file-név paramétereinek beállítása, SETNAM** **\$FFBD — \$FDF9**
Használt regiszterek: A, X, Y
Bemeneti adatok: A: a név hossza
X: a név címe, alsó byte
Y: a név címe, felső byte

82. A file-paraméterek beállítása, SETFLS \$FFBA — \$FE00
 Használt regiszterek: A, X, Y
 Bemeneti adatok: A: a logikai file-szám
 X: az egységszám
 Y: a másodlagos cím
83. A státus beolvasása, READST \$FFB7 — \$FE07
 Használt regiszter: A
 Bemeneti adatok: —
 A státusbyte beolvasása az akkumulátorba
84. A státus beállítása, SETMSG \$FF90 — \$FE18
 Használt regiszter: A
 Bemeneti adat: A, a státusbyte új értéke
85. Video-reset, CINT \$FF81 — \$FF5B
 Használt regiszterek: A, X
 Bemeneti adatok: —
 Inicializálja a VIC-et, és beállítja a megszakítást vezérlő regisztereket.

4.5. BASIC programozás során használható trükkök és megoldások

A BASIC programtár kezdetének beállítása:

POKE 43, LB
POKE 44, HB
POKE 256 * HB + LB, 0

LB: alsó byte, HB: felső byte

A BASIC programtár végének beállítása:

POKE 55, LB
POKE 56, HB

A NEW parancs hatástalanítása:

POKE 2050,1 : SYS 42291 : POKE 45, PEEK (34) : POKE 46, PEEK (35) : CLR

Ha a NEW parancs kiadása és a fenti parancssorozat végrehajtása között nem történt olyan művelet, amely használja a BASIC programtárat, a törölt BASIC program lemezre menthető.

Ha BASIC-programból indítunk BASIC programot, az új programnak a következő néhány utasítással kell kezdődnie:

POKE 45, PEEK (174) : POKE 46, PEEK (175) : CLR

mert a LOAD rutin sajátossága, hogy programmódban előlről kezdi meg az új program végrehajtását, amelynek tármutatói nem feltétlenül azonosak az előző program mutatóival.

Ha BASIC-program tölt gépi programot, a helyzet némileg más. A BASIC-program ekkor ugyanis általában ugyanaz marad. A program viszont a LOAD rutin lefutása után hasonlóképp előlről indul. Ilyenkor azt kell biztosítani, hogy ne keletkezhessen végtelen ciklus. A LOAD rutin önmagában nem törli a változók értékét, ezért ezt a végtelen ciklus elhárítására fel lehet használni. Például:

```
10 IF A=0 THEN A=1 : LOAD "P1",8,1
20 IF A=1 THEN A=2 : LOAD "P2",8,1
30 ...
```

A kurzor pozicionálására több változat is létezik:

```
POKE 211,Y : POKE 214,X : SYS 58732
POKE 211,Y : POKE 214,X : SYS 58640
POKE 781,X : POKE 782,Y : SYS 58636
```

Várakozás a kurzor KI fázisára:

```
WAIT 207,1 vagy 10 IF PEEK(207) THEN 10
```

A kurzor be- és kikapcsolása:

```
POKE 204,0      bekapcsolás
POKE 204,1      kikapcsolás
```

A billentyűzettel kapcsolatos trükkök:

```
POKE 650,128    minden billentyű ismétel
POKE 650,64     egyik billentyű sem ismétel
POKE 650,0      alaphelyzet
POKE 657,128    a SHIFT letiltva
POKE 657,0      a SHIFT engedélyezve
```

```
WAIT 197,4      várakozás a SPACE-re
WAIT 198,1      várakozás egy billentyűre
WAIT 198,255
```

WAIT 653,1	<i>várakozás a SHIFT-re</i>
WAIT 653,2	<i>várakozás a Commodore-billentyűre</i>
WAIT 653,4	<i>várakozás a CTRL-re</i>
 WAIT 203,64	 <i>várakozás addig, amíg egy billentyű le van nyomva</i>
 POKE 56322,224	 <i>a billentyűzet bénítása</i>
POKE 56322,0	
POKE 56323,255	
 POKE 649,0	 <i>a billentyűzet bénítása</i>
POKE 649,10	<i>a billentyűzet felszabadítása</i>
POKE 198,0	<i>a billentyűzetpuffer törlése</i>

A képernyővel kapcsolatos trükkök:

POKE 53280,X	<i>a képernyő kerete felveszi az X. színt</i>
POKE 53281,X	<i>a képernyő háttere felveszi az X. színt</i>
 POKE 53272,21	 <i>kisbetű/nagybetű üzemmód</i>
POKE 53272,23	<i>nagybetű/grafika üzemmód</i>
POKE 56325,0	<i>a kiírások jelentősen lelassulnak</i>

Karakterek többszínű üzemmódja:

POKE 53270, PEEK (53270) OR 16

Karakterek egyszínű üzemmódja:

POKE 53270, PEEK (53270) AND NOT 16

A képernyő kikapcsolása:

POKE 53265, PEEK (53265) AND NOT 16

A képernyő visszakapcsolása:

POKE 53265, PEEK (53265) OR 16

POKE 646,X	<i>a következő karakter színkódja: X</i>
POKE 648,X	<i>X*256: a képernyőmemória kezdőcíme</i>

Az X. képernyősor törlése:

POKE 781,X : SYS 59903

Az X. sorban Y karakter törlése

POKE 781,X : POKE781,Y : SYS 59905

Egyéb trükkök

POKE 1,54	<i>a BASIC ROM átkapcsolva RAM-ra</i>
POKE 1,53	<i>a KERNAL ROM átkapcsolva RAM-ra</i>
POKE 1,51	<i>a CHAREN PEEK-kel elérhető (a megszakítást előzetesen le kell tiltani)</i>
POKE 152,0	<i>lezárja az összes nyitott file-t</i>
POKE 152,11	<i>megakadályozza a file-nyitást</i>
POKE 193,1	<i>az inverz mód bekapcsolása</i>
POKE 212,1	<i>a PRINT kiírja a vezérlőjeleket</i>
WAIT 56576,128	<i>várakozás a DOS-RESET végére</i>
POKE 775,200	<i>a LIST letiltása</i>
POKE 775,191	
POKE 808,239	<i>a STOP letiltása</i>
POKE 809,255	
POKE 818,32	<i>a SAVE letiltása</i>
POKE 56334,0	<i>a megszakítások kikapcsolása</i>
POKE 56334,1	<i>a megszakítások engedélyezése</i>
POKE 2048,1	<i>RUN-ra SYNTAX ERROR</i>

Megadott címen kezdődő szöveg kiírása:

POKE 780,LB : POKE 782,HB : SYS 43806

16 bites egész szám kiírása:

POKE 781,LB : POKE 780,HB : SYS 48589
SYS 58592 *várakozás 8,5 sec-ig vagy a Commodore billentyűre*
PRINT MID\$(STR(I),2) *az I változó előjel nélküli kiírása*
X = RND(-TI) *valódi véletlenszám generálása*

A STOP/RESTORE letiltása:

POKE 788,52: POKE 792,193

A TI\$ gyors nullázása:

SYS 65499

Az 1. botkormány lekérdezése:

```
100 POKE 56320,127
110 A = PEEK (56321)
120 IF NOT A AND 1 THEN PRINT "FEL"
130 IF NOT A AND 2 THEN PRINT "LE"
140 IF NOT A AND 4 THEN PRINT "BAL"
150 IF NOT A AND 8 THEN PRINT "JOBB"
```

```
160 IF NOT A AND 16 THEN PRINT "TUZ"  
170 GOTO 110
```

A 2. botkormány lekérdezéséhez a csak a 100 és a 110 sorokat kell megváltoztatni:

```
100 POKE 56322,224  
110 A = PEEK (56320)
```

Függelék

A számítástechnika fejlődése

Világunkban, amely körülvesz bennünket, nem a számszerű mennyiségek a meghatározóak. Számszerű mennyiségen itt az egész számokkal egzakt, pontos módon mért mennyiséget értjük. Sokkal döntőbbek azok a mennyiségek, amelyek számokkal csak közelíthetők. Legtöbb mennyiségünk ilyen. A számszerűsítés emberi találmány.

Az elmúlt évszázadok legnagyobb tudósai közül sokan ismerték fel, hogy a természettudományok jelentős részének fejlődését erősen lassítja az a körülmény, hogy a matematikai levezetések csak kézzel számíthatják ki. A számítás nagyon sokáig tart, és nem biztos, hogy hibátlan. Ezért amikor a mechanika már volt annyira fejlett, hogy bonyolultabb automatákat lehetett építeni, többen megpróbálkoztak számológépek építésével, így pl. Leibniz, Pascal és mások. Ezek az automaták csak a számtani alpműveleteket voltak képesek elvégezni, de akkoriban már ezek is lehetővé tették közelítő eljárások segítségével bizonyos bonyolultabb számítások elvégzését.

Az első komolyabb számológépet az angol Babbage építette a XIX. században. Ez még mindig mechanikus szerkezet volt, de Babbage úgy alkotta meg, hogy a vele elvégezhető műveleteket már programozni lehetett. Babbage munkásságát sokan folytatták, olyanok is, mint Maxwell vagy Lord Kelvin. Mégis egy kevésbé ismert név emelkedik ki közülük, Herman Hollerith-é. Hollerith találta fel ugyanis a lyukkártyával működő számológépet még az 1800-as évek végén. A mai számítógépgyártó óriás, az IBM, alapítása után sokáig Hollerith-féle automatákat gyártott, és nagymértékben hozzájárult a mechanikus számolóautomaták elterjedéséhez. A mechanikus automatákat hamarosan felváltották az elektromechanikus elven működő számológépek. Az elektromosság térhódítása és az elektroncső feltalálása után néhány amerikai tudósnak sikerült megnyernie az Egyesült Államok hadseregének vezetőit ahhoz, hogy anyagilag támogassák egy tisztán elektromos elven működő számológép kifejlesztését. A második világháború utolsó évében el is készült ez a gép, amely már a modern értelemben vett számítógépnek tekinthető.

A számítástechnika fejlődésének következő lépcsője a tranzisztor alkalmazása volt. A tranzisztor az elektroncsövet váltotta fel. Jelentős előnye kis helyszükséglete és sokkal csekélyebb energiaigénye volt. A ma gyártott számítógépek is tranzisztorok sokaságát alkalmazzák.

A mai számítógépek elvi alapjait azonban mégsem fizikusok és gyakorlati szakemberek öntötték végső formájába. Ennek megalkotása egy magyar származású matematikus, Neumann János (1903 — 1957) nevéhez fűződik. Az ő elképzelése alapján működik a ma használatos számítógépek túlnyomó többsége.

A Neumann-féle számítógép lényegében csak két mennyiséget ismer, az 1-et (igaz), és a 0-at (hamis). Ebből a két értékből és a hozzájuk tartozó jelölésrendszerből felépíthető az egész világ modellje, mása. Ezt a két mennyiséget logikai állapotnak is szokás nevezni.

A modern számítógépekben a két logikai állapotnak feszültségértékeket feleltetnek meg. Az 1 logikai szintnek +5 V, a 0 szintnek 0 V felel meg. A logikai állapotokat digitális jelek néven is szokták emlegetni. A digitális kifejezés a latin digitus (ujj) szóból származik. Az ilyen módon történő számolás kizárja a tört számok közvetlen alkalmazását.

A számítógépek működésének logikai alapjai — logikai algebra

A két logikai állandó használata megköveteli a 2-es (bináris) számrendszer alkalmazását. A két logikai állandó által létrehozható logikai kapcsolatokat a logikai algebra tárgyalja.

A logikai algebra elméleti alapjait az ún. Boole-algebra alkotja, amely George Boole (1815 — 1864) angol matematikus munkája. Boole volt az első matematikus, aki a logikát precíz matematikai formába öntötte. Boole mutatta meg elsőként, hogy a logika igen egyszerű algebrai rendszerekre bontható. A logikai algebra rendszerének minden elemére igaz a következő állítás:

$$x = x^2$$

Az egyenletnek csak két megoldása van: 0 és 1. Ez az oka annak, hogy a mai számítógépekben olyan fontos szerepet játszik a bináris számrendszer: a gépek a logikai feladatokat bináris műveletekként végzik el.

A logikai algebra tehát csak két állandót ismer. Az 1 jelenti az összes szóba jöhető elemek halmazát, a 0 pedig az üres halmazt. A logikai algebra számára változók azok a mennyiségek, amelyek egyaránt felvehetik a 0 és az 1 állapotokat.

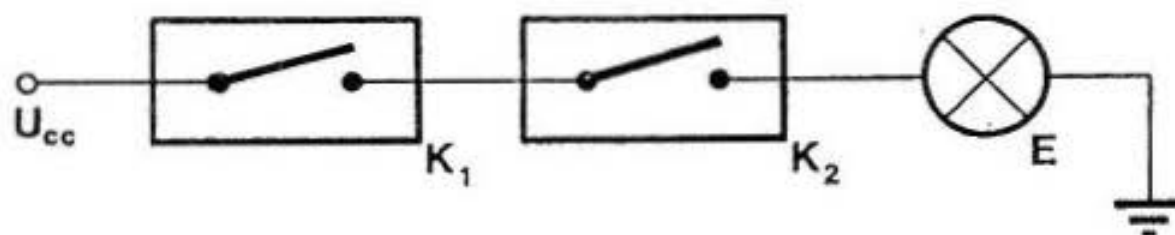
Minden logikai kapcsolat kifejezhető logikai egyenlet alakjában. A logikai algebra három alapvető logikai műveletet különböztet meg:

- az ÉS (AND) kapcsolatot, ez a *logikai szorzás*. Jelölése: * vagy \wedge .
- a VAGY (OR) kapcsolatot, ez a *logikai összeadás*. Jelölése: + vagy \vee .
- a NEM (NOT) kapcsolatot, ez a *negáció* vagy *tagadás*. Jelölése: felülvonás, például: \bar{A} .

A logikai ÉS kapcsolatot a 37. ábrán szemléltetjük. Az izzó előtt két kapcsoló helyezkedik el: az, amelyik magát az izzót kapcsolja, a másik voltaképpen a főkapcsoló.

Ahhoz, hogy az izzó világítson, be kell kapcsolni a K_2 kapcsolót ÉS be kell kapcsolni a K_1 kapcsolót is. A hangsúly az ÉS szón van. Ennek megfelelően egy ÉS kapcsolatot, amelynek két bemenete (K_1 és K_2), és egy kimenete (E) van, az ábrán megadott táblázat jellemez.

Eset	K_2	K_1	E
1	0	0	0
2	0	1	0
3	1	0	0
4	1	1	1



37. ábra. ÉS kapcsolat

Ha a bemenetek felveszik a logikai állandók valamelyikét, a kimenet a táblázat szerint alakul. A táblázat az ÉS logikai kapcsolat igazságtáblázata.

Az ÉS kapcsolat logikai egyenlete a következő:

$$E = K_1 \cdot K_2$$

Az ÉS kapcsolatot jelölhetjük a K_1 & K_2 jellel is.

ÉS kapcsolatot nemcsak két bemenet között teremthetünk. Lehetséges három vagy ennél több bemenet is. Két bemenet esetén négy lehetséges kombináció jöhet létre. Három bemenetnél 8, négyenél 16, ötnél 32 eset adódik és így tovább.

Hárombemenetű ÉS kapcsolat igazságtáblázata:

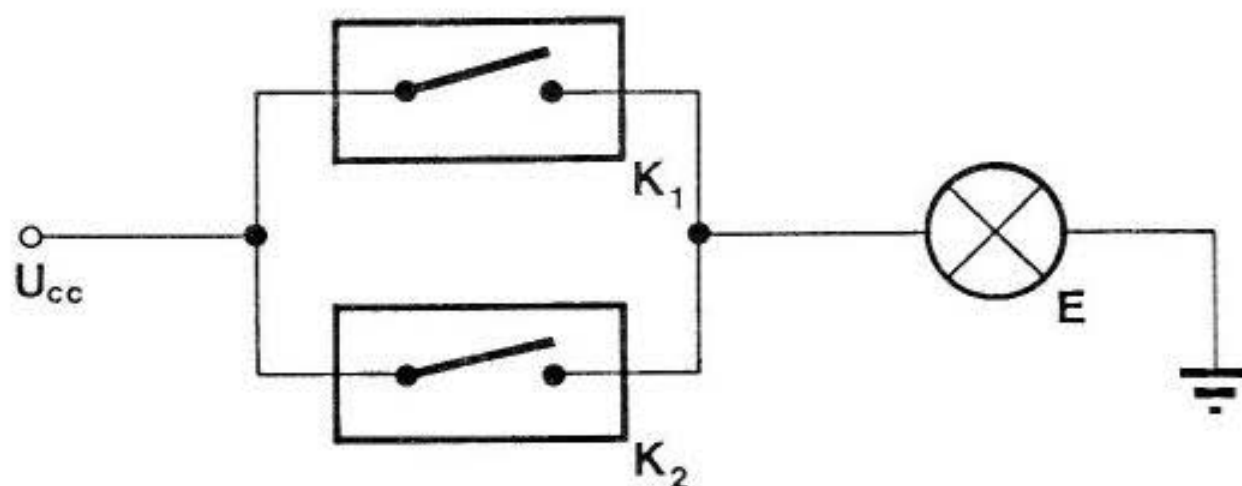
Eset	K_1	K_2	K_3	E
1	0	0	0	0
2	0	0	1	0
3	0	1	0	0
4	0	1	1	0
5	1	0	0	0
6	1	0	1	0
7	1	1	0	0
8	1	1	1	1

Az ÉS kapcsolat kimenete csak akkor 1, ha minden bemenete 1. Az általánosan elterjedt angol terminológia az ÉS kapcsolatot az AND szóval jelöli.

A logikai VAGY kapcsolat szemléltetéséhez vegyük elő ismét az izzólámpás példánkat. A 38. ábra szerinti kapcsolók bármelyikének bekapcsolása maga után vonja az izzó felizzását. Tehát ahhoz, hogy az izzó világítson, be kell kapcsolni a K_1 kapcsolót VAGY a K_2 kapcsolót.

A kapcsolatot MEGENGEDŐ VAGY-nak is szokták nevezni, mert megengedi azt is, hogy mindkét kapcsoló be legyen kapcsolva. A VAGY logikai kapcsolatot jellemző táblázatot is feltüntettük az ábrán.

Eset	K ₁	K ₂	E
1	0	0	0
2	0	1	1
3	1	0	1
4	1	1	1



38. ábra. VAGY kapcsolat

A VAGY kapcsolat logikai egyenlete:

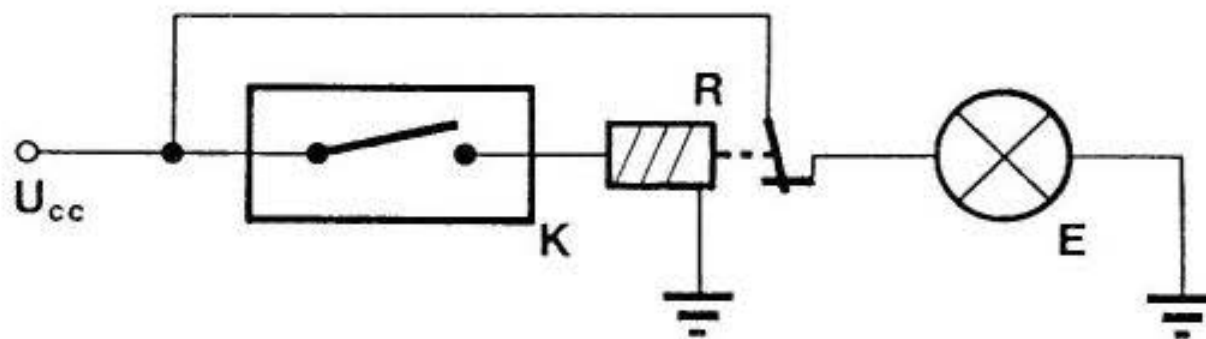
$$E = K_1 + K_2$$

Természetesen VAGY kapcsolatot is lehet több bemenet között létesíteni. A VAGY kapcsolatot az jellemzi, hogy kimenete mindig 1, ha valamelyik bemenete 1. A VAGY kapcsolatot az angol terminológia szerint az OR szóval is jelölhetjük.

A NEM logikai kapcsolathoz az izzólámpás példánkat némileg módosítanunk kell: az áramkörbe beiktattunk egy relét is. Ha a K kapcsolót bekapcsoljuk, az R relé meghúz, és megszakítja az E izzó áramkörét, ha kikapcsoljuk, az izzó ismét világít.

A NEM kapcsolat kimenetén mindig a bemenet ellentettje jelenik meg; ennek megfelelő igazságtáblázata is.

Eset	K	E
1	0	1
2	1	0



39. ábra. NEM kapcsolat

A kapcsolat logikai egyenlete:

$$E = \bar{K}$$

A NEM kapcsolat elterjedt neve a NOT vagy az inverter.

A C64-es logikai algebrája

A C64-es MOS 6510 processzora mindhárom logikai alpműveletet végre tudja hajtani. A processzor nyelve azonban csak az AND és az OR utasításokat tartalmazza; ezt a két műveletet a processzor az előzőekben ismertetett formában alkalmazza. A NOT művelet nincs a nyelvben, helyette a kizáró VAGY (EXOR — Exclusive OR) szerepel.

A kizáró VAGY művelet felépíthető a megismert három alpműveletből:

$$A \text{ EXOR } B = (A \text{ OR } B) \text{ AND NOT } (A \text{ AND } B)$$

A gépi nyelvnek azért nem szükséges tartalmaznia a NOT műveletet, mert ha a kizáró VAGY művelet egyik operandusa logikai 1, akkor az eredmény a másik operandus negáltja lesz. Azaz az EXOR művelet egyszerűen használható NOT műveletként is.

Az EXOR művelet igazságtáblázata:

Eset	K ₁	K ₂	E
1	0	0	0
2	0	1	1
3	1	0	1
4	1	1	0

Az EXOR művelet logikai egyenlete:

$$E = (K_1 + K_2) \cdot \overline{(K_1 \cdot K_2)} \text{ vagy}$$

$$E = (K_1 \cdot \overline{K_2}) + (\overline{K_1} \cdot K_2)$$

Az, hogy a processzor gépi nyelvében nem a NOT, hanem az EXOR művelet szerepel, nem véletlen. Ennek oka az, hogy kizáró VAGY műveletre gyakran van szükség programozás közben, s a processzor hardverébe épített EXOR sokkal gyorsabb, mintha diszkrét utasításokból építenénk fel.

A C64-esen alkalmazott számrendszerek

A C64-es természetesen csak egy számrendszert alkalmaz, a 2-est. A 10-es és 16-os számrendszert a programozó alkalmazza.

A 10-es alapú számrendszert magától értetődő természetességgel tanuljuk meg életünk során. Ebben nagy szerepe van biológiai adottságainknak: két kezünkön 10 ujjunk van. Ha egy számot, pl. a 456-ot ábrázoljuk 10-es számrendszerben, mindenki tudja, hogy ez a

$$4 * 100 + 5 * 10 + 6$$

műveletek eredményével egyenlő.

Így van ez a kettes és a tizenhatos számrendszerben is, csak fennáll az a különbség, hogy kevesebb, illetve több a felhasználható számjegyek száma. Vegyük előbb a kettes számrendszert.

Itt csak két érték lehetséges: a 0 és az 1. Mint a tízes számrendszerben, itt is helyiértékük szabja meg azt, hogy valójában mennyit is érnek. A kettes számrendszerben a helyiértékek szorzói 2 hatványai. Tehát a %11001000 nyolcbites bináris szám így is felírható:

$$1*128 + 1*64 + 0*32 + 0*16 + 1*8 + 0*4 + 0*2 + 0*1 = 200_{10}$$

vagy hatványalakban:

$$1*2^7 + 1*2^6 + 0*2^5 + 0*2^4 + 1*2^3 + 0*2^2 + 0*2^1 + 0*2^0 = 200_{10}$$

Jól látható, hogy a hatványkitevők megfelelnek a helyiértékek sorszámanak. Soha ne felejtsük el, hogy a hatványkitevők és ezáltal a helyiértékek sorszámozása jobbról balra tart, és 0-val kezdődik. Ha a tízes számrendszerre példaként mutatott #456 (456_{10}) számot akarjuk kettes számrendszerben ábrázolni, akkor nincs más dolgunk, mint megtoldani az előző példában szereplő bináris számot egy egyessel:

$$\%111001000 = \#456.$$

Tudjuk, hogy egy byte nyolc bites, azaz helyiértégeit 0-tól 7-ig számozhatjuk. A 8 biten ábrázolható legnagyobb szám a #255, amelyben minden helyiértéken 1-es áll ($\%11111111 = 255$). A példában felírt 456-os szám 9 bites. A C64-esben kétféle bináris szám gyakori: a 8 bites (byte), illetve a 16 bites (szó, cím). A 16 biten 2^{16} különböző érték ábrázolható, ha a nullát is értéknek tekintjük.

Hogy ne kelljen mindig hosszú számokat írni, amikor táruk kapacitását, programok hosszúságát vagy adathalmazok nagyságát kívánjuk jelölni, bevezették a prefixumok korlátozott használatát. Ennek megfelelően ha valamely mennyiség meghaladja a 2^{10} (#1024) számot, akkor kbyte-okban (kbitekben) ha a 2^{20} (#1048576) számot, akkor Mbyte-okban (Mbitekben), és ha a 2^{30} (#1073741824) számot, akkor Gbyte-okban (Gbitekben) számolunk. A 6510-es processzor címterülete 65536 byte, ez egyszerűbben 64 kbyte. A mai modern 32 bites

számítógépek elméletileg lehetséges címterülete 16 Tbyte (Terabyte), ha címbuszuk kétszer olyan széles, mint az adatbuszuk. ($16 \text{ Tbyte} = 2^{64} \text{ byte}$.)

Hogy a 16 Tbyte milyen hatalmas memória, erre példa egy kis kitérő.

Amikor az ókorban az indiai király udvarában egy ma már névtelen zseni feltalálta a sakkot, a király a remek játék fölött érzett örömeiben megkérdezte a feltalálót, mit kíván érte jutalmul. Ő szerényen csak annyi búzaszemet kért, amennyit a következő elv alapján kaphat: a 64 mezőre helyezzenek el búzaszemeket úgy, hogy az első mezőre egyet, a másodikra kettőt, a harmadikra négyet, a negyedikre nyolcat és így tovább. A kérés teljesíthetetlen volt. Valaki egyszer kiszámította, hogy ez a $2^{64} - 1$ (18 446 744 073 709 551 615) búzaszem egy Föld nagyságú sík gömb felületét 30 cm magasan borítaná be. Ennyi búza a földi vegetáció teljes ideje alatt összesen sem termett.

A 16-os számrendszer is ugyanolyan alapon nyugszik, mint a 2-es vagy a többi. A helyiértékek itt 16 hatványait jelentik. A különbség végeredményben annyi, hogy itt a tíz numerikus számjegyet (0 — 9) hat betű (A — F) egészíti ki. Ezek megfeleltetése:

A = 10, B = 11, C = 12, D = 13, E = 14, F = 15.

Nagyon könnyű bináris számrendszerből hexadecimálisba átszámolni, mert minden hexadecimális számjegy megfelel 4 bináris jegynek. Pl. a \$EE szám bináris rendszerben: %11101110 (#238) vagy a \$2A = %00101010 (#42). A 16 bites bináris számok hexadecimális rendszerben 4 jegyen ábrázolhatók.

A BCD-kód

A MOS 6510-es processzor még egy számrendszert alkalmaz, amely szintén bináris alapokon nyugszik. Ez a számrendszer az ún. BCD (Binary Coded Decimal) számrendszer. A BCD-kódolás jelentése binárisan kódolt decimális szám; a tízes számrendszer számjegyeit kódolja 4 bites bináris formában.

A BCD-kód formája megoldás kérdése. Itt a 6510-es processzor által alkalmazott BCD-rendszert ismertetjük, de meg kell említeni, hogy más BCD-rendszerek is léteznek (Gray-kód, Aiken-kód stb.). A kódok megfeleltetése:

Decimális	BCD	Decimális	BCD
0	0000	5	0101
1	0001	6	0110
2	0010	7	0111
3	0011	8	1000
4	0100	9	1001

A processzor BCD üzemmódban is képes számításokat végezni. Ilyenkor egy byte-ot két félbyte-ra bont, az alsó félbyte az egyes helyiértékű, a felső félbyte a tízes helyiértékű számokat jelenti. Egy byte-on tehát 99-ig ábrázolható számok. A számítási műveletek a tízes számrendszer szabályai szerint történnek. A BCD-kód használatára viszonylag ritkán van szükség, de néhány programozási feladat megköveteli használatát. Például a CIA TOD órája is BCD-módban programozható.

A processzort az állapotregiszter D bitje kapcsolja át BCD üzemmódra. Ha a bit magas, a processzor BCD üzemmódban számlál.

Negatív számok bináris rendszerben — a komplement

Bináris számrendszerben elméletileg csak pozitív számok ábrázolhatók; az előjel tárolására nincs külön információhordozó. Azonban ez csak megállapodás kérdése. Megállapodás alapján a bináris szám egy bitjét kijelölhetjük az előjel meghatározására. Ez a bit kézenfekvő módon mindig a legfelső bit. Ha a bit magas, a szám negatív. A negatív szám értékét azonban nem kapjuk úgy meg, hogy az előjelbit mögötti számértéket kiegészítjük egy negatív előjellel.

A számérték valódi megfelelőjét az adott szám logikai komplemente adja. Napjaink számítógépei negatív számok ábrázolására a számok logikai komplementeit használják. Bináris rendszerben ezeket a számokat kettes komplementnek is nevezhetjük.

A logikai komplement legfelső bitje (7., 15. stb.) tölti be az előjel szerepét. Ha ez magas, a szám negatív. A negatív szám abszolút értékét a következő módon számíthatjuk ki:

- a legfelső bitet alacsonyra állítjuk;
- az összes többi bitet ellenkezőjére fordítjuk;
- az előző műveletek eredményéhez hozzáadunk egyet.

Az így számított érték pontosan a keresett negatív szám abszolút értéke. Nézzünk egy példát!

A negatív szám:	%10101110	#174 (−82)
1. művelet:	%00101110	#46
2. művelet:	%01010001	#81
3. művelet:	%01010010	#82

Azaz a 174-nek 82 a logikai komplemente (vagy a 82-nek a 174). Ha pozitív szám komplementjét akarjuk kiszámítani, a műveletsor hasonló:

- a legfelső bitet magasra állítjuk;

- a többi bitet invertáljuk;
- a művelet eredményéhez hozzáadunk egyet.

A pozitív szám: %01010010 #82

1. művelet: %11010010 #210

2. művelet: %10101101 #173

3. művelet: %10101110 #174 (−82)

Ha jobban megnézzük, a három művelet kettőre csökkenthető. Az első és a második művelet egybevonható. Ennek megfelelően a logikai komplement előállítása mind pozitív számra, mind negatív számra a következő:

1. Minden bitet az ellenkezőjére fordítunk (invertálunk).
2. Az előző művelet eredményéhez hozzáadunk egyet.

A MOS 6510 utasításkészlete

00 - BRK	: 33 - RAN IY	: 66 - ROR N	: 99 - STA AY	: CC - CPY A
01 - ORA IX	: 34 - NO2 NX	: 67 - RAD N	: 9A - TXS	: CD - CMP A
02 - ABS	: 35 - AND NX	: 68 - PLA	: 9B - AXS AY	: CE - DEC A
03 - ASO IX	: 36 - ROL NX	: 69 - ADC #	: 9C - YXA AX	: CF - DEM A
04 - NO2 N	: 37 - RAN NX	: 6A - ROR BE	: 9D - STA AX	: D0 - BNE R
05 - ORA N	: 38 - SEC	: 6B - DAR #	: 9E - XYA AY	: D1 - CMP IY
06 - ASL N	: 39 - AND AY	: 6C - JMP I	: 9F - AYY AY	: D2 - ABS
07 - ASO N	: 3A - NO1	: 6D - ADC A	: A0 - LDY #	: D3 - DEM IY
08 - PHP	: 3B - RAN AY	: 6E - ROR A	: A1 - LDA IX	: D4 - NO2 NX
09 - ORA #	: 3C - NO3 AX	: 6F - RAD A	: A2 - LDX #	: D5 - CMP NX
0A - ASL BE	: 3D - AND AX	: 70 - BVS R	: A3 - LDT IX	: D6 - DEC NX
0B - ANN #	: 3E - ROL AX	: 71 - ADC IY	: A4 - LDY N	: D7 - DEM NX
0C - NO3 A	: 3F - RAN AX	: 72 - ABS	: A5 - LDA N	: D8 - CLD
0D - ORA A	: 40 - RTI	: 73 - RAD IY	: A6 - LDX N	: D9 - CMP AY
0E - ASL A	: 41 - EOR IX	: 74 - NO2 NX	: A7 - LDT N	: DA - NO1
0F - ASO A	: 42 - ABS	: 75 - ADC NX	: A8 - TAY	: DB - DEM AY
10 - BPL R	: 43 - LSE IX	: 76 - ROR NX	: A9 - LDA #	: DC - NO3 AX
11 - ORA IY	: 44 - NO2 N	: 77 - RAD NX	: AA - TAX	: DD - CMP AX
12 - ABS	: 45 - EOR N	: 78 - SEI	: AB - ANX #	: DE - DEC AX
13 - ASO IY	: 46 - LSR N	: 79 - ADC AY	: AC - LDY A	: DF - DEM AX
14 - NO2 NX	: 47 - LSE N	: 7A - NO1	: AD - LDA A	: E0 - CPX #
15 - ORA NX	: 48 - PHA	: 7B - RAD AY	: AE - LDX A	: E1 - SBC IX
16 - ASL NX	: 49 - EOR #	: 7C - NO3 AX	: AF - LDT A	: E2 - NO2 IX
17 - ASO NX	: 4A - LSR BE	: 7D - ADC AX	: B0 - BCS R	: E3 - INB IX
18 - CLC	: 4B - ANL #	: 7E - ROR AX	: B1 - LDA IY	: E4 - CPX N
19 - ORA AY	: 4C - JMP A	: 7F - RAD AX	: B2 - ABS	: E5 - SBC N
1A - NO1	: 4D - EOR A	: 80 - NO2 IX	: B3 - LDT IY	: E6 - INC N
1B - ASO AY	: 4E - LSR A	: 81 - STA IX	: B4 - LDY NX	: E7 - INB N
1C - NO3 AX	: 4F - LSE A	: 82 - NO2 IX	: B5 - LDA NX	: E8 - INX
1D - ORA AX	: 50 - BVC R	: 83 - AXX IX	: B6 - LDX NY	: E9 - SBC #
1E - ASL AX	: 51 - EOR IY	: 84 - STY N	: B7 - LDT NY	: EA - NOP
1F - ASO AX	: 52 - ABS	: 85 - STA N	: B8 - CLV	: EB - SBC #
20 - JSR A	: 53 - LSE IY	: 86 - STX N	: B9 - LDA AY	: EC - CPX A
21 - AND IX	: 54 - NO2 NX	: 87 - AXR N	: BA - TSX	: ED - SBC A
22 - ABS	: 55 - EOR NX	: 88 - DEY	: BB - TSA AY	: EE - INC A
23 - RAN IX	: 56 - LSR NX	: 89 - NO2 #	: BC - LDY AX	: EF - INB A
24 - BIT N	: 57 - LSE NX	: 8A - TXA	: BD - LDA AX	: F0 - BEQ R
25 - AND N	: 58 - CLI	: 8B - TAN #	: BE - LDX AY	: F1 - SBC IY
26 - ROL N	: 59 - EOR AY	: 8C - STY A	: BF - LDT AY	: F2 - ABS
27 - RAN N	: 5A - NO1	: 8D - STA A	: C0 - CPY #	: F3 - INB IY
28 - PLP	: 5B - LSE AY	: 8E - STX A	: C1 - CMP IX	: F4 - NO2 NX
29 - AND #	: 5C - NO3 AX	: 8F - AAX A	: C2 - NO2 IX	: F5 - SBC NX
2A - ROL BE	: 5D - EOR AX	: 90 - BCC R	: C3 - DEM IX	: F6 - INC NX
2B - ANN #	: 5E - LSR AX	: 91 - STA IY	: C4 - CPY N	: F7 - INB NX
2C - BIT A	: 5F - LSE AX	: 92 - ABS	: C5 - CMP N	: F8 - SED
2D - AND A	: 60 - RTS	: 93 - AXI IY	: C6 - DEC N	: F9 - SBC AY
2E - ROL A	: 61 - ADC IX	: 94 - STY NX	: C7 - DEM N	: FA - NO1
2F - RAN A	: 62 - ABS	: 95 - STA NX	: C8 - INY	: FB - INB AY
30 - BMI R	: 63 - RAD IX	: 96 - STX NY	: C9 - CMP #	: FC - NO3 AX
31 - AND IY	: 64 - NO2 N	: 97 - AXY NY	: CA - DEX	: FD - SBC AX
32 - ABS	: 65 - ADC N	: 98 - TYA	: CB - XAS #	: FE - INC AX
-	:	-	:	: FF - INB AX

N : nulláslap
IX : (indirekt),X
AX : abszolút,X

A : abszolút
IY : (indirekt),Y
AY : abszolút,Y

NX : nulláslap,X
: közvetlen
I : indirekt

NY : nulláslap,Y
BE : akkumulátor
R : relativ

Standard utasítások (utasításcsoportonként rendezve)

: A3 - LDA #\$FF	: A2 - LDX #\$FF	: A0 - LDY #\$FF	: AA - TAX
: A5 - LDA \$FF	: A6 - LDX \$FF	: A4 - LDY \$FF	: AB - TAY
: B5 - LDA \$FF,X	: B6 - LDX \$FF,Y	: B4 - LDY \$FF,X	: BA - TXA
: AD - LDA \$FFFF	: AE - LDX \$FFFF	: AC - LDY \$FFFF	: 98 - TYA
: BD - LDA \$FFFF,X	: BE - LDX \$FFFF,Y	: BC - LDY \$FFFF,X	: 9A - TXS
: B9 - LDA \$FFFF,Y	:	:	: BA - TSX
: A1 - LDA (\$FF,X)	: 86 - STX \$FF	: 84 - STY \$FF	: 38 - SEC
: B1 - LDA (\$FF),Y	: 96 - STX \$FF,Y	: 94 - STY \$FF,X	: F8 - SED
:	: 8E - STX \$FFFF	: 8C - STY \$FFFF	: 18 - CLC
: 85 - STA \$FF	:	:	: D8 - CLD
: 95 - STA \$FF,X	: E6 - INC \$FF	: C6 - DEC \$FF	: B8 - CLV
: 8D - STA \$FFFF	: F6 - INC \$FF,X	: D6 - DEC \$FF,X	: 58 - CLI
: 9D - STA \$FFFF,X	: EE - INC \$FFFF	: CE - DEC \$FFFF	: 78 - SEI
: 39 - STA \$FFFF,Y	: FE - INC \$FFFF,X	: DE - DEC \$FFFF,X	: CA - DEX
: 81 - STA (\$FF,X)	:	:	: 88 - DEY
: 91 - STA (\$FF),Y	: 48 - PHA	: 2A - ROL A	: E8 - INX
:	: 68 - PLA	: 26 - ROL \$FF	: C8 - INY
: 4C - JMP \$FFFF	: 08 - PHP	: 36 - ROL \$FF,X	:
: 6C - JMP (\$FFFF)	: 28 - PLP	: 2E - ROL \$FFFF	: 6A - ROR A
: 20 - JSR \$FFFF	:	: 3E - ROL \$FFFF,X	: 66 - ROR \$FF
: 60 - RTS	: 0A - ASL A	:	: 76 - ROR \$FF,X
: 40 - RTI	: 06 - ASL \$FF	: 4A - LSR A	: 6E - ROR \$FFFF
: 00 - BRK	: 16 - ASL \$FF,X	: 46 - LSR \$FF	: 7E - ROR \$FFFF,X
: 90 - BCC \$FFFF	: 1E - ASL \$FFFF,X	: 56 - LSR \$FF,X	:
: B0 - BCS \$FFFF	: 0E - ASL \$FFFF	: 4E - LSR \$FFFF	: 49 - EOR #\$FF
: F0 - BEQ \$FFFF	:	: 5E - LSR \$FFFF,X	: 45 - EOR \$FF
: D0 - BNE \$FFFF	: 29 - AND #\$FF	:	: 55 - EOR \$FF,X
: 30 - BMI \$FFFF	: 25 - AND \$FF	: 09 - ORA #\$FF	: 4D - EOR \$FFFF
: 10 - BPL \$FFFF	: 35 - AND \$FF,X	: 05 - ORA \$FF	: 5D - EOR \$FFFF,X
: 50 - BVC \$FFFF	: 2D - AND \$FFFF	: 15 - ORA \$FF,X	: 59 - EOR \$FFFF,Y
: 70 - BVS \$FFFF	: 3D - AND \$FFFF,X	: 0D - ORA \$FFFF	: 41 - EOR (\$FF,X)
:	: 39 - AND \$FFFF,Y	: 1D - ORA \$FFFF,X	: 51 - EOR (\$FF),Y
: 24 - BIT \$FF	: 21 - AND (\$FF,X)	: 19 - ORA \$FFFF,Y	:
: 2C - BIT \$FFFF	: 31 - AND (\$FF),Y	: 01 - ORA (\$FF,X)	: C9 - CMP #\$FF
:	:	: 11 - ORA (\$FF),Y	: C5 - CMP \$FF
: 69 - ADC #\$FF	: E9 - SBC #\$FF	:	: D5 - CMP \$FF,X
: 65 - ADC \$FF	: E5 - SBC \$FF	: E0 - CPX #\$FF	: CD - CMP \$FFFF
: 75 - ADC \$FF,X	: F5 - SBC \$FF,X	: E4 - CPX \$FF	: DD - CMP \$FFFF,X
: 6D - ADC \$FFFF	: ED - SBC \$FFFF	: EC - CPX \$FFFF	: D9 - CMP \$FFFF,Y
: 7D - ADC \$FFFF,X	: FD - SBC \$FFFF,X	:	: C1 - CMP (\$FF,X)
: 79 - ADC \$FFFF,Y	: F9 - SBC \$FFFF,Y	: C0 - CPY #\$FF	: D1 - CMP (\$FF),Y
: 61 - ADC (\$FF,X)	: E1 - SBC (\$FF,X)	: C4 - CPY \$FF	:
: 71 - ADC (\$FF),Y	: F1 - SBC (\$FF),Y	: CC - CPY \$FFFF	: EA - NOP

Tiltott utasítások (utasításcsoportonként rendezve)

: 07 - ASO \$FF	: 1A - NO1	: 02 - ABS	: 47 - LSE \$FF
: 17 - ASO \$FF,Y	: 3A - NO1	: 12 - ABS	: 57 - LSE \$FF,X
: 0F - ASO \$FFFF	: 5A - NO1	: 22 - ABS	: 4F - LSE \$FFFF
: 1F - ASO \$FFFF,X	: 7A - NO1	: 32 - ABS	: 5F - LSE \$FFFF,X
: 1B - ASO \$FFFF,Y	: DA - NO1	: 42 - ABS	: 5B - LSE \$FFFF,Y
: 03 - ASO (\$FF,X)	: FA - NO1	: 52 - ABS	: 43 - LSE (\$FF,X)
: 13 - ASO (\$FF),Y		: 62 - ABS	: 53 - LSE (\$FF),Y
	: 89 - NO2 #\$FF	: 72 - ABS	
: C7 - DEM \$FF	: 04 - NO2 \$FF	: 92 - ABS	: 0B - ANN #\$FF
: D7 - DEM \$FF,Y	: 44 - NO2 \$FF	: B2 - ABS	: 2B - ANN #\$FF
: CF - DEM \$FFFF	: 64 - NO2 \$FF	: D2 - ABS	
: DF - DEM \$FFFF,X	: 14 - NO2 \$FF,X	: F2 - ABS	: EB - SBC #\$FF
: DB - DEM \$FFFF,Y	: 34 - NO2 \$FF,X		
: C3 - DEM (\$FF,X)	: 54 - NO2 \$FF,X	: 67 - RAD \$FF	: 8F - AAX \$FFFF
: D3 - DEM (\$FF),Y	: 74 - NO2 \$FF,X	: 77 - RAD \$FF,X	: 4B - ANL #\$FF
	: D4 - NO2 \$FF,X	: 6F - RAD \$FFFF	: AB - ANX #\$FF
: E7 - INB \$FF	: F4 - NO2 \$FF,X	: 7F - RAD \$FFFF,X	: 93 - AXI (\$FF),Y
: F7 - INB \$FF,X	: 80 - NO2 (\$FF,X)	: 7B - RAD \$FFFF,Y	: 87 - AXR \$FF
: EF - INB \$FFFF	: 82 - NO2 (\$FF,X)	: 63 - RAD (\$FF,X)	: 9B - AXS \$FFFF,Y
: FF - INB \$FFFF,X	: C2 - NO2 (\$FF,X)	: 73 - RAD (\$FF),Y	: 83 - AXX (\$FF,X)
: FB - INB \$FFFF,Y	: E2 - NO2 (\$FF,X)		: 97 - AXY \$FF,Y
: E3 - INB (\$FF,X)		: 27 - RAN \$FF	: 9F - AYY \$FFFF,Y
: F3 - INB (\$FF),Y	: 0C - NO3 \$FFFF	: 37 - RAN \$FF,X	: 6B - DAR #\$FF
	: 1C - NO3 \$FFFF,X	: 2F - RAN \$FFFF	: 8B - TAN #\$FF
: A7 - LDT \$FF	: 3C - NO3 \$FFFF,X	: 3F - RAN \$FFFF,X	: BB - TSA \$FFFF,Y
: B7 - LDT \$FF,Y	: 5C - NO3 \$FFFF,X	: 3B - RAN \$FFFF,Y	: CB - XAS #\$FF
: AF - LDT \$FFFF	: 7C - NO3 \$FFFF,X	: 23 - RAN (\$FF,X)	: 9E - XYA \$FFFF,Y
: BF - LDT \$FFFF,Y	: DC - NO3 \$FFFF,X	: 33 - RAN (\$FF),Y	: 9C - YXA \$FFFF,Y
: A3 - LDT (\$FF,X)	: FC - NO3 \$FFFF,X		
: B3 - LDT (\$FF),Y			

A MOS 6510 utasításai és hatásuk a jelzőkre

Mnemonic	:	N	:	V	:	-	:	B	:	O	:	I	:	Z	:	C	:	Mnemonic
AAX	:	-	:	-	:	.	:	-	:	-	:	-	:	-	:	-	:	AAX
ADC	:	x	:	x	:	.	:	-	:	-	:	-	:	x	:	x	:	ADC
AND	:	x	:	-	:	.	:	-	:	-	:	-	:	x	:	-	:	AND
ANL	:	x	:	-	:	.	:	-	:	-	:	-	:	x	:	x	:	ANL
ANN	:	x	:	-	:	.	:	-	:	-	:	-	:	x	:	x	:	ANN
ANX	:	x	:	-	:	.	:	-	:	-	:	-	:	x	:	-	:	ANX
ASL	:	x	:	-	:	.	:	-	:	-	:	-	:	x	:	x	:	ASL
ASO	:	x	:	-	:	.	:	-	:	-	:	-	:	x	:	x	:	ASO
AXI	:	-	:	-	:	.	:	-	:	-	:	-	:	-	:	-	:	AXI
AXR	:	-	:	-	:	.	:	-	:	-	:	-	:	-	:	-	:	AXR
AXS	:	-	:	-	:	.	:	-	:	-	:	-	:	-	:	-	:	AXS
AXX	:	-	:	-	:	.	:	-	:	-	:	-	:	-	:	-	:	AXX
AXY	:	-	:	-	:	.	:	-	:	-	:	-	:	-	:	-	:	AXY
AYY	:	-	:	-	:	.	:	-	:	-	:	-	:	-	:	-	:	AYY
BCC	:	-	:	-	:	.	:	-	:	-	:	-	:	-	:	-	:	BCC
BCS	:	-	:	-	:	.	:	-	:	-	:	-	:	-	:	-	:	BCS
BEQ	:	-	:	-	:	.	:	-	:	-	:	-	:	-	:	-	:	BEQ
BIT	:	M	:	M	:	.	:	-	:	-	:	-	:	x	:	-	:	BIT
BMI	:	-	:	-	:	.	:	-	:	-	:	-	:	-	:	-	:	BMI
BNE	:	-	:	-	:	.	:	-	:	-	:	-	:	-	:	-	:	BNE
BPL	:	-	:	-	:	.	:	-	:	-	:	-	:	-	:	-	:	BPL
BRK	:	-	:	-	:	.	:	1	:	-	:	1	:	-	:	-	:	BRK
BVC	:	-	:	-	:	.	:	-	:	-	:	-	:	-	:	-	:	BVC
BVS	:	-	:	-	:	.	:	-	:	-	:	-	:	-	:	-	:	BVS
CLC	:	-	:	-	:	.	:	-	:	-	:	-	:	-	:	0	:	CLC
CLD	:	-	:	-	:	.	:	-	:	0	:	-	:	-	:	-	:	CLD
CLI	:	-	:	-	:	.	:	-	:	-	:	0	:	-	:	-	:	CLI
CLV	:	-	:	0	:	.	:	-	:	-	:	-	:	-	:	-	:	CLV
CMP	:	x	:	-	:	.	:	-	:	-	:	-	:	x	:	x	:	CMP
CPX	:	x	:	-	:	.	:	-	:	-	:	-	:	x	:	x	:	CPX
CPY	:	x	:	-	:	.	:	-	:	-	:	-	:	x	:	x	:	CPY
DAR	:	x	:	x	:	.	:	-	:	-	:	-	:	x	:	-	:	DAR
DEC	:	x	:	-	:	.	:	-	:	-	:	-	:	x	:	-	:	DEC
DEM	:	x	:	-	:	.	:	-	:	-	:	-	:	x	:	x	:	DEM
DEX	:	x	:	-	:	.	:	-	:	-	:	-	:	x	:	-	:	DEX
DEY	:	x	:	-	:	.	:	-	:	-	:	-	:	x	:	-	:	DEY
EOR	:	x	:	-	:	.	:	-	:	-	:	-	:	x	:	-	:	EOR
INB	:	x	:	x	:	.	:	-	:	-	:	-	:	x	:	x	:	INB
INC	:	x	:	-	:	.	:	-	:	-	:	-	:	x	:	-	:	INC
INX	:	x	:	-	:	.	:	-	:	-	:	-	:	x	:	-	:	INX
INY	:	x	:	-	:	.	:	-	:	-	:	-	:	x	:	-	:	INY

x = változik ; 0 és 1 = a jelző beállítódik ; M = módosul

A MOS 6510 utasításai és hatásuk a jelzőkre

Mnemonic	:	N	:	V	:	-	:	B	:	D	:	I	:	Z	:	C	:	Mnemonic
JMP	:	-	:	-	:	.	:	-	:	-	:	-	:	-	:	-	JMP
JSR	:	-	:	-	:	.	:	-	:	-	:	-	:	-	:	-	JSR
LDA	:	x	:	-	:	.	:	-	:	-	:	-	:	x	:	-	LDA
LDT	:	x	:	-	:	.	:	-	:	-	:	-	:	x	:	-	LDT
LDX	:	x	:	-	:	.	:	-	:	-	:	-	:	x	:	-	LDX
LDY	:	x	:	-	:	.	:	-	:	-	:	-	:	x	:	-	LDY
LSE	:	x	:	-	:	.	:	-	:	-	:	-	:	x	:	x	LSE
LSR	:	0	:	-	:	.	:	-	:	-	:	-	:	x	:	x	LSR
NOP	:	-	:	-	:	.	:	-	:	-	:	-	:	-	:	-	NOP
ORA	:	x	:	-	:	.	:	-	:	-	:	-	:	x	:	-	ORA
PHA	:	-	:	-	:	.	:	-	:	-	:	-	:	-	:	-	PHA
PHP	:	-	:	-	:	.	:	-	:	-	:	-	:	-	:	-	PHP
PLA	:	x	:	-	:	.	:	-	:	-	:	-	:	x	:	-	PLA
PLP	:	x	:	x	:	.	:	x	:	x	:	x	:	x	:	x	PLP
RAD	:	x	:	x	:	.	:	-	:	-	:	-	:	x	:	x	RAD
RAN	:	x	:	-	:	.	:	-	:	-	:	-	:	x	:	x	RAN
ROL	:	x	:	-	:	.	:	-	:	-	:	-	:	x	:	x	ROL
ROR	:	x	:	-	:	.	:	-	:	-	:	-	:	x	:	x	ROR
RTI	:	x	:	x	:	.	:	x	:	x	:	x	:	x	:	x	RTI
RTS	:	-	:	-	:	.	:	-	:	-	:	-	:	-	:	-	RTS
SBC	:	x	:	x	:	.	:	-	:	-	:	-	:	x	:	x	SBC
SEC	:	-	:	-	:	.	:	-	:	-	:	-	:	-	:	1	SEC
SED	:	-	:	-	:	.	:	-	:	1	:	-	:	-	:	-	SED
SEI	:	-	:	-	:	.	:	-	:	-	:	1	:	-	:	-	SEI
STA	:	-	:	-	:	.	:	-	:	-	:	-	:	-	:	-	STA
STX	:	-	:	-	:	.	:	-	:	-	:	-	:	-	:	-	STX
STY	:	-	:	-	:	.	:	-	:	-	:	-	:	-	:	-	STY
TAN	:	x	:	-	:	.	:	-	:	-	:	-	:	x	:	-	TAN
TAX	:	x	:	-	:	.	:	-	:	-	:	-	:	x	:	-	TAX
TAY	:	x	:	-	:	.	:	-	:	-	:	-	:	x	:	-	TAY
TSA	:	-	:	-	:	.	:	-	:	-	:	-	:	-	:	-	TSA
TSX	:	x	:	-	:	.	:	-	:	-	:	-	:	x	:	-	TSX
TXA	:	x	:	-	:	.	:	-	:	-	:	-	:	x	:	-	TXA
TXS	:	-	:	-	:	.	:	-	:	-	:	-	:	-	:	-	TXS
TYA	:	x	:	-	:	.	:	-	:	-	:	-	:	x	:	-	TYA
XAS	:	x	:	-	:	.	:	-	:	-	:	-	:	x	:	x	XAS
XYA	:	-	:	-	:	.	:	-	:	-	:	-	:	-	:	-	XYA
YXA	:	-	:	-	:	.	:	-	:	-	:	-	:	-	:	-	YXA

x = változik ; 0 és 1 = a jelző beállítódik ; M = módosul

Kódátszámítási táblázat

#	\$	%	Memo.	Token	ASCII	Screen	DeKóder	\$	#
000	00	00000000	BRK			@ : @	Inst/Del	00	000
001	01	00000001	ORA IX			A : a	RETURN	01	001
002	02	00000010	ABS			B : b	CRSR right	02	002
003	03	00000011	ASO IX			C : c	F7	03	003
004	04	00000100	NO2 N			D : d	F1	04	004
005	05	00000101	ORA N		White	E : e	F3	05	005
006	06	00000110	ASL N			F : f	F5	06	006
007	07	00000111	ASO N			G : g	CRSR down	07	007
008	08	00001000	PHP			H : h	3	08	008
009	09	00001001	ORA #			I : i	W	09	009
010	0A	00001010	ASL BE			J : j	A	0A	010
011	0B	00001011	ANN #			K : k	4	0B	011
012	0C	00001100	NO3 A			L : l	Z	0C	012
013	0D	00001101	ORA A		Return	M : m	S	0D	013
014	0E	00001110	ASL A			N : n	E	0E	014
015	0F	00001111	ASO A			O : o	SHIFT left	0F	015
016	10	00010000	BPL R			P : p	5	10	016
017	11	00010001	ORA IY		CRSR dn	Q : q	R	11	017
018	12	00010010	ABS		RVS on	R : r	D	12	018
019	13	00010011	ASO IY		Home	S : s	6	13	019
020	14	00010100	NO2 NX		Del	T : t	C	14	020
021	15	00010101	ORA NX			U : u	F	15	021
022	16	00010110	ASL NX			V : v	T	16	022
023	17	00010111	ASO NX			W : w	X	17	023
024	18	00011000	CLC			X : x	7	18	024
025	19	00011001	ORA AY			Y : y	Y	19	025
026	1A	00011010	NO1			Z : z	G	1A	026
027	1B	00011011	ASO AY			[: [8	1B	027
028	1C	00011100	NO3 AX		Red	£ : £	B	1C	028
029	1D	00011101	ORA AX		CRSR rt] :]	H	1D	029
030	1E	00011110	ASL AX		Green	↑ : ↑	U	1E	030
031	1F	00011111	ASO AX		Blue	+ : +	V	1F	031
032	20	00100000	JSR A		Space	Space	9	20	032
033	21	00100001	AND IX		! : !	! : !	I	21	033
034	22	00100010	ABS		" : "	" : "	J	22	034
035	23	00100011	RAN IX		# : #	# : #	0	23	035
036	24	00100100	BIT N		\$: \$	\$: \$	M	24	036
037	25	00100101	AND N		% : %	% : %	K	25	037
038	26	00100110	ROL N		& : &	& : &	O	26	038
039	27	00100111	RAN N		' : '	' : '	N	27	039

Kódatszámítási táblázat

#	\$	%	Mnemo.	Token	ASCII	Screen	Dekóder	\$	#
040	28	00101000	PLP		(: ((: (+	28	040
041	29	00101001	AND #) :)) :)	P	29	041
042	2A	00101010	ROL BE		* : *	* : *	L	2A	042
043	2B	00101011	ANN #		+ : +	+ : +	-	2B	043
044	2C	00101100	BIT A		, : ,	, : ,	.	2C	044
045	2D	00101101	AND A		- : -	- : -	:	2D	045
046	2E	00101110	ROL A		. : .	. : .	@	2E	046
047	2F	00101111	RAN A		/ : /	/ : /	,	2F	047
048	30	00110000	BMI R		0 : 0	0 : 0	£	30	048
049	31	00110001	AND IY		1 : 1	1 : 1	*	31	049
050	32	00110010	ABS		2 : 2	2 : 2	:	32	050
051	33	00110011	RAN IY		3 : 3	3 : 3	Clr/Home	33	051
052	34	00110100	NO2 NX		4 : 4	4 : 4	SHIFT right	34	052
053	35	00110101	AND NX		5 : 5	5 : 5	=	35	053
054	36	00110110	ROL NX		6 : 6	6 : 6	†	36	054
055	37	00110111	RAN NX		7 : 7	7 : 7	/	37	055
056	38	00111000	SEC		8 : 8	8 : 8	1	38	056
057	39	00111001	AND AY		9 : 9	9 : 9	+	39	057
058	3A	00111010	NO1		:	:	CTRL	3A	058
059	3B	00111011	RAN AY		! : !	! : !	2	3B	059
060	3C	00111100	NO3 AX		< : <	< : <	Space	3C	060
061	3D	00111101	AND AX		= : =	= : =	Commodore	3D	061
062	3E	00111110	ROL AX		> : >	> : >	Q	3E	062
063	3F	00111111	RAN AX		? : ?	? : ?	RUN/STOP	3F	063
064	40	01000000	RTI		@ : @	- : -	Nincs bill.	40	064
065	41	01000001	EOR IX		a : A	◆ : A		41	065
066	42	01000010	ABS		b : B	: B		42	066
067	43	01000011	LSE IX		c : C	- : C		43	067
068	44	01000100	NO2 N		d : D	- : D		44	068
069	45	01000101	EOR N		e : E	- : E		45	069
070	46	01000110	LSR N		f : F	- : F		46	070
071	47	01000111	LSE N		g : G	: G		47	071
072	48	01001000	PHA		h : H	: H		48	072
073	49	01001001	EOR #		i : I	~ : I		49	073
074	4A	01001010	LSR BE		j : J	~ : J		4A	074
075	4B	01001011	ANL #		k : K	~ : K		4B	075
076	4C	01001100	JMP A		l : L	L : L		4C	076
077	4D	01001101	EOR A		m : M	/ : M		4D	077
078	4E	01001110	LSR A		n : N	/ : N		4E	078
079	4F	01001111	LSE A		o : O	┐ : O		4F	079
080	50	01010000	BVC R		p : P	┐ : P		50	080
081	51	01010001	EOR IY		q : Q	● : Q		51	081
082	52	01010010	ABS		r : R	- : R		52	082
083	53	01010011	LSE IY		s : S	♥ : S		53	083
084	54	01010100	NO2 NX		t : T	: T		54	084
085	55	01010101	EOR NX		u : U	┐ : U		55	085
086	56	01010110	LSR NX		v : V	X : V		56	086
087	57	01010111	LSE NX		w : W	o : W		57	087

Kódátszámítási táblázat

#	\$	%	Memo.	Token	ASCII	Screen	Dekóder	\$	#
088	58	01011000	CLI		x : I	♣ : X		58	088
089	59	01011001	EOR AY		y : Y	: Y		59	089
090	5A	01011010	NO1		z : Z	♦ : Z		5A	090
091	5B	01011011	LSE AY		[: [+ : +		5B	091
092	5C	01011100	NO3 AX		£ : £	£ : £		5C	092
093	5D	01011101	EOR AX] :]	:		5D	093
094	5E	01011110	LSR AX		↑ : ↑	⌘ : ⌘		5E	094
095	5F	01011111	LSE AX		+ : +	⌘ : ⌘		5F	095
096	60	01100000	RTS		- : -	SPACE		60	096
097	61	01100001	ADC IX		A : ♠	♠ : ♠		61	097
098	62	01100010	ABS		B :	■ : ■		62	098
099	63	01100011	RAD IX		C : -	- : -		63	099
100	64	01100100	NO2 N		D : -	- : -		64	100
101	65	01100101	ADC N		E : -	:		65	101
102	66	01100110	ROR N		F : -	⌘ : ⌘		66	102
103	67	01100111	RAD N		G :	:		67	103
104	68	01101000	PLA		H :	⌘ : ⌘		68	104
105	69	01101001	ADC #		I : ~	⌘ : ⌘		69	105
106	6A	01101010	ROR BE		J : ~	:		6A	106
107	6B	01101011	DAR #		K : ~	:		6B	107
108	6C	01101100	JMP I		L : L	⌘ : ⌘		6C	108
109	6D	01101101	ADC A		M : \	⌘ : ⌘		6D	109
110	6E	01101110	ROR A		N : /	⌘ : ⌘		6E	110
111	6F	01101111	RAD A		O : ⊥	- : -		6F	111
112	70	01110000	BVS R		P : ⊥	⌘ : ⌘		70	112
113	71	01110001	ADC IY		Q : ●	⌘ : ⌘		71	113
114	72	01110010	ABS		R : -	⌘ : ⌘		72	114
115	73	01110011	RAD IY		S : ♥	⌘ : ⌘		73	115
116	74	01110100	NO2 NX		T :	:		74	116
117	75	01110101	ADC NX		U : /	:		75	117
118	76	01110110	ROR NX		V : X	:		76	118
119	77	01110111	RAD NX		W : O	- : -		77	119
120	78	01111000	SEI		X : ♣	■ : ■		78	120
121	79	01111001	ADC AY		Y :	■ : ■		79	121
122	7A	01111010	NO1		Z : ♦	⌘ : ⌘		7A	122
123	7B	01111011	RAD AY		+ : +	■ : ■		7B	123
124	7C	01111100	NO3 AX		£ : £	■ : ■		7C	124
125	7D	01111101	ADC AX		:	⌘ : ⌘		7D	125
126	7E	01111110	ROR AX		⌘ : ⌘	■ : ■		7E	126
127	7F	01111111	RAD AX		⌘ : ⌘	⌘ : ⌘		7F	127
128	80	10000000	NO2 IX	END	Orange			80	128
129	81	10000001	STA IX	FOR				81	129
130	82	10000010	NO2 IX	NEXT				82	130
131	83	10000011	AXX IX	DATA				83	131
132	84	10000100	STY N	INPUT				84	132
133	85	10000101	STA N	INPUT#	F1			85	133
134	86	10000110	STX N	DIM	F3			86	134
135	87	10000111	AXR N	READ	F5			87	135

Kódátszámítási táblázat

#	\$	%	Memo.	Token	ASCII	Screen	Dekóder	\$	#
136	88	10001000	DEY	LET	F7			88	136
137	89	10001001	NO2 #	GOTO	F2			89	137
138	8A	10001010	TXA	RUN	F4			8A	138
139	8B	10001011	TAN #	IF	F6			8B	139
140	8C	10001100	STY A	RESTORE	F8			8C	140
141	8D	10001101	STA A	GOSUB	RETURN			8D	141
142	8E	10001110	STX A	RETURN				8E	142
143	8F	10001111	AAX A	REM				8F	143
144	90	10010000	BCC R	STOP	Black			90	144
145	91	10010001	STA IY	ON	CRSR up			91	145
146	92	10010010	ABS	WAIT	RVS off			92	146
147	93	10010011	AXI IY	LOAD	Clr			93	147
148	94	10010100	STY NX	SAVE	Inst			94	148
149	95	10010101	STA NX	VERIFY	Brown			95	149
150	96	10010110	STX NY	DEF	Lt.red			96	150
151	97	10010111	AXY NY	POKE	Grey1			97	151
152	98	10011000	TYA	PRINT#	Grey2			98	152
153	99	10011001	STA AY	PRINT	L.green			99	153
154	9A	10011010	TXS	CONT	Lt.blue			9A	154
155	9B	10011011	AXS AY	LIST	Grey3			9B	155
156	9C	10011100	YXA AX	CLR	Purple			9C	156
157	9D	10011101	STA AX	CMD	CRSR lt			9D	157
158	9E	10011110	XYA AY	SYS	Yellow			9E	158
159	9F	10011111	AYY AY	OPEN	Cyan			9F	159
160	A0	10100000	LDY #	CLOSE	SPACE			A0	160
161	A1	10100001	LDA IX	GET	█ : █			A1	161
162	A2	10100010	LDX #	NEW	■ : ■			A2	162
163	A3	10100011	LDT IX	TAB<	— : —			A3	163
164	A4	10100100	LDY N	TO	— : —			A4	164
165	A5	10100101	LDA N	FN	:			A5	165
166	A6	10100110	LDX N	SPC<	⌘ : ⌘			A6	166
167	A7	10100111	LDT N	THEN	:			A7	167
168	A8	10101000	TAY	NOT	⌘ : ⌘			A8	168
169	A9	10101001	LDA #	STEP	⌘ : ⌘			A9	169
170	AA	10101010	TAX	+	:			AA	170
171	AB	10101011	ANX #	-	:			AB	171
172	AC	10101100	LDY A	*	■ : ■			AC	172
173	AD	10101101	LDA A	/	L : L			AD	173
174	AE	10101110	LDX A	↑	⌘ : ⌘			AE	174
175	AF	10101111	LDT A	AND	— : —			AF	175
176	B0	10110000	BCS R	OR	r : r			B0	176
177	B1	10110001	LDA IY	<	⌘ : ⌘			B1	177
178	B2	10110010	ABS	=	T : T			B2	178
179	B3	10110011	LDT IY	>	⌘ : ⌘			B3	179
180	B4	10110100	LDY NX	SGN	:			B4	180
181	B5	10110101	LDA NX	INT	:			B5	181
182	B6	10110110	LDX NY	ABS	:			B6	182
183	B7	10110111	LDT NY	USR	— : —			B7	183

Kódatszámítási táblázat

#	\$	%	Mnemo.	Token	ASCII	Screen	Dekóder	\$	#
184	B8	10111000	CLV	FRE	— : —			B8	184
185	B9	10111001	LDA AY	POS	— : —			B9	185
186	BA	10111010	TSX	SQR	✓ : ┘			BA	186
187	BB	10111011	TSA AY	RND	■ : ■			BB	187
188	BC	10111100	LDY AX	LOG	■ : ■			BC	188
189	BD	10111101	LDA AX	EXP	┘ : ┘			BD	189
190	BE	10111110	LDX AY	COS	■ : ■			BE	190
191	BF	10111111	LDT AY	SIN	▣ : ▣			BF	191
192	C0	11000000	CPY #	TAN	— : —			C0	192
193	C1	11000001	CMP IX	ATN	A : ♠			C1	193
194	C2	11000010	NO2 IX	PEEK	B :			C2	194
195	C3	11000011	DEM IX	LEN	C : —			C3	195
196	C4	11000100	CPY N	STR\$	D : —			C4	196
197	C5	11000101	CMP N	VAL	E : —			C5	197
198	C6	11000110	DEC N	ASC	F : —			C6	198
199	C7	11000111	DEM N	CHR\$	G :			C7	199
200	C8	11001000	INY	LEFT\$	H :			C8	200
201	C9	11001001	CMP #	RIGHT\$	I : ~			C9	201
202	CA	11001010	DEX	MID\$	J : ~			CA	202
203	CB	11001011	XAS #	GO	K : ~			CB	203
204	CC	11001100	CPY A		L : L			CC	204
205	CD	11001101	CMP A		M : \			CD	205
206	CE	11001110	DEC A		N : /			CE	206
207	CF	11001111	DEM N		O : ▮			CF	207
208	D0	11010000	BNE R		P : 7			D0	208
209	D1	11010001	CMP IY		Q : ●			D1	209
210	D2	11010010	ABS		R : —			D2	210
211	D3	11010011	DEM IY		S : ♥			D3	211
212	D4	11010100	NO2 NX		T :			D4	212
213	D5	11010101	CMP NX		U : ~			D5	213
214	D6	11010110	DEC NX		V : X			D6	214
215	D7	11010111	DEM NX		W : O			D7	215
216	D8	11011000	CLD		X : ♣			D8	216
217	D9	11011001	CMP AY		Y :			D9	217
218	DA	11011010	NO1		Z : ♦			DA	218
219	DB	11011011	DEM AY		+ : +			DB	219
220	DC	11011100	NO3 AX		≡ : ≡			DC	220
221	DD	11011101	CMP AX		:			DD	221
222	DE	11011110	DEC AX		⊗ : 4			DE	222
223	DF	11011111	DEM AX		≡ : ▽			DF	223
224	E0	11100000	CPX #					E0	224
225	E1	11100001	SBC IX		■ : ■			E1	225
226	E2	11100010	NO2 IX		— : —			E2	226
227	E3	11100011	INB IX		— : —			E3	227
228	E4	11100100	CPX N		— : —			E4	228
229	E5	11100101	SBC N		:			E5	229
230	E6	11100110	INC N		≡ : ≡			E6	230
231	E7	11100111	INB N		:			E7	231

Kódszámítási táblázat

#	\$	%	Memo.	Token	ASCII	Screen	Dekóder	\$	#
232	E8	11101000	INX		⌘ : ⌘			E8	232
233	E9	11101001	SBC #		⌘ : ⌘			E9	233
234	EA	11101010	NOP		⌘ : ⌘			EA	234
235	EB	11101011	SBC #		⌘ : ⌘			EB	235
236	EC	11101100	CPX A		⌘ : ⌘			EC	236
237	ED	11101101	SBC A		⌘ : ⌘			ED	237
238	EE	11101110	INC A		⌘ : ⌘			EE	238
239	EF	11101111	INB A		⌘ : ⌘			EF	239
240	F0	11110000	BEQ R		⌘ : ⌘			F0	240
241	F1	11110001	SBC IY		⌘ : ⌘			F1	241
242	F2	11110010	ABS		⌘ : ⌘			F2	242
243	F3	11110011	INB IY		⌘ : ⌘			F3	243
244	F4	11110100	NO2 NX		⌘ : ⌘			F4	244
245	F5	11110101	SBC NX		⌘ : ⌘			F5	245
246	F6	11110110	INC NX		⌘ : ⌘			F6	246
247	F7	11110111	INB NX		⌘ : ⌘			F7	247
248	F8	11111000	SED		⌘ : ⌘			F8	248
249	F9	11111001	SBC AY		⌘ : ⌘			F9	249
250	FA	11111010	NO1		⌘ : ⌘			FA	250
251	FB	11111011	INB AY		⌘ : ⌘			FB	251
252	FC	11111100	NO3 AX		⌘ : ⌘			FC	252
253	FD	11111101	SBC AX		⌘ : ⌘			FD	253
254	FE	11111110	INC AX		⌘ : ⌘			FE	254
255	FF	11111111	INB AX	d	⌘ : ⌘			FF	255

Kislexikon

AC	Alternating Current, váltakozóáram.
AC	Accumulator, a processzor egyik munkaregisztere.
Access	Hozzáférés, egy adott tárcellához való hozzáférés.
Adat	Numerikusan vagy alfanumerikusan kódolt tény.
Adatbusz	A processzort a tárral és perifériáival összekötő adatsínek gyűjtőneve. Általában háromállapotú.
Adattár	Olyan tár, amely kizárólag adatokat tárol.
ADSR	Attack-Decay-Sustain-Release, a SID burkológörbe-generátora.
A/D	Analog/Digital, analóg-digitális átalakító.
AEC	Address Enable Control, címzésengedélyezés.
Akkumulátor	Munkaregiszter, amelyben az aritmetikai műveletek eredményei képződnek.
ALU	Arithmetic Logic Unit, az aritmetikai és logikai műveleteket végző egység.
AM	Address Manager, a C64 tárkezelő egysége.
AM/PM	Ante Meridiem/Post Meridiem, az angolszász nyelvterületen a délelőtt (AM) és a délután (PM) rövidítése.
ASCII	American Standard Code for Information Interchange, 256 tagú, szabványosított jelkészlet számítógépek számára.
Assembler	Programozási nyelv, a gépi kódú programozást támogatja.
ATN	Attention, a soros busz vezérlővonala.
BA	Bus Available, a busz használható.
BAM	Block Allocate Map, blokkfoglaltsági térkép.
BASIC	Egyfelől magas szintű programnyelv, másfelől a C64 belső ROM tárolóinak egyike.
Baud	A soros adatátvitel sebességének mértékegysége, bit/sec.
BCD	Binary Coded Decimal. Bináris kód decimális számjegyek számára.
Bináris	Kétértékű; igaz (igen), hamis (nem).
Bit	A bináris ábrázolás alapegysége.
Byte	8 bitből álló bináris adat.

CAM	Computer Aided Manufacturing, számítógéppel támogatott gyártás.
Carry	A processzor átvitelbitje.
CAS	Column Address Select. Oszlop kiválasztó címzőimpulzus dinamikus tá- raknál.
CBM	Commodore Business Machines, a Commodore cég teljes neve.
$\overline{\text{CE}}$	Chip Enable, az integrált áramkör engedélyező bemenete.
CHAREN	A C64 ROM tárolóinak egyike, a karaktergenerátor tárolója.
Chip	Integrált áramkör.
CIA	Complex Interface Adapter, a 6510-es processzor perifériakezelője.
Ciklusidő	Egyetlen utasítás feldolgozási ideje.
Cím	Tárak, illetve területek azonosítója.
Címbusz	A címzővezetékek gyűjtősíne. Általában háromállapotú.
CLK	Clock, a soros busz vezérlővonal.
Clock	Órajel, ütemjel.
CNT	Counter, számláló.
COLOR RAM	A VIC szintárolója, amely külön áramkörben van.
CPU	Central Processor Unit. Központi irányító és feldolgozó egység, mikro- processzor.
CP/M	Olyan szoftvercsomag, amely az egyébként erősen különböző mikroszá- mítógépek közötti kompatibilitást hivatott biztosítani. Többféle mikro- gépre is kidolgozták, így a C64-re is.
CRA	Control Register A; A vezérlőregiszter.
CRB	Control Register B; B vezérlőregiszter.
$\overline{\text{CS}}$	Chip Select, áramkör kiválasztó jel (több hasonló közül).
DATA	A soros busz adatvonal. Egyébként adatot jelöl.
DC	Direct Current, egyenáram.
Digitális	Számjegyekkel vezérelt, számokkal dolgozó.
DIL	Dual In Line, az integrált áramkörök tokozási megoldásainak egyike.
DIN	Deutsche Industrie Normen, német ipari szabványok, sok tekintetben világszabvány, számos ország szabványrendszere igazodik hozzá.
Dinamikus RAM	Olyan tár, amelyben az információt ciklikusan frissíteni kell.
$\overline{\text{DMA}}$	Direct Memory Access, közvetlen memóriáhozáférés.
DOS	Disk Operating System, a lemezegység vezérlőprogramja.
Dot Clock	Ütemjel a VIC számára, számos kimeneti frekvenciájának forrása.
EOT	End Of Tape, szalagvégjel.
EPROM	Erasable Programmable Read Only Memory. Ultraibolya fénnel töröl- hető, újraprogramozható fix tár.
Fényceruza	Olyan segédeszköz, amely lehetővé teszi, hogy a billentyűzet megkerü- lésével kapcsolatot tarthassunk a számítógépben futó programmal.
FIFO	First In First Out, olyan tár, amelyből az először bevitt adat olvasható ki elsőként, silótároló.

File	Háttértárolón elhelyezkedő adatállomány.
FLAG	A CIA chippek egyik handshake vonala.
FPLA	Field Programmable Logic Array, olyan logikai mátrix, amelyet a felhasználó programozhat.
GAP	Rés, adatállományok közötti definiálatlan, használaton kívüli terület.
GCR	Group Code Recording, a VC-1541 lemezegység tárolási rendszere.
GND	Ground, föld. Pozitív logikáknál a 0 V jelzése.
Handshake	Olyan kapcsolattartás esetén használják, amikor a két digitális egység két, esetleg több vezetéke felcserélve van összekötve.
Hardver	Digitális áramkör műszaki megvalósítása.
Header	Fejléc, bevezető, azonosító adatok.
Hexadecimális	16-os számrendszerbeli.
High, HI	Logikai magas szint. Pozitív logika esetén az 1 állapotnak felel meg.
HIRES	High Reselution, nagy felbontású képernyőszerkesztési üzemmód.
IC	Integrated Circuit, integrált áramkör.
ICR	Interrupt Control Register, a megszakítások ellenőrző-vezérlő regisztere.
IEC	International Electrical Commission, nemzetközi szabványbizottság, olyan termékek viselik nevét, amelyeket a bizottság elfogadott.
INPUT, IN	Bemenet, adatbemenet, beolvasás.
Interface	Illesztőegység, interfész. Olyan áramkör, amely lehetővé teszi két digitális egység között az adatforgalmat.
Interpreter	Értelmező programnyelv, a magas szintű programnyelvek működési formáinak egyike.
IRQ	Interrupt Request, maszkolható megszakításkérelem.
I/O	Input/Output, bemenet/kimenet. Kétirányú adatáramlás.
Joystick	Botkormány. Elsősorban játékok irányítására alkalmazzák.
Karakter	Olyan alfanumerikus vagy grafikus jel, amelyet egy ASCII-kód jellemez, nagyjából egy betű.
Kbyte, kB	Kilobyte, $2^{10} = 1024$ byte.
KERNAL	A C64 ROM tárolóinak egyike, az operációs rendszer tárolója.
Kompatíbilis	Egymással felcserélhető, egymásnak megfelelő, helyettesíthető.
Komplemens	A bináris aritmetika egyik kifejezése, egy adott bináris szám negatív előjelű megfelelője.
LIFO	Last In First Out, az utoljára bevitt adat olvasható ki először, zsáktároló.
Low, LO	Logikai alacsony szint, pozitív logika esetén a 0 állapotnak felel meg.
LP	Light Pen, lásd fényceruza.
LSB	Last Significant Bit (Byte), a legalacsonyabb bit (byte).
Mező	Adattárolási egység, a rekord része.
Mnemonik	Szimbólum. A gépi nyelv utasításainak összefoglaló neve.

Monitor	Egyrészt képernyős adatmegjelenítő, amely a televíziós adások vételére nem alkalmas. Másrészt a memóriában tárolt adatok olvasására alkalmas programot is így nevezik.
MOS	Metall Oxide Semiconductor, fénoxid alapú félvezető.
MPU	Microprocessor Unit, mikroprocesszor, lásd CPU.
MSB	Most Significant Bit (Byte), a legfőbb bit (byte).
Multicolor	Többszínű nagyfelbontású üzemmód.
Multiplexer	Többszöröző, több bemeneti adat jelenhet meg felváltva egy kimeneten.
NMI	Non Maskable Interrupt, nem maszkolható megszakításkérelem.
NTSC	National Television System Committee, amerikai videoszabvány, nevét az elfogadó bizottság nevének rövidítéséből kapta.
\overline{OE}	Output Enable, kimenetengedélyezés.
Órajel	Négyszögjel, amely a processzor ütemjele.
OUTPUT, OUT	Kimenet, adatkimenet, kiírás.
PAL	Phase Alternating Line, nyugat-európai videoszabvány.
PC	Program Counter, a processzor programszámláló regisztre.
\overline{PC}	Port Control, a CIA chipek egyik handshake vonala.
Periféria	Olyan külső (vagy belső) egység, amely feladatait többé-kevésbé önállóan végzi.
PLL	Phase Locked Loop, fázisban szabályzott ciklus. Olyan áramkör, amely pl. tetszőleges mértékű frekvenciaosztást képes szabályozni.
Port	Adatok fogadására vagy továbbítására alkalmas kapu vagy csatlakozó.
PPM	Pulse Position Modulation, a szalagos egység tárolási rendszere, impulzusok közvetlen tárolása.
Processzor	A digitális számítógép olyan része, amely vezérli és ellenőrzi a számítógép működését.
PS	Processor Status, a processzor állapotregisztere.
Puffer	Közbenső tárolásra kijelölt tár.
RAM	Random Access Memory, közvetlen hozzáférésű tároló.
RAS	Row Address Select, a dinamikus táruk sorcímző jele.
Rasztensor	Egy sor a televízió képernyőjén, de nem azonos a képernyősorral.
\overline{RDY}	Ready, a processzor jele, amely lehetővé teszi a VIC számára, hogy a rendszerbuszt használhassa akkor is, amikor egyébként a processzor használná.
Regiszter	Tárolóhely, amely megállapodás szerint állhat egy, illetve több byte-ból is.
Rekord	Adattárolási egység, egy címszó alá tartozó adatok.
RESET	A processzor bemenete; alacsony impulzus alapállapotba hozza a processzort.
RF	Radio Frequency, rádiófrekvencia, olyan jel, amelynek vivőfrekvenciája több száz, esetleg több ezer kilohertz.

ROM	Read Only Memory, csak olvasható fix tároló.
RS232	Soros adatátviteli rendszer. Eredetileg mérőműszerek számítógéphez való illesztésére alkalmazták, később azonban más viszonylatokban is elterjedt.
R/W	Read/Write, írás- vagy olvasásművelet kijelölése.
Scroll	Screen roll, a képernyő görgetése.
SDR	Serial Data Register, adatregiszter soros átvitelhez.
Serial	Soros, szekvenciális, egymás után következő.
Shift	Léptetés, eltolás.
SID	Sound Interface Device, három szólamú szintetizátor áramkör.
SP	Stackpointer, a processzor veremmutatója.
SP	Serial Port, olyan csatlakozó, amelyen az adatok sorban egymás után jelennek meg.
Sprite	A VIC speciális szolgáltatása. Külön programozható ábrát takar. Nevét magyarra fordítani rendkívül nehéz, mert találó elnevezés nincs, helyettesítése pedig az elterjedt idegen kifejezést nem pótolhatja.
Szoftver	Program, egyaránt jelölhet operációs rendszert és felhasználói programot.
Statikus RAM	Olyan táráramkör, amely beállítása után nem igényli az adatok ciklikus felfrissítését.
Time-sharing	Időosztás, a mai számítógépek többsége időosztásban hajtja végre többirányú feladatait.
TOD	Time Of Day, valós idejű óra.
Token	A BASIC utasítások nyolcbites kódja, azonosító.
TTL	Transistor Transistor Logic, tranzisztor-tranzisztor logika, általában +5 V tápfeszültséget igénylő, digitális logika.
USER	Felhasználói.
Ütemfrekvencia	A processzor órajelének frekvenciája. A processzor működési sebességére lehet belőle következtetni.
Verem	Olyan tároló, amelynek jellemző tulajdonsága, hogy az utoljára belekerült adat elsőként hagyja el. Lásd még LIFO.
VIA	Versatile Interface Adapter, perifériakezelő áramkör, a CIA elődje.
VIC	Video Interface Controller, a televíziós képet előállító áramkör.
VCO	Voltage Controlled Oscillator, feszültségvezérelt rezgőkör, olyan áramkör, amely rezgésszámát a bemeneti feszültség függvényében változtatni képes.
WE	Write Enable, írásengedélyezés.
XR	A processzor X indexregisztere, munkaregiszter.
YR	A processzor Y indexregisztere, munkaregiszter.
Z80	Processzor, a 8 bites processzorok egyik jellegzetes képviselője, a 6510 egyik piaci konkurense.

Irodalomjegyzék

1. *Gerd Thiele*: Félvezetős tárak. Műszaki Könyvkiadó, 1990
2. *Angerhausen — Brückmann — Englisch — Gerits*: A Commodore 64-es belső felépítése. Data Becker — Novotrade, 1985
3. *Angerhausen — Englisch — Gerits*: Tippek és trükkök a Commodore 64-esre. Data — Becker — Novotrade, 1985
4. *Brückmann*: A Commodore 64-es csatlakozási lehetőségei. Data Becker — Novotrade, 1988
5. *Englisch*: Gépi kódú programozás a Commodore 64-esen. Data Becker — Novotrade, 1985
6. *Dachsel*: Zenekönyv a Commodore 64-eshez. Data Becker — Novotrade, 1986
7. *Dr. Lengyel — Varga*: Lakat alatt. Novotrade, 1990
8. Commodore 64, Bedienungshandbuch Commodore
9. *Beuth*: Az elektronika alapjai I-III. Műszaki Könyvkiadó, 1990
10. *Obádovics — Hartyányi — Lengyel — Reményi*: Számítástechnika C64. Novotrade, 1989
11. *Farkas — Bálint*: Commodore 64 file-kezelés és input-output. LSI ATSZ, 1986
12. *Englisch — Szczepanowski*: A VC 1541-es lemezegység programozása. Data Becker — Novotrade, 1985
13. *Nick Hampshire*: Commodore 64 ROM's Revealed. Collins Professional and Technical Books, London, 1985
14. *Liesert*: PEEK-ek és POKE-ok a C64-esen. Data Becker — Novotrade, 1987
15. *H. H. Goldstine*: A számítógép — Pascaltól Neumannig. Műszaki Könyvkiadó, 1987

590 - Ft

A könyv címe, ami egyáltalán nem túlzás, mindent elmond a tartalomról. Az igen igényes szerző több, mint egy évtizedes tapasztalatait, valamint a C64-esről szóló könyvtárnyi irodalmának minden ellenőrzött ismeretanyagát egybegyűjtötte ebbe a kötetbe.

Az olvasó, ismereteitől függetlenül, immár egyetlen kötetben megtalálhatja az összes tudnivalót, amelyre a



hardverrel, szöfverrel, gépi kódú vagy BASIC programozással kapcsolatban szüksége lehet.

Az ismereteket gyorsan beírható és kipróbálható programpéldák illusztrálják.

A könyv az érdeklődőknek könnyen érthető, szakszerű, tömör és igen hasznos ismereteket ad, a profioknak pedig teljeskörű, rendszerezett referenciát, anélkül, hogy fárasztó magyarázatokba bocsátkozna.

Új könyveink

P. Norton
**Az IBM Pc és a PS/2
anatómiája**
ára: 899 Ft

S.B. Lipmann
C++ először
ára: 899 Ft

J.E. Hursch - C.J. Hursch
dBASEIV-SQL
ára: 690 Ft

Előkészületben

Dr. Hack Erigyes
**Turbo Pascal 6.0,
Turbo Vision**
ára: 590 Ft

R. Sluman
**WordPerfect 5.1
alapozó**
ára: 590 Ft

P. Rinearson
**Word 5.5
alapfoktól-mesterfokig
I.-II.-III.-IV.**
ára: 2490 Ft

**Novotrade Kiadó kft. és
Könyvmintaboltja
1119 Budapest, Albert u. 11.
Tel./fax.: 186-7318**